

**Etudiants : ZHANG Zhenyi
ZHANG Xiaowen**

Formation : ISI04 & ISI05

Type TPE : TX

**Enseignant responsable TPE :
LEMERCIER Marc**

**Tite sujet : Vulnérabilités de l'iPhone aux
vols de données personnelles**

Résumé :

Dans le cadre de ce projet TPEX, nous avons étudié la vulnérabilité des données personnels sur un iPhone en développant une application qui essaye de récupérer les données sensibles: les contacts, les calendriers, les données de localisations, les photos et les propriétés du dispositif.

Nous avons commencé ce projet par l'apprentissage des outils de conception et de développement dans le plateforme iOS. Ensuite, nous avons développé une application sur iPhone au fur et à mesure avec l'études des APIs dans ce système. En fin, nous avons proposé nos conseils pour la protection de données personnels.

Sommaire

Remerciement	4
I. Introduction.....	5
1. Contexte.....	5
2. Sujet	6
3. Problématiques.....	6
II. Apprentissage.....	8
1. Environnement de travail : Xcode	8
1) Initialisation de l'environnement.....	8
2) Utilisation de Xcode	8
2. Langage : Objective-C	11
1) Classes et Objets	11
2) Category	12
3) Attributs	13
4) Méthodes et appels.....	14
5) Fonctions utiles.....	14
3. Design Pattern : MVC	15
4. Les modèles classiques.....	17
5. Créer une Master-Detail application	18
1) Génération automatique des fichiers	19
2) Crédit de la couche modèle	20
3) Personnalisation de la scène principale.....	20
4) Affichage de données dans la scène secondaire	21
III. Conception	22
1. Prototypage	22
1) Outils.....	22
2) Ergonomie	24
IV. Réalisation d'un démonstrateur de collecte de données	26
1. Contacts	26
1) Objectif.....	26
2) Etude des APIs.....	26
3) Présentation de l'application	28
2. Calendriers & Rappels	28
1) Terminologie.....	28
2) Techniques.....	28
3) Description de l'application "CalSpy"	29
4) Description de l'application "RemSpy"	30
3. Photos.....	30
1) Objectif.....	30
2) Etudes des APIs.....	31
3) Description de l'application	32
4. Service de localisation	34
1) Objectif.....	34
2) Etudes des APIs.....	34
3) Description de l'application	38
5. Applications	39
1) Processus	39
2) Identifier les applications	40
3) Description de l'application "iHasApp"	44
4) Analyser l'utilisation des applications	45

5) Description de l'application "Addictions"	48
6. Device	49
1) Objectif.....	49
2) Etude des APIs.....	49
3) Description de l'application.....	50
7. Autres	51
1) Données non accessibles.....	51
2) SandBox.....	53
V. Bilan de protection.....	55
1. Confidentialité dans IOS 6.0.....	55
2. Guide directive d'Apple.....	57
3. Conseils de protection	58
VI. Conclusion	59
VII. Bibliographie	60
VIII. ANNEXES.....	64
1. Annexe 1 : Métadonnées d'une photo.....	64
2. Annexe 2: Liste des attributs des processus	66

Remerciement

Tout d'abord, nous tenons à remercier M. Marc LEMERCIER, responsable du branche ISI de l'Université de Technologie de Troyes, qui nous a encadré, guidé et enseigné pendant ce projet TPEX. Nous le remercions également pour ses aides et ses conseils pendant l'édition de ce rapport.

Nous souhaitons remercier également M. Michel DOUSSOT, responsable de filière TMSE de l'UTT, pour ses aides et ses conseils sur les questions techniques lors du développement de notre application.

I. Introduction

1. Contexte

Avec le développement rapide de l'Internet Mobile, le Smartphone devient un outil indispensable dans notre vie quotidienne et contient de plus en plus d'informations sur son propriétaire. Pourtant, contrairement à un ordinateur, nous sécurisons très peu, voire pas du tout, son accès à nos données privées et personnelles. Une récente étude de la CNIL (Commission Nationale de l'Informatique et des Libertés) et Médiamétrie révèle que 40% des possesseurs de Smartphone stockent des données à caractère sensible (coordonnées bancaires 7%, codes secrets 17%, codes d'accès aux immeubles 17%, informations médicales 3%) dans leur téléphone. Et 30% d'entre eux déclarent n'avoir aucun code de protection actif sur leur téléphone [2012, CNIL].

Cette faible protection donne aux « spywares » une opportunité de collecter illégalement des données personnelles. Certaines, voire beaucoup, d'applications pour Smartphone traitent des données personnelles à la légère. Au début de février 2012, un développeur Singapourien a découvert qu'une application iPhone du réseau social « Path » transmettait le carnet d'adresse complet sur ses serveurs à distance sans même les prévenir [2012, MCLOV]. En fait, beaucoup d'autres applications populaires sur la plate-forme iOS, comme Twitter, Foursquare et Instagram, collectent aussi les informations du carnet d'adresse vers leurs services pour tenter de retrouver des relations entre les utilisateurs déjà présents sur le réseau. Pourtant, on ne sait jamais si ces informations sont aussi transmises à un tiers. Le cas de Twitter est encore plus grave parce qu'il conserve des données pour une durée de plus que 18 mois. Les données sensibles comme le carnet d'adresse, le calendrier, les photos ainsi que les données géographiques ouvrent la porte aux cyber-criminels.

Depuis quelques mois, Apple est pointé au doigt par les médias et les experts de sécurité car ses protections sont insuffisants pour éviter que les applications malveillantes passent les barrières de la validation de l'AppStore. C'est pour cela qu'Apple a mis en place une fonctionnalité de protection des données personnelles à partir de iOS6. Ainsi, dans l'application « Réglages » sur iPhone, il y a un nouveau champ « Confidentialité » dans lequel l'utilisateur peut contrôler le droit d'accès à ses données application par application. De plus, avant que les applications essayent d'accéder à nos données, il y aura une demande de confirmation qui nous précise le service qu'elles ont besoin. Cette fonctionnalité est sans doute une amélioration de la sécurité de nos Smartphones iOS. Pourtant, l'accès aux contacts ou au calendrier est une des démarches indispensables pour beaucoup d'applications. La plupart des utilisateurs autorisent leurs accès sans même réfléchir.

Certaines organisations comme la CNIL ont commencé à notifier les utilisateurs des dangers liés à leurs données personnelles présentes sur les terminaux nomades et leur propose des conseils de protection. Mais, la conscience de la protection des données sur les Smartphones reste encore très faible.

2. Sujet

Dans le cadre d'un projet personnel encadré, nous voulons montrer la vulnérabilité des données personnelles d'iPhone en essayant de voler les informations de la même façon que le font les applications malicieuses. Les données critiques peuvent être catégorisées dans les trois types suivants :

- Les enregistrements dans les applications intégrés fournis par Apple : les contacts, les événements du calendrier, les rappels et les photos;
- Les fonctionnalités fondamentales d'un téléphone mobile : les SMS, les appels et même les mails;
- Non seulement les enregistrements nous intéressent, les traces des activités d'utilisateur sont également importantes : les traces de mobilité avec le suivi du GPS et des statistiques sur l'utilisation des applications via le registre des processus.

Nous allons simuler un scénario simple de vol en commençant par récupérer les données pour les afficher sur l'écran, et ensuite les envoyer vers un serveur web. Les questions suivantes nous intéressent :

- Est-ce qu'il est possible d'accéder à ces données ?
- Les utilisateurs seront-ils au courant ce qui se passe dans le scénario ? Seront-ils notifiés quand des applications voudront accéder à ses données et les envoyer ? Peuvent-ils interrompre le processus ?

Après ces études, nous allons essayer de donner des conseils de protection sur les données privées en approfondissant les politiques de confidentialité d'iOS 6.

3. Problématiques

Afin de réussir à étudier le sujet, il faut passer par trois étapes : l'apprentissage, la conception et la réalisation.

Tout d'abord, il faut apprendre à développer des applications d'iPhone. Même si il n'y a pas besoin de comprendre tous les fonctionnements et les techniques de programmation, des connaissances sur les modèles classiques sont indispensables. L'environnement de travail est Xcode qui est exclusif pour le développement des applications d'iOS et de Mac OS. Xcode est un outil puissant avec son éditeur de code, son outil de construction d'interfaces graphiques et son simulateur pour tester. Il facilite le développement en fournissant plein de raccourcis. Le langage utilisé est également spécial : Objective-C qui est un langage orienté objet mais avec des syntaxes très différents des langages classiques (Java, C++, etc.). Il faut donc non seulement réviser les conceptions de programmation orienté objet, mais également apprendre les particularités d'Objective-C. À part le langage, le modèle de conception MVC est également critique pour réussir à développer une application d'iOS. Nous avons donc approfondi notre compréhension sur ce modèle. Apple fournit aux développeurs un certain nombre de démonstrations. Des pratiques sur ces programmes nous aident à établir un flux ordinaire pour développer une application.

Avant de commencer à étudier les techniques pour la réalisation, il faut d'abord concevoir ce que nous allons faire. Vu que les fonctionnalités sont relativement simples dans notre cas, nous avons mis l'accent sur les méthodes et les outils. Les maquettes nous aident à établir une vue plus concrète sur le projet. Afin de mettre en place ces deux méthodes, plusieurs outils subsidiaires sont disponibles. Ils peuvent être utilisés selon les différents besoins. Par contre, parce que la conception d'interface graphique d'applications iPhone est différente de celle d'une page web, nous avons donc effectué une étude de plus sur l'ergonomie.

Enfin, nous nous sommes focalisés sur les techniques d'obtention des différentes données. Certaines données sont accessibles depuis les interfaces existantes fournies par Apple, il faut donc consulter les APIs. Certaines n'ont pas de méthodes publiques pour y accéder, nous allons faire des recherches pour savoir s'il existe d'autres façons de collecter des données. Pour les traces des utilisateurs, nous pensons profiter du mécanisme d'exécution en l'arrière-plan afin de collecter de temps en temps les informations sur les utilisateurs. Après avoir répondu à toutes ces problématiques, nous pourrons comprendre les vulnérabilités associées aux vols de données personnelles sur iPhone et proposer des méthodes de protection.

II. Apprentissage

1. Environnement de travail : Xcode

1) Initialisation de l'environnement

Le développement de l'application iPhone se fait en langage Objective-C, un langage qui nécessite l'utilisation de Xcode. Xcode est un environnement de développement dans Mac OS X. Un Mac est donc un outil indispensable pour le développement des applications. Une fois nous avons un Mac à disposition, nous pouvons télécharger gratuitement Xcode sur [App Store](#). Logiciel Xcode contient l'ensemble des SDKs (kit de développement) nécessaires, quelques outils pour le test et un simulateur d'iOS. Après avoir installé Xcode, il suffit de l'ouvrir et créer le premier projet.

Le simulateur iOS nous permet de tester notre application sur le Mac. Par contre, si nous voulons exploiter nos applications sur un vrai dispositif, il faut inscrire dans le programme de développeur de l'Apple, qui coûte 79 euro par an. Avec ce compte, nous avons aussi le droit d'avoir des versions beta de système iOS et distribuer nos applications sur l'App Store.

2) Utilisation de Xcode

Xcode est un environnement de développement ressemble à la plupart de IDE comme Eclipse et Netbeans. Dans Xcode, nous pouvons créer une application à partir du zéro. Des outils divers dans Xcode nous permet également de gagner du temps pour les activités liée aux développement et au test :

- Interface Builder : C'est un outil qui nous permet de créer un « Storyboard » (enchainement des fenêtres) pour une application. Il suffit de glisser-déposer des composants graphiques sur la toile et configurer la relation entre le code et les composants par un click droit.
- Instruments : C'est un outil qui peut nous aider à analyser une application à partir de plusieurs aspects. Par exemple, nous pouvons l'utiliser à tester une fuite de mémoire, à détecter le temps de calcul de chaque opération et à simuler les actions de l'utilisateur.
- Unit Test : Xcode comporte aussi une fonctionnalité de test. Nous pouvons créer des classes de test similaire à celles crée pour JUnit. Le test de l'interface graphique se fait avec un outil dans les « Instruments ».
- Contrôle de source : Un système de Git est intégré dans Xcode. Nous pouvons gérer les répertoires et les branches de notre projet par une interface graphique.

a) Point de départ



Image II.1.a : Interface d'accueil de Xcode

Après avoir ouvert Xcode, une fenêtre d'accueil va apparaître. A gauche, il y a des buttons qui nous permet de créer un nouveau projet ou se connecter avec un répertoire de Git à distance. A droite, il y une liste de projets qui sont récemment ouverts par utilisateur. Nous pouvons aussi ouvrir d'autres projets à partir du bouton « Open Other... », qui est situé en bas à gauche.

b) Interface principale

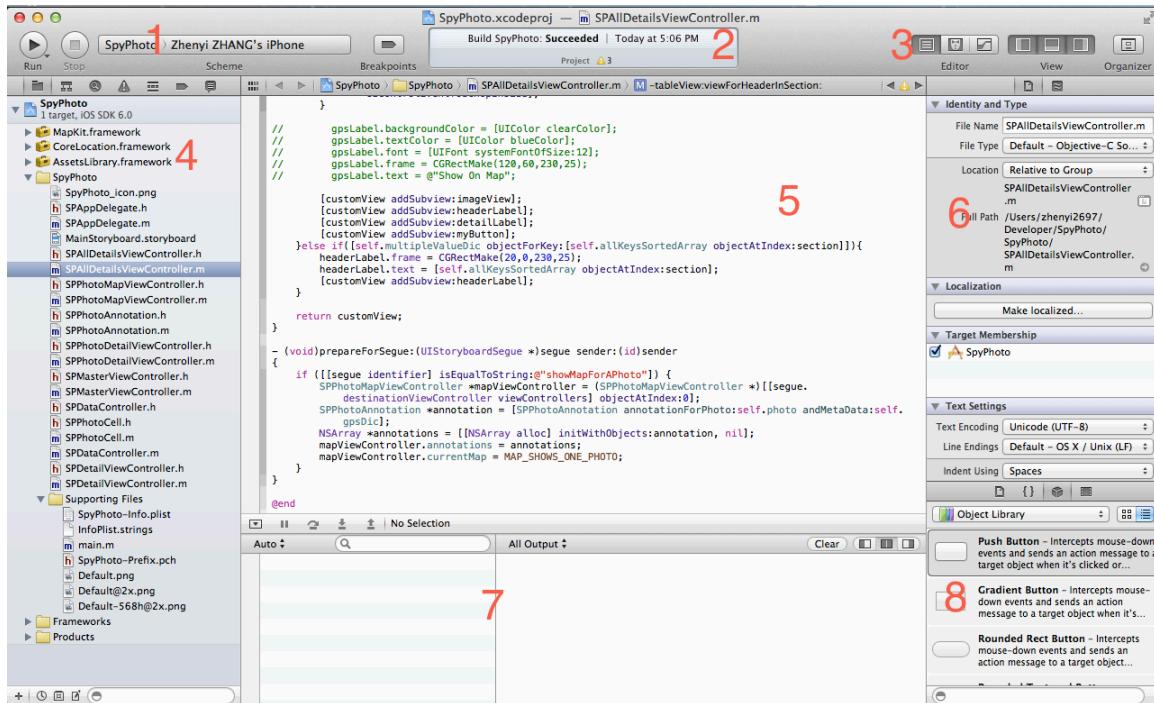


Image II.1.b : Interface principale de XCode

Interface principale de Xcode est composé de plusieurs parties:

1. Les boutons à gauche en haut nous permettent de lancer l'application et de choisir le support d'exécution entre le simulateur et les vrais dispositifs.
2. Le barre d'affichage en haut au milieu va afficher les informations liées à l'exécution du programme, y compris l'heure de compilation, nombre d'erreurs, résultat de l'exécution, etc.
3. Toutes les parties dans cette interface sont réglables. Les boutons en haut à droite nous permettent d'afficher ou de cacher ces parties. Par exemple, si nous voulons plus de place pour le code, nous pouvons cacher toutes les parties sur le bord.
4. L'arborescence à gauche représente l'ensemble des fichiers dans ce projet. Nous pouvons y accéder par un simple click.
5. Zone d'édition principale. Lors que le fichier « Storyboard » est choisi, cette partie va devenir une toile sur laquelle nous pouvons « dessiner » les scènes de notre application (voir image ci-dessous).
6. La partie à droite va afficher toutes les propriétés de l'élément en cours d'édition. Dans l'Interface graphique, elle va afficher les propriétés de l'élément choisi.
7. Console et debugger.
8. Cette petite fenêtre contient les composants graphiques disponibles dans la librairie Cocoa. Nous pouvons les ajouter dans notre application par un coup de glisser-déposer.

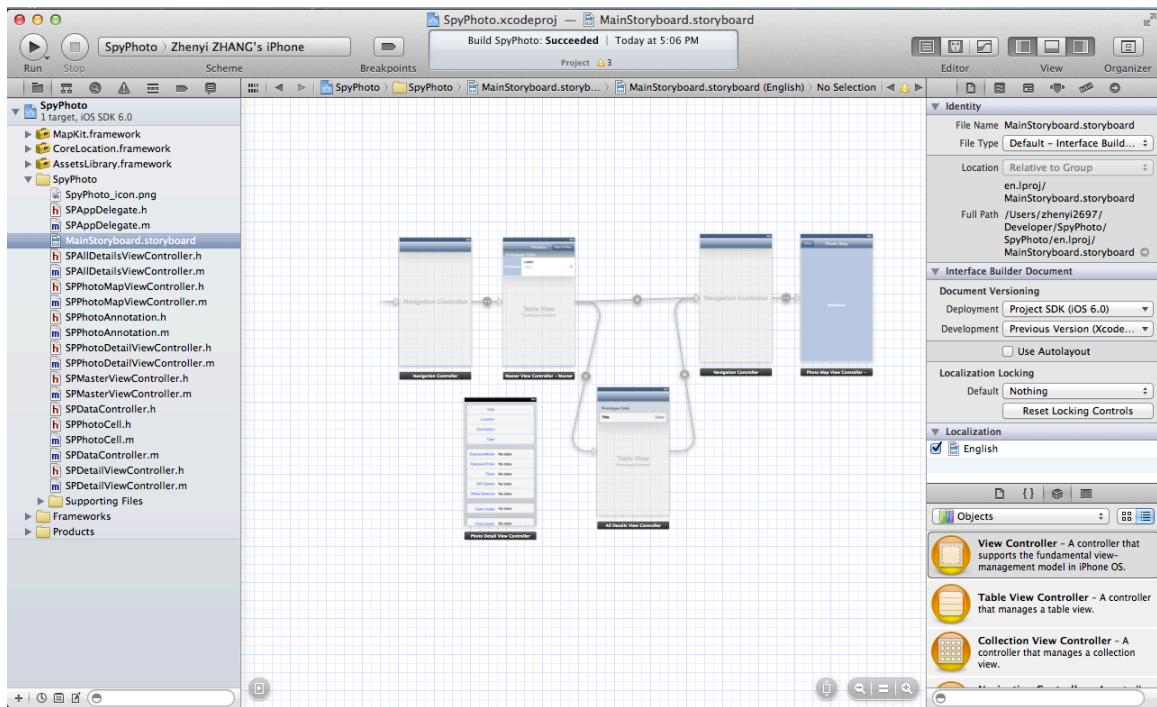


Image II.1.c : « Interface Builder » dans XCode

2. Langage : Objective-C

Le langage qu'Apple a choisi pour la programmation des applications iPhone est Objective-C. Il n'est pas exclusif pour la programmation d'iOS. En effet, il est indispensable pour le développement de presque tous les produits d'Apple, Mac OS, par exemple. Objective-C est un langage orienté objet. Il partage donc beaucoup d'éléments avec Java et C++. Par contre, les types de données sont différents d'autres langages. Ils commencent par NS et possèdent beaucoup d'APIs utiles comme NSString. Il faut consulter les documentations pour en savoir plus.

1) Classes et Objets

Une classe dans Objective-C est spécifiée dans deux fichiers : un fichier .h qui décrit l'interface et un autre fichier .m qui contient les implémentations.

L'importation des frameworks et la déclaration des propriétés, des variables et des méthodes doivent apparaître dans l'interface. Les classes doivent hériter d'une et seulement d'une classe, par défaut, c'est la classe NSObject. Les classes peuvent implémenter plusieurs interfaces par le syntaxe : <MyProtocol>.

L'implémentation d'une classe doit posséder toutes les méthodes déclarées dans l'interface et les méthodes non optionnelles des protocoles implémentés. Evidemment, elle peut contenir les fonctions privées qui seront utilisées à l'intérieur de la classe.

```

#import <Gizmo/Gizmo.h> //importation des frameworks
    //déclaration l'interface de la classe
@interface MyClass : NSObject <MyProtocole> {
    int *count
}
    //déclaration des attributs de la classe
@property (copy) MyModelObject *theObject;
@property (readonly) NSView *rootView;
@property (weak) id delegate;
@property (nonatomic, copy) NSString *userName;
    //déclaration des méthodes
- (id)initWithString: (NSString *)aName;
+ (MyClass *)createMyClassWithString: (NSString *)aName;
@end

    //implémentation de la classe
@implementation
@synthesize theObject = _theObject;
@synthesize rootView = _rootView, userName = _userName;
@synthesize delegate = _delegate;
- (void)setUserName:(NSString *)newUserName {
    _userName = newUserName;
    //d'autres instructions;
}
- (id)initWithString:(NSString *)aName {
    self.userName = aName;
    //équivalent à [self setUserName : aName];
}
+ (MyClass *)myClassWithString:(NSString *)aName {
    //codes
}
@end

```

Code II.2.a: Example d'une classe en Objective-C

2) Category

Dans Objective-C, une « Category » nous permet d'ajouter des méthodes à une classe existante, même à celle pour laquelle nous n'avons pas la source. La

« Category » est une fonctionnalité puissante qui nous permet d'étendre les fonctionnalités des classes existantes sans l'héritage. Par exemple, si nous voulons simplement avoir un « String » qui gère la compression, au lieu d'utiliser l'héritage, nous pouvons juste créer une « Category » qui ajoute la compression à la classe « String » sans toucher les codes existants. En utilisant les « Category », nous pouvons également distribuer la mise en œuvre de nos propres classes entre plusieurs fichiers. Par contre, avec « Category », nous ne pouvons ajouter des attributs à une classe existante. C'est aussi une différence majeure entre la « Category » et l'héritage.

La création d'une « Category » est similaire à celle d'une classe : la déclaration dans un fichier .h et l'implémentation dans un fichier .m. Par exemple, nous pouvons ajouter une méthode « `numberOfIntegers` » à la classe « `NSArray` », qui est une classe prédéfinie dans la librairie « `Cocoa` ».

```
Dans le fichier « NSArray+Extension.h » :
#import <Foundation/Foundation.h>
@interface NSArray (Extension)
-(int)numberOfIntegers;
@end

Dans le fichier « NSArray+Extension.m » :
#import "NSArray+Extension.h"
@implementation NSArray (Extension)
-(int)numberOfIntegers
{
    //Implementations
    return 0;
}
@end
```

Code II.2.b: Création d'une category pour la classe « `NSArray` »

3) Attributs

Les attributs de classe sont déclarés suivant le syntaxe `@property`. Entre les parenthèses, on peut préciser les options : `copy` qui signifie que l'objet sera copié lors de l'affectation ; `readonly` ne permet pas la définition de setter pour cet attribut ; `strong` déclare une référence forte, c'est-à-dire que son espace sera libéré uniquement quand il n'y plus de pointeur vers lui ; `weak` déclare une référence plus faible, dès qu'il n'y a plus de pointeur fort vers lui, il sera libéré quelque soient le nombre de références faibles existants ; La dernière option est

`nonatomic` qui peut être appliquée dans la plupart des cas afin d'améliorer l'efficacité lors de la synchronisation de multithread.

Xcode génère automatiquement les setters et getters avec l'instruction `@synthesize`. Les attributs de l'objet peuvent être accédés et modifiés par un appel de type `object.attribut`. Réécrire des setters et getters est également possible en utilisant la variable `nomDAttribut`.

4) Méthodes et appels

Les méthodes sont déclarées suivant un minus - ou un plus + qui représentent respectivement les méthodes d'instance et les méthodes statiques. L'écriture des noms de méthodes est différente avec Objective-C qui définit les méthodes séparés par les paramètres d'entrées. Le but est de rendre la lecture des méthodes plus proche de l'anglais parler. Par exemple :

```
- (void)insertObject:(id)anObject atIndex:(NSInteger)index;
```

Code II.2.c: déclaration d'une méthode

Dans la première parathèse, c'est le type de valeur de retour, void signifie que c'est une méthode sans valeur de retour; `anObject` et `index` sont tous des paramètres d'entrée de cette méthode.

L'appel de méthodes dans Objective-C suit la même principe : l'habitude de communication humaine en anglais. Par exemple, la méthode ci-dessous peut être lue comme "contacts, ajouter l'objet person pour l'indice 5":

```
[contacts insertObject:person atIndex:5];
```

Code II.2.d: Appel d'une méthode

5) Fonctions utiles

➤ Débogage

```
NSLog("Personne : %@", personne);
```

Code II.2.e : Impression de l'information dans la console

➤ Traitement de chaînes de caractères

Il y a deux types de chaînes de caractères dans Objective-C : `NSString` et `NSMutableString`. `NSString` est immutable, c'est-à-dire qu'une fois la valeur d'une instance est affectée, il ne peut plus être modifiée (définition d'une constante). C'est pour ça que l'on utilise souvent `NSMutableString` dans une fonction mais les valeurs de retour sont de `NSString`.

```
[NSString stringWithFormat:@"personne No%d : %@", 1, personne];
```

Code II.2.f : Example de « NSString»

➤ Traitement de tableaux

Les deux classes les plus utilisées sont NSDictionary et NSArray. NSArray représente les tableaux et NSDictionary contient les couples de clés-valeurs. Il existe également les versions mutables pour ces deux classes.

```
//{"a", "b", "c"}  
NSArray *tab = [NSArray arrayWithObjects: @"a", @"b", @"c", nil];  
[tab objectAtIndex: 2];  
  
//{"a"=>"1", "b"=>"2", "c"=>3}  
NSDictionary *dict = [NSDictionary dictionaryWithObjects:[NSArray  
arrayWithObjects:@"1", @"2",[NSNumber numberWithInt:3], nil]  
forKeys:[NSArray arrayWithObjects:@"a",@"b",@"c", nil]];  
[dict objectForKey: @"b"];
```

Code II.2.g : Example de « NSArray» & de « NSDictionary»

➤ Traitement de date

NSDate contient toutes les informations concernant une date et heure. Souvent, on a beoin de l'interpréter sous une chaîne de caractères.

```
//now  
NSDate *now = [NSDate date];  
  
//toString  
NSDateFormatter *formatter = [[NSDateFormatter alloc] init];  
//Il existe plusieurs formats prédéfinis, on peut également le  
définir personnellement  
[formatter setDateFormat:@"dd-MM-YYYY HH:mm"];  
[formatter stringFromDate: now];
```

➤ Code II.2.h : Example de « NSDate»

3. Design Pattern : MVC

Le pattern MVC (Modèle-Vue-Controleur) est utilisé partout dans iPhone. De façon plus générale, il est le cœur de Cocoa. D'après les documentations d'Apple, la conception et le développement de toutes les applications dans iOS doivent suivre ce pattern pour avoir une structure propre et parfaite.

Plus concrètement, dans iOS, le pattern MVC représente des termes suivants :

- Modèle : Le modèle contient des données centrales et les logiques qui manipulent ces données. Idéalement, il n'y a pas de connexion entre le modèle et la vue. Par exemple, si on veut créer une application qui enregistre les informations d'une personne. Le modèle peut être une classe « Personne » qui contient le nom, le prénom et la date de naissance. Ces données sont réutilisables pour les interfaces différentes.
- Vue : la vue est une couche qui interagisse avec l'utilisateur. Dans iOS, chaque vue est représentée par une scène. L'ensemble des scènes (fenêtres dans l'application) constitue le fichier « Storyboard ». Dans XCode, on peut personnaliser l'interface de chaque scène à l'aide d'un outil graphique. L'enchaînement des scènes est réalisé par des liaisons « segue » (relation entre deux scènes).
- Contrôleur : Un objet contrôleur est une couche intermédiaire qui relie les objets de la vue et les objets du modèle. Ils sont souvent chargés de s'assurer que les vues ont accès aux objets du modèle quand elles ont besoin d'afficher des données. Ils travaillent aussi comme un coordinateur qui gère le cycle de vie des autres objets. Par exemple, le contrôleur de type « Navigation Controller » gère la navigation entre différentes scènes. Le contrôleur de type « TableViewController » définit le contenu de chaque cellule dans la liste.

La relation entre les objets de vue et les contrôleurs est réalisée par deux façons. Pour les éléments d'affichage, comme le label et l'image, on définit dans le contrôleur un attribut de type « IBOutlet ». On envoie comme paramètre l'objet de l'élément graphique au contrôleur pour que le contrôleur puisse accéder à ses attributs. Par exemple, dans l'en-tête un « ViewController », on peut définir un élément d'affichage :

```
@property (weak, nonatomic) IBOutlet UILabel *display;
```

Code II.3.a : Définition d'un « IBOutlet »

Pour afficher un contenu sur cet élément, il suffit de changer un attribut « text » de l'objet « display » :

```
self.display.text = @"Hello World!";
```

Code II.3.b : Affichage de l'information sur l'écran

Le deuxième type de liaison s'appelle « IBAction ». On peut définir les opérations à réaliser suivant une action de l'utilisateur. Par exemple, une fois que nous avons défini un bouton à la scène, nous pouvons ajouter une méthode qui renvoie une « IBAction » dans le contrôleur correspond :

```
(IBAction) ditBonjour: (UIButton *) sender;
```

Code II.3.c: Définition d'une « IBAction »

Le « sender » dans la définition ci-dessus représente le bouton sur lequel l'utilisateur vient d'appuyer. On peut ensuite définir une opération dans cette méthode. La méthode ci-dessous va afficher le texte sur le bouton appuyé dans la console :

```
- (IBAction) ditBonjour(UIButton *) sender {  
    NSLog(@"%@", was pressed., sender.text);  
}
```

Code II.3.d: Opération suivant l'appui d'un bouton

4. Les modèles classiques

A la création d'un projet, Xcode demande de choisir un modèle pour l'application. Les trois types qui sont utilisé souvent sont : Master-Detail Application, Paged-Based Application et Tabbed Application. Les trois modèles s'adaptent aux différents types d'applications que l'on veut créer et ils contiennent par défaut quelques vues et interactions entre eux.

- Master-Detail : L'application de ce type nous permet d'afficher une liste d'élément dans une scène principale et leurs détails dans une scène secondaire. C'est un style typique dans iPhone dont beaucoup d'applications, comme « Contacts », « Messages » et « Mail » ont utilisé. L'intérêt de ce type d'application c'est de pouvoir naviguer entre plusieurs scènes dans lesquelles on peut présenter des contenus sous forme hiérarchique.
- Pages-Based Application : Ce modèle fonctionne comme un livre qui contient plusieurs pages. Les contenus dans chaque page peuvent être personnalisés librement en implémentant le contrôleur de vue créé par défaut. A la création du projet, un délégué est initialisé pour répondre aux interactions des utilisateurs, par exemple, tourner une page.
- Tabbed Applications : Les applications basées d'onglet peuvent être trouvés dans iTunes, Horloge et Téléphone qui sont intégrées dans l'iPhone. Il possède souvent quatre onglets et chacun fournit une fonctionnalité principale. C'est très facile de changer de page sans retourner au menu principal quand il n'y que peu de fonctionnalités. Xcode initialise un contrôleur des onglets et deux pages exemples à la création d'un projet de ce type.

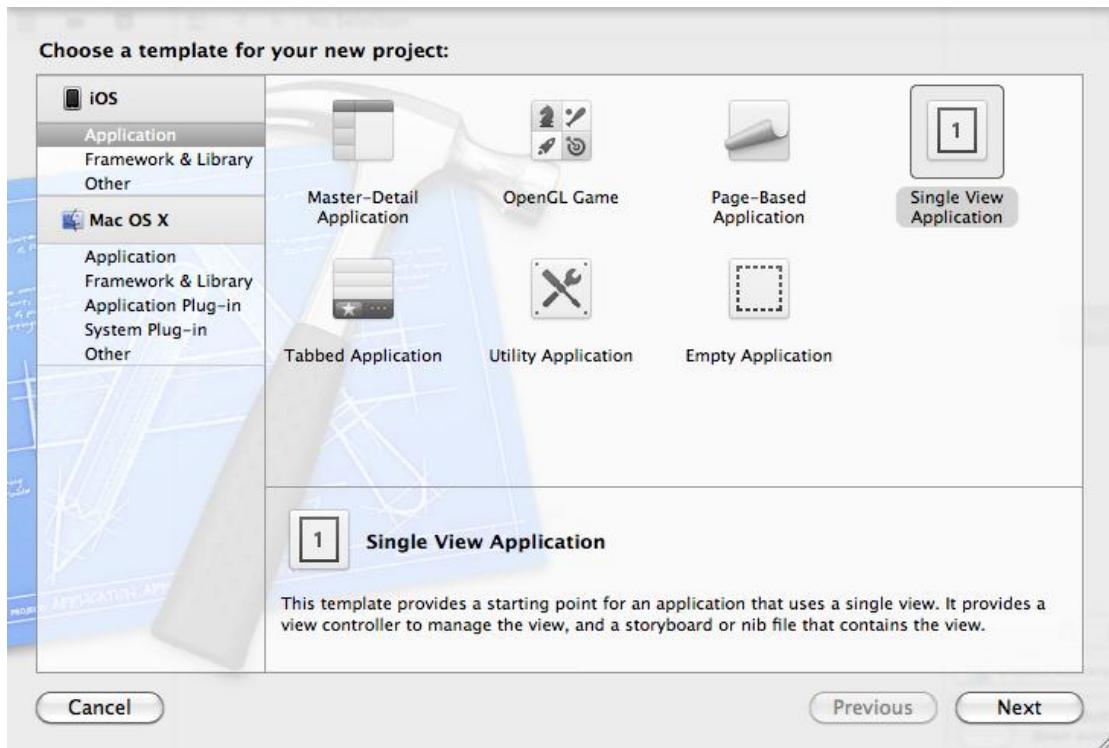


Image II.4.a : Différents types d'applications dans Xcode

Dans le cadre de ce projet, les applications sont basées sur les données de même structure. Il faut les afficher clairement dans une liste et offrir la possibilité d'accéder à leurs détails. C'est le modèle Master-Detail qui est plus cohérent avec le sujet.

5. Créer une Master-Detail application

Pour créer une application « Master-Detail View », on doit commencer par créer un nouveau projet dans XCode et choisir le style « Master-Detail Application ». Le schéma ci-dessous représente une démarche globale de la création d'un projet.

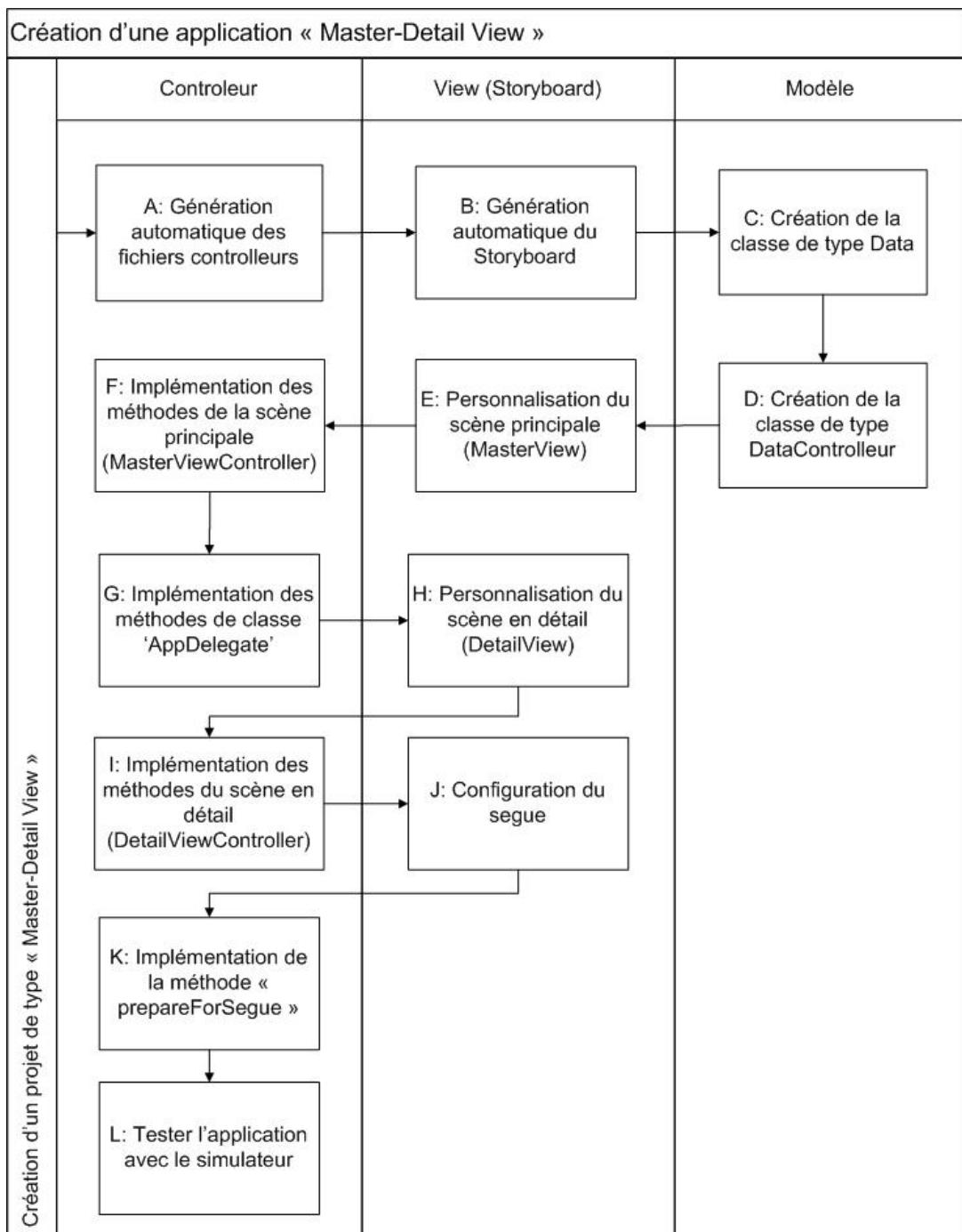


Image II.5.a : Crédit à la création d'une « Master-Detail » application

1) Génération automatique des fichiers

- A. Après la création d'un projet « brut », XCode va générer deux contrôleurs : « MasterViewController » et « DetailViewController » qui sont chargés respectivement de gérer la scène principale et secondaire. Elle génère aussi deux fichiers « AppDelegate » qui sont chargés de configurer le démarrage de l'application.

B. Le fichier d'interface « Storyboard » est également généré automatiquement. Comme nous voyons dans l'image ci-dessous, une « TableView » est embarqué dans la scène principale (Master View). Dans la scène secondaire, par contre, il y a une vue avec un label pour afficher le détail de l'élément choisi. La mission de « NavigationController » qui se trouve à gauche est de gérer la navigation entre les différentes scènes.

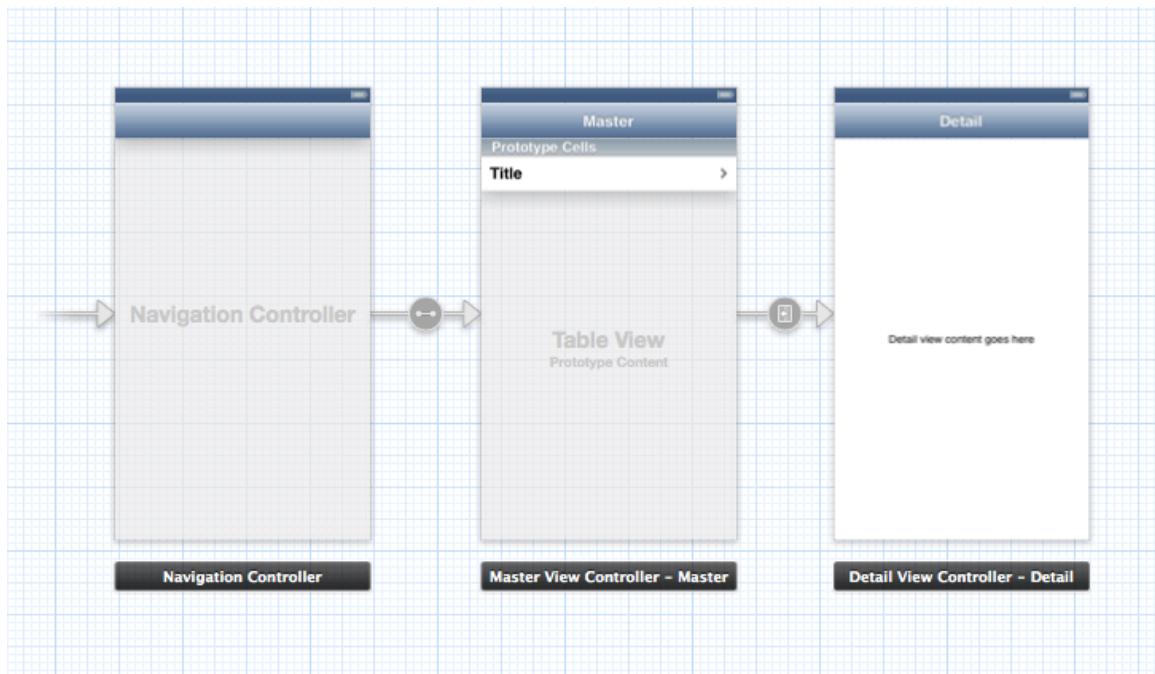


Image II.5.b : Création de « Storyboard »

2) Crédation de la couche modèle

- C. Pour la plupart des applications, nous avons besoin une classe qui contienne les données. D'après le pattern MVC, ces données sont dans la couche « Modèle » et sont donc isolées de « Vue » et de « Contrôleurs ».
- D. Nous avons aussi besoin de créer une classe « DataController » qui gère la communication entre le modèle et les contrôleurs de vue. Sa mission est de préparer des données nécessaire pour leur présentation à l'écran. Par exemple, on peut créer un attribut dans la classe « MasterViewController » qui est de type « DataController ». Le « MasterViewController » va communiquer avec le « DataController » pour récupérer des données à afficher dans la scène principale.

3) Personnalisation de la scène principale

- E. Dans cette étape, on doit personnaliser le prototype de cellule utilisé dans la liste de la scène principale.
- F. Dans le « MasterViewController.m », il y a deux méthodes importantes : la méthode « numberOfRowsInSection » qui renvoie le nombre d'élément dans la liste, et la méthode « tableView: cellForRowAtIndexPath:indexPath » qui crée une nouvelle cellule à partir du prototype et la remplit avec des données.

G. Dans le fichier « AppDelegate.m », on doit lier le « MasterViewController » et le « DataController » pour que les deux peuvent se communiquent. Cette opération est réalisée dans la méthode « application : didFinishLaunchingWithOptions ».

4) Affichage de données dans la scène secondaire

- H. Maintenant, il faut personnaliser la vue secondaire pour afficher les détails d'un élément choisi dans la scène principale. On peut ajouter des éléments graphiques en les glissant et déposant dans la scène.
- I. Ensuite, on doit définir les données à afficher en implémentant des méthodes dans la classe « DetailViewContrller ».
- J. Le passage de données entre la scène principale et la scène secondaire se fait par une liaison « Seque ». Dans le fichier StoryBoard, on doit définir l'identifiant de cette liaison.
- K. Dans la méthode « prepareForSegue » de la classe « MasterView Controller », on définit les données à transmettre à la scène secondaire.
- L. A la fin, on peut compilé le projet et le tester avec le simulateur.

III. Conception

1. Prototypage

1) Outils

Cocoa offre plein de layouts et widgets prédéfinis, par exemple, UITableView et UITableViewCell qui permettent d'afficher des données d'une manière ergonomique et réaliser des interactions classiques. Ces éléments ne peuvent pas répondre à tous les besoins, il est nécessaire de personnaliser les items et d'ajouter les effets. Donc, un bon outil de maquettage pour des applications iPhone doit d'une part offrir tous les éléments de base et refléter leurs tailles optimisées à l'écran et d'autre part proposer la possibilité de personnalisation. Xcode, l'environnement de travail pour le développement, propose des outils de construction de vues: .xib et le storyboard. Ces outils sont pratiques, avec les librairies Cocoa et toutes les configurations. Par contre, ce n'est pas la meilleure solution pour le prototypage parce que toutes les modifications sont reliées avec le code.

Il existe d'autres outils de maquettage. Quand on cherche les mots "iphone mockup", plus que dizaine d'outils afficheront dans le résultat. Par contre, ils ne sont pas tous cohérents avec nos besoins précisés précédemment. Une comparaison est indispensable avant de faire le choix.

Les outils peuvent être séparés dans 3 catégories principales :

- Prototypes en esquisse : Les maquettes de ce type sont souvent des wireframes. Les éléments ne reflètent pas forcément les tailles réelles. L'intérêt de ce type d'outils est qu'ils sont très simples. Ils ne permettent qu'à faire peu de personnalisation. Ces outils aident, dans la première phase de conception, à définir les différentes zones sans entrer dans les détails de chaque partie. Par exemple, Balsamiq et iphonemockup offre ce genre de vue globale. Les concepteurs peuvent discuter ces maquettes avec les clients pour comprendre leurs besoins avant de spécifier les différentes parties et d'appliquer les stratégies d'ergonomie.



Image III.1.a : Prototypes en esquisse

- Les libraires des images : Les libraires des images contiennent les icônes, les images de fond de taille optimisée pour iPhone. Pour concevoir une maquette, il faut placer ces éléments dans une image de fond d'iPhone. Ce n'est pas toujours facile à les utiliser. Pour ajouter des textes ou modifier des icônes, il demande beaucoup de travail avec les logiciels comme Photoshop. Avec ces outils, on peut créer des maquettes très jolies et très personnalisées, mais vu qu'ils ne tiennent pas compte des contraintes de programmation, ce n'est pas toujours facile à implémenter lors de la réalisation. Keynotopia, MockApps sont tous de ce type. Ces librairies sont souvent disponibles pour des logiciels comme Powerpoint et Key Note, ils sont peut-être un bon choix pour une présentation de produit.

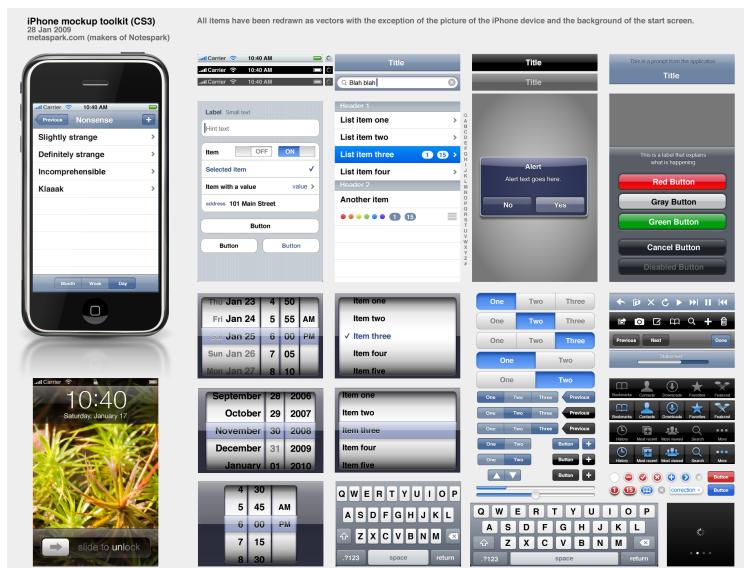


Image III.1.b : Librairie des images

- Maquette complète : Le dernier type d'outils ressemble beaucoup au storyboard de Xcode. Ils offrent la possibilité d'ajouter les éléments de Cocoa et de les personnaliser sous les contraintes de programmation. Par exemple, on peut personnaliser une cellule dans une liste en ajoutant les textes et les images, mais les modifications sur une barre de navigation sont limitées. Les outils sont différenciés par leur plateforme supporté, la possibilité de partage, etc. La façon d'interaction entre pages est également intéressante. Certains outils entre eux génèrent une page web que l'on peut accéder depuis un navigateur d'iPhone pour simuler le scénario d'utilisation. Ci-dessous la comparaison entre les différents outils dans ce type :



Image III.1.c : Maquette complète

	interface2	axure	Lucidchart	mockko
éléments de base	complet	peu	plupart	plupart
interaction	avec animation	liens	peu	liens
simplicité	simple	moyen	moyen	simple
plateforme	iOS	Mac/PC	web	web
format d'export	Xcode, pdf	page web	pdf, image	page web
partage	email pdf	non	avec url	avec url
prix	\$ 9,99	\$ 289	\$ 9,95/mois	gratuit

Image III.1.d : Comparaison de différentes méthodes

Le cas de cette étude de vulnérabilité consiste à créer des applications traditionnelles. Il faut simplement utiliser des listes et des éléments de navigation avec peu de personnalisation. Cependant, le nombre de données à afficher est important, il faut donc les définir avec la taille précise. Un outil de ce dernier type répond correctement aux besoins. Parmi les outils disponibles, nous avons sélectionnée mockko parce qu'il est facile à manipuler et à partager. De plus, sa possibilité de simuler le fonctionnement sur une page web est très intéressante pour faire les démonstrations.

2) Ergonomie

L'interface d'iPhone est spéciale. Sur un écran de 3,5 pouces (ou 4 pouces pour iPhone5), il faut placer un certain nombre d'informations dans un espace très limité. Il existe quelques conventions pour la conception de l'interface d'utilisateur. Une page est réparti dans plusieurs parties : la navigation en haut, la barre d'outils en bas et des informations essentiels au centre. En respectant ces modes de design, on aide à l'utilisateur à découvrir les applications. Pour la navigation, il faut se poser trois questions :

- 1) Où suis-je ? La réponse est donc le titre de ce page qui doit situer au centre de la navigation ;
- 2) Comment peux-je retourner à la page précédente ? Donc, nous avons besoin d'un bouton "Retour" à gauche ;
- 3) Qu'est-ce que je peux faire dans cette page ? Ce sont les items dans la barre d'outils qui doivent indiquer les actions possibles. D'autres places libres dans la navigation peuvent également être utilisées pour ces usages. Par exemple, la partie droite ou la deuxième ligne.

Afin d'afficher les données d'une manière lisible, on peut jouer avec les différents types de représentations : les couleurs, les positions, les tailles et les formes après l'analyse des ressemblances et des différences de données.

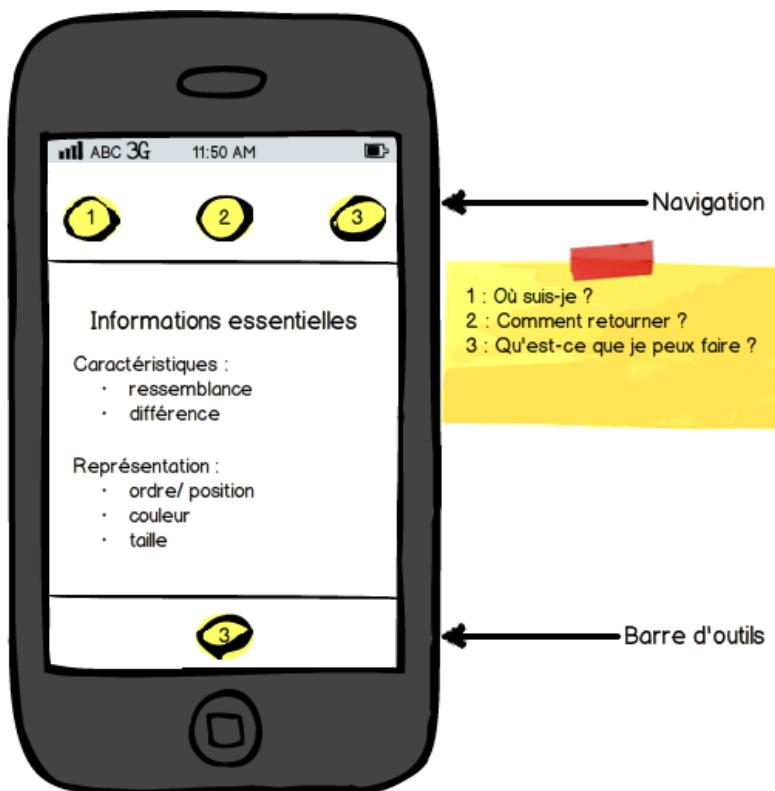


Image III.1.e : Ergonomie de l'interface graphique

En tant que débutants, il ne faut pas essayer d'appliquer trop de personnalisation sur l'interface. Il vaut mieux de tout d'abord concevoir des interfaces classiques et comprendre les logiques internes des widgets, layouts et interactions proposés par la plateforme et ensuite les personnaliser si on a une bonne raison.

Xcode propose plusieurs modèles à la création d'un nouveau projet. Les trois types que l'on utilise souvent sont : Master-Detail Application, Page-Based Application et Tabbed Application.

IV. Réalisation d'un démonstrateur de collecte de données

1. Contacts

1) Objectif

Nous avons commencé ce projet par études des données liées aux contacts, qui sont les données les plus sensibles dans un iPhone : non seulement parce qu'ils contiennent les vrais nom et numéros, les relations entre l'utilisateur et ses contacts peuvent intéresser aussi beaucoup d'entreprises qui travaillent avec les données personnelles. Le vol de données des contacts est un des problèmes majeurs pour les applications d'iPhone.

Dans cette première partie, nous avons essayé d'étudier la possibilité de récupérer des données de contacts sur un iPhone.

2) Etude des APIs

Dans l'iOS, l'accès aux contacts est géré par la classe « ABAddressBookRef », qui représente le carnet d'adresse dans iPhone. Avant d'accéder aux contacts, il faut demander l'autorisation de l'utilisateur : une fenêtre de confirmation va apparaître sur l'écran.



Image IV.1.a : Autorisation d'accès aux contacts

Après avoir eu la permission, on peut avoir accès aux contacts par le code suivant :

```
ABAddressBookRef addressBook = ABAddressBookCreateWithOptions(NULL, NULL);
CFArrayRef allPeople = ABAddressBookCopyArrayOfAllPeople( addressBook );
CFIndex numberOfPeople = ABAddressBookGetPersonCount( addressBook );
```

Code IV.1.a : Crédit du carnet d'adresse

Les contacts sont regroupés dans un objet de type « CFArrayRef », qui est en fait une référence vers un tableau en langage C. Dans ce tableau, chaque personne est représentée par un objet de type « ABRecordRef ». Nous pouvons accéder aux toutes les données liées à cette personne par la méthode « ABRecordCopyValue » avec des clés différentes définies par Apple. [Apple AddressBook Reference].

Par exemple, on se dispose du code suivant pour récupérer le nom et prénom d'un contact:

```
ABRecordRef thePeople = CFArrayGetValueAtIndex( allPeople, 0);
NSString *firstName = (__bridge NSString *)ABRecordCopyValue(thePeople,
kABPersonFirstNameProperty);
NSString *lastName = (__bridge NSString *)ABRecordCopyValue(thePeople,
kABPersonLastNameProperty);
```

Code IV.1.b: Accès aux informations d'un contact

Pour les données multi-valuées, par exemple, les numéros de téléphone ou les emails, il faut parcourir ces valeurs en utilisant une boucle:

```
ABRecordRef thePeople = CFArrayGetValueAtIndex( allPeople, 0);
const ABMultiValueRef *emails = ABRecordCopyValue(thePeople,
kABPersonEmailProperty);
CFStringRef emailRef;
NSString *emailStringValue;
for (int i = 0; i < ABMultiValueGetCount(emails); i++) {
    emailRef = ABMultiValueCopyValueAtIndex(emails, i);
    if (emailRef) {
        emailStringValue = (__bridge NSString *)emailRef;
        CFRelease(emailRef);
    }
}
```

Code IV.1.c : Accès aux données multivaluées

3) Présentation de l'application

Nous avons crée une petite application pour présenter les données de contacts récupérés. La vue principale de cette application est une liste de contacts avec leurs noms, prénoms, emails et numéros de téléphones. Une case contenant plus d'informations va afficher après la clique sur une cellule. Pour simuler le vol de données, un bouton en haut sur la droite nous permet d'envoyer l'ensemble des informations dans le carnet d'adresse vers un serveur prédéfini.



Image IV.1.b : Interface principale + Résultat retrouvé sur le serveur

2. Calendriers & Rappels

1) Terminologie

Anglais	Français	Description
Event	Événement	les événements dans l'application "Calendrier"
Reminder	Rappel	les tâches dans l'application "Rappel"
Calendar	Calendrier	groupe d'événements ou de rappels

2) Techniques

Apple fournie dans son framework Event Kit les APIs qui permet de lire, modifier et supprimer les événements et rappels. Les enregistrements sont sauvegardés dans une même base de données. Afin d'accéder à ces données, IOS demande l'autorisation d'utilisateurs.

Afin de récupérer les données, il faut d'abord instancier un objet d'EKEventStore de type EKEntityTypeEvent ou de type EKEntityTypeReminder et demande autorisation d'accès aux données.

Les groupes d'événements et de rappels s'appellent tous les deux "Calendar". La liste des calendriers est accessible en appelant la méthode [store calendarsForEntityType:EKEntityTypeEvent];

Ensuite, nous voudrons récupérer un tableau de dictionnaires des événements ou de rappels qui correspondent à un certain nombre de critères.

```
NSPredicate *predicate = [store  
predicateForEventsWithStartDate:dateBegin endDate:dateEnd  
calendars:calendars];  
  
NSArray *events = [store eventsMatchingPredicate:predicate];
```

Code IV.2.a: Récupération des événements

Pour le faire, il faut d'abord créer un "predicate" qui est en effet les conditions que l'on va utiliser pour récupérer les données. Il fonctionne comme les requête SQL "where attribut=valeur". L'instruction suivante [store eventsMatchingPredicate:predicate] envoie la requête à la base de données.

Par contre, les APIs sont un peu différent vu que "Rappels" sort dès iOS4.0 qui est beaucoup plus tard que "Calendrier". Il utilise les blocs "completion handler". "Completion handler" permettent d'effectuer une action lorsqu'une méthode termine sa tâche [2012, Apple F].

Après avoir créé le "predicate", on envoie la requête. Quand la fonction fini son travail, les codes dans le bloc "completion" vont être appelés afin de mettre à jour l'interface graphique.

```
NSPredicate *predicate = [store  
predicateForRemindersInCalendars:[NSArray arrayWithObjects:cal, nil]];  
  
[store fetchRemindersMatchingPredicate:predicate completion:^(NSArray  
*reminders) {completion(reminders)}];
```

Code IV.2.b : Récupération des rappels

3) Description de l'application "CalSpy"

L'application CalSpy permet d'afficher les événements selon le choix de calendriers et la période spécifiée. L'interface des applications précédentes est utilisée pour afficher la liste d'événements. Les événements sont divisés en sections de date et sont représentés dans l'ordre chronologique. Les informations affichées sont : le titre, l'heure de début et de fin, le calendrier, la location, et la note si il existe. Les couleurs choisies pour les calendriers par l'utilisateur sont représentées par une petite ligne verticale avant chaque événement.

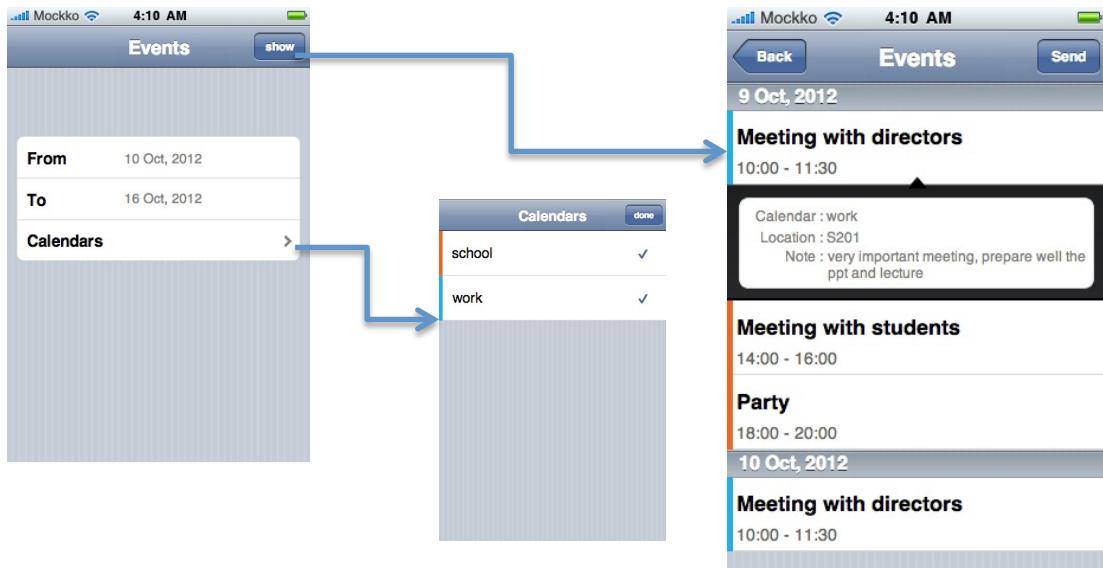


Image IV.2.a : Maquette de « CalSpy »

4) Description de l'application "RemSpy"

RemSpy fonctionne du même principe que CalSpy. L'utilisateur peut choisir un calendrier pour voir sa liste de rappels. Les informations essentielles sont le titre, la priorité, le temps voulu ou le temps fini si il existe. La priorité est lue d'un chiffre de 0 à 9, plus haut la valeur, moins prioritaire cette tâche. Nous l'avons donc interprétée dans 3 niveaux. Les tâches réalisées sont marquées par un petit tick dans sa ligne.

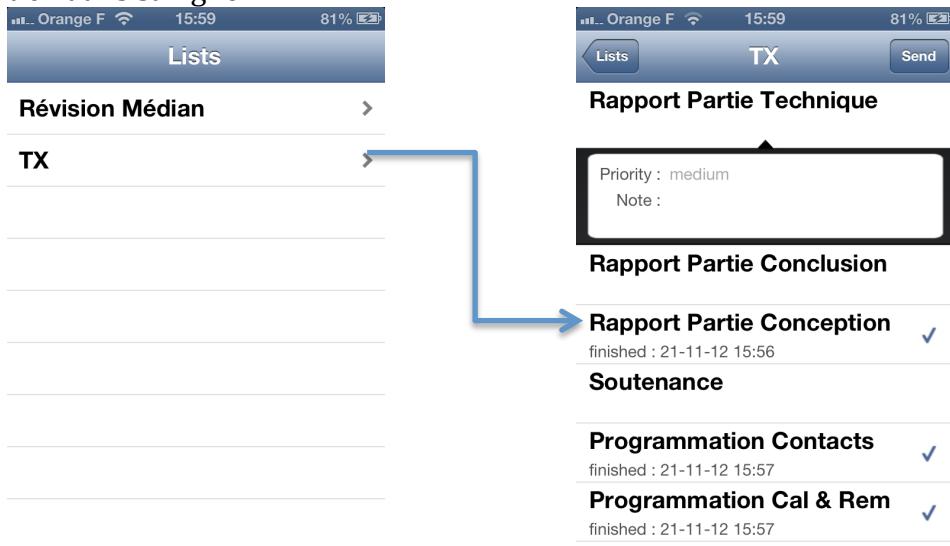


Image IV.3.b : Maquette de « BamSpy »

3 Photos

1) Objectif

Les photos sont les ressources les plus sensibles sur un Smartphone. Non seulement parce qu'elles sont des données les plus privées, elles contiennent aussi une grande quantité de métadonnées comme le EXIF, le TIFF et les informations de GPS.

- Les informations EXIF contiennent la description spécifique et technique d'une photo:
 - Information de l'heure et la date
 - La dimension de photo
 - Les réglages de l'appareil : cela comporte les informations telles que la vitesse d'obturation, la longueur de focale, la sensibilité, etc.
- Les informations GPS sont générées automatiquement lors que la création d'une photo. Elles contiennent l'altitude, la longitude et l'heure de la prise de photo. Nous pouvons positionner l'emplacement de photo sur une carte à partir de ces informations.
- Les informations TIFF contiennent des données complémentaires d'une photo, y compris les informations de l'appareil et du logiciel de retouche.

L'objectif de cette partie est de montrer la vulnérabilité des données photos sur un smart phone. Nous avons commencé à étudier les interfaces de programmation fournies par Apple. Et puis, nous avons essayé de récupérer les photos par un programme et lister leurs informations dans notre application.

2) Etudes des APIs

a) AssetsLibrary

L'accès aux photos est manipulé par la librairie « AssetsLibrary » [2012 Apple B]. C'est une librairie qui nous permet d'accéder aux toutes les photos et les vidéos contrôlées par l'application « Photos ». Les photos sont regroupées par des albums. Chaque photo ou vidéo est représenté par un objet de type « ALAsset ».

L'accès aux photos se fait par les codes suivants :

```
ALAssetsLibrary *library = [[ALAssetsLibrary alloc] init];
[library enumerateGroupsWithTypes:ALAssetsGroupSavedPhotos
    usingBlock:^(ALAssetsGroup *group, BOOL *stop) {
        [group enumerateAssetsUsingBlock:^(ALAsset *asset, NSUInteger index, BOOL
*stop) {
            if (asset) {
                //do something with this photo ...
            }
        }];
    }
failureBlock:^(NSError *error) { NSLog(@"Boom!!!!"); }
];
```

Code IV.3.a : Parcours de photos dans la librairie

Après avoir créé un default librairie, il suffit de la parcourir avec le méthode « enumerateGroupsWithTypes:usingBlock: », en précisant le type d'album. Et puis, dans chaque album, nous allons parcourir toutes les assets en utilisant la méthode « enumerateAssetsUsingBlock: ».

b) ALAsset

Un « ALAsset » représente une photo ou un morceau de vidéo. Un « ALAsset » peut avoir plusieurs représentations. Par exemple, une photo peut être capturé à la fois en RAW ou en JPEG. Les différentes représentations de la même photo peuvent avoir des dimensions différentes.

L'accès aux métadonnées d'une photo se fait par les codes suivants :

```
ALAssetRepresentation *representation = [asset defaultRepresentation];
NSDictionary *metaDictionary = [representation metadata];
NSArray *allKeys = [self.photoMetaData allKeys];
for(NSString *key in allKeys) {
    NSLog(@"%@", @"Value for key %@ is : %@", key, [[metaDictionary objectForKey:key] description]);
    //do somethings with the metadata ...
}
```

Code VI.3.b : Accès aux métadonnées d'une photo

Après avoir obtenu le défaut représentation d'une photo, nous pouvons accéder à ses métadonnées en plantant la méthode « metadata ». Les métadonnées sont représentées sous forme d'un dictionnaire, dont les clés sont prédefinies par Apple [Annexe 1]. Pour les photos qui sont prises par iPhone, les informations GPS vont être présentes dans les métadonnées et ce champ va devenir vide pour les images importées de l'extérieur.

3) Description de l'application

Pour la représentation des métadonnées d'une photo, nous avons construit une application de type « Master-Detail ». Au démarrage de l'application, nous allons parcourir l'album par défaut de l'utilisateur et récupérer toutes les photos dans un « NSArray » qui est un attribut du contrôleur principal. Chaque fois qu'il y une modification sur l'album, nous allons recharger cet attribut et actualiser la vue.

La vue principale de cette application est une liste (UITableView) des photos avec pour chaque photo une image miniature et le temps de capture. Le bouton « Open on Map » en haut à droite va afficher les photos sur une carte.



Image IV.3.a : Vue principale de l'application « SpyPhoto »

Après un click sur une cellule dans la liste, nous allons entrer dans l'interface secondaire avec l'affichage de toutes les métadonnées d'une photo. L'affichage se fait d'une manière dynamique: c'est-à-dire que nous affichons uniquement les informations présentes dans la photo sélectionnée. Par exemple, on n'affiche pas les champs GPS pour une photo importée de l'extérieur.

Nous avons affiché les photos dans une carte à l'aide de la classe « MKMapView ». Chaque photo est représentée par une annotation (MKAnnotation). Si on clique sur une annotation, une vue de type « Popup » contenant la miniature de la photo va apparaître.



Image IV.3.b : Métadonnées d'une photo

4. Service de localisation

1) Objectif

Depuis longtemps, la géolocalisation d'iPhone suscite une certaine inquiétude. Apple a été beaucoup reproché pour le fait qu'il collecte les informations d'emplacement de ses utilisateurs en temps réel. De nombreuses autres sociétés le font aussi, même sans laisser savoir les clients. Selon Apple, ces informations sensibles et privées ont été utilisées à améliorer la fonctionnalité de certains services. Mais, nous ne savons jamais si ces données sont transférées à un tiers.

L'objectif de cette partie est de rechercher sur les APIs d'Apple pour la géolocalisation et essayer de montrer la possibilité de tracer le déplacement en arrière-plan sans prévenir l'utilisateur.

2) Etudes des APIs

c) Géolocalisation

D'après les recherches sur la documentation officielle d'Apple, nous avons trouvé que l'accès aux données de déplacement est géré par la Classe « *CLLocationManager* » [2012, Apple A].

Il y a deux approches pour obtenir le changement de déplacement de l'utilisateur.

- Utiliser le service de localisation standard : spécifier la précision de déplacement et recevoir le mis à jour lorsqu'il y a un changement d'emplacement. Ce service est disponible pour toutes les versions d'iOS.
- Demander uniquement les changements d'emplacement significatifs, qui fournissent un service de suivi plus limité, mais offrent des économies d'énergie considérables. Ce service est disponible seulement dans l'iOS 4.0 ou plus et nécessite un appareil avec un radio cellulaire.

D'après des essais, nous avons trouvé que la précision de la deuxième méthode n'est pas très optimisée. Pour obtenir des données optimales, nous avons choisi d'utiliser la première approche. De plus, le changement de méthode est tout à fait possible et configurable dans l'application.

d) *CLLocationManager*

La création d'un objet « *CLLocationManager* » se fait avec le code suivant :

```
CLLocationManager *locationManager = [[CLLocationManager alloc] init];
locationManager.distanceFilter = 50;
locationManager.desiredAccuracy = kCLLocationAccuracyBest;
locationManager.delegate = self;
```

Code IV.4.a : Création de « CLLocationManager »

Après avoir crée un « CLLocationManager », il faut le configurer avec la précision souhaitée et le filtre de distance. Apple nous a fournit une échelle de précision cité ci-dessous :

```
kCLLocationAccuracyBestForNavigation  
kCLLocationAccuracyBest  
kCLLocationAccuracyNearestTenMeters  
kCLLocationAccuracyBestHundredMeters  
kCLLocationAccuracyKilometer  
kCLLocationAccuracyThreeKilometers
```

Code IV.4.b : Précision de données d'emplacement

La différence entre la « BestForNavigation » et la « Best » est que la première va utiliser non seulement les données de GPS mais aussi de les combiner avec des données supplémentaires détectées par de capteurs. La consommation d'énergie va diminuer suivant la perte de précision.

Le filtre de distance nous permet de contrôler la fréquence de déclenchement des événements. Par exemple, si nous fixons le filtre à 50 meters (exemple ci-dessus), l'événement de changement d'emplacement va être déclenché uniquement si la distance dépasse 10 mètres. Si nous choisissons la valeur « kCLLocationAccuracyNone » qui est la valeur par défaut, l'événement va être déclenché chaque seconde.

Pour utiliser le service de localisation standard, il faut appeler la méthode « startUpdatingLocation ». Si nous souhaitons utiliser la deuxième approche, il suffit d'appeler la méthode « startMonitoringSignificantLocationChanges ».

```
[self.locationManager startUpdatingLocation];  
//[self.locationManager startMonitoringSignificantLocationChanges];
```

Code IV.4.c : Démarrage de localisation

Le mis à jour d'emplacement est réalisé dans la méthode « locationManager: didUpdateLocations: ». Cette méthode est appelée chaque fois qu'il y a un changement d'emplacement. Chaque emplacement est encapsulé dans un objet de type « CLLocation », dans lequel on trouve l'altitude et la longitude.

```
-(void)locationManager:(CLLocationManager *)manager  
didUpdateLocations:(NSArray *)locations {  
    CLLocation *newLocation = [locations objectAtIndex:0];  
    // to do with newLocation ...  
}
```

Code IV.4.d : Mis à jour de l'emplacement

Pour arrêter la localisation, il suffit d'appeler la méthode « stopUpdatingLocation » pour la première approche ou « stopMonitoringSignificantLocationChanges » pour la deuxième.

```
[self.locationManager stopUpdatingLocation];
// [self.locationManager stopMonitoringSignificantLocationChanges];
```

Code IV.4.e : Fin de localisation

e) Exécution en arrière-plan

Pour la raison d'énergie, iOS autorise que l'exécution de trois types de services en arrière-plan : le GPS, VOIP (téléphone) et le music. Pour d'autres types de service, le gestionnaire de tâches dans iOS va les arrêter après 10 minutes d'exécution. Pour tracer la trajectoire de l'utilisateur, nous devons enregistrer le service GPS dans un mode « background ». Il faut ouvrir le fichier de configuration « info.plist » et ajouter la valeur « App registers for location updates » dans le champ « Required background modes », comme dans l'image ci-dessous :

Key	Type	Value
▼ Information Property List	Dictionary	(16 items)
Localization native development reg	String	en
Bundle display name	String	\${PRODUCT_NAME}
▼ Required background modes	Array	(1 item)
Item 0	String	App registers for location updates
Executable file	String	\${EXECUTABLE_NAME}
Bundle identifier	String	com.zhangzhenyi.\${PRODUCT_NAME}Identifier
InfoDictionary version	String	6.0
Bundle name	String	\${PRODUCT_NAME}
Icon file	String	SpyGeo.png
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0

Image IV.4.a: fichier « info.plist » pour localisation

f) Persistance de données

Nous avons construit une base de données « CoreData », afin de stocker les données d'emplacement que nous avons capturé. La base de données « CoreData » est en fait une base de données SQLite. Apple a ajouté des fonctionnalités supplémentaires pour le rendre plus simple à gérer. Pour chaque base de données, il faut créer un fichier de type « UIManagedDocument » en précisant le chemin de stockage. Et puis, il suffit d'envoyer des requêtes pour insérer ou récupérer des données.

Le modèle de la base de données est représenté par l'image suivante:

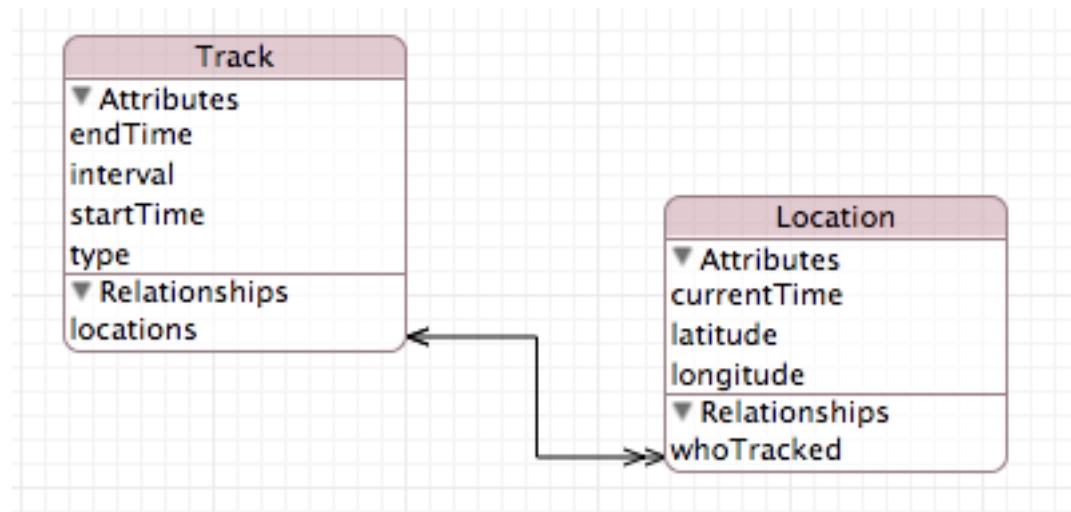


Image IV.4.b : modèle de la base de données « SpyGéo »

g) Représentation sur la carte

La représentation d'emplacement sur la carte est réalisée par la « MKMapView », où MK représente « Map Kit ». Dans la carte, chaque emplacement est représenté par une annotation de type « MKAnnotation ». Les coordonnées de chaque emplacement sont définies par l'attribut « coordinate » de cet objet. Nous pouvons personnaliser la forme et le contenu des annotations en surchargeant la méthode « title » et « subtitle ».

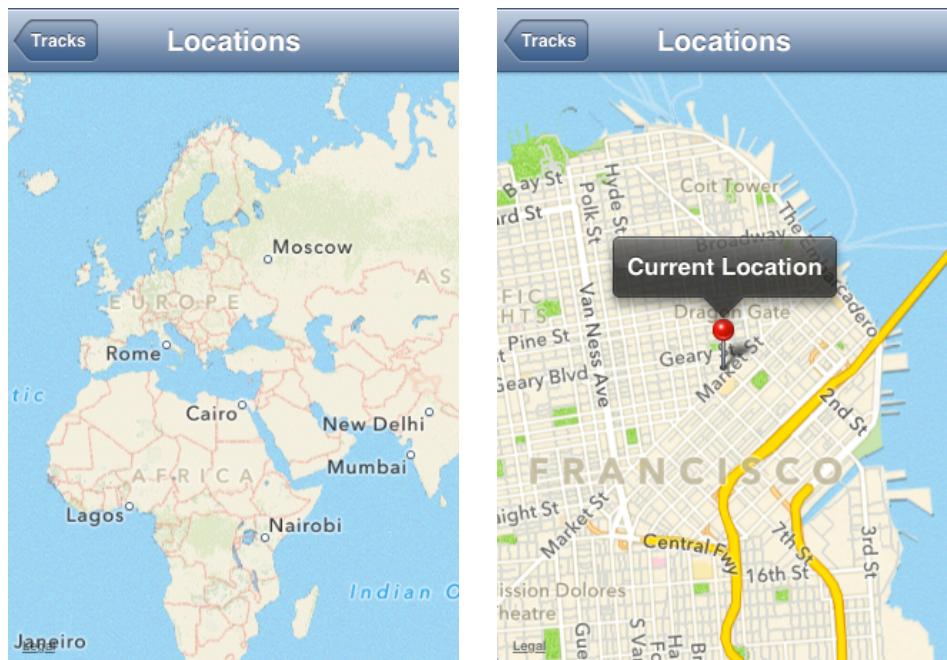


Image IV.4.c : MKMapView et MKAnnotation

3) Description de l'application

L'application que nous avons développée vous permet de tracer vos trajectoires et de les visualiser sur une carte.

La vue principale de cette application est une liste des pistes enregistrées dans la base de données. Vous pouvez lancer un nouvel enregistrement en appuyant sur le bouton « Plus » en haut sur la droite. Et puis, vous pouvez choisir le mode d'enregistrement entre « Auto » et « Timer ». Dans le mode « Auto », tous les changements d'emplacement vont être sauvegardé dans la base de données. Dans le mode « Timer », la détection va lancer régulièrement suivant l'intervalle choisi par l'utilisateur.

Vous pouvez visualiser les emplacements en cliquant sur une cellule dans la liste. En appuyant sur une annotation dans la carte, l'heure d'enregistrement et les coordonnées de cet emplacement vont être affiché sur l'écran.

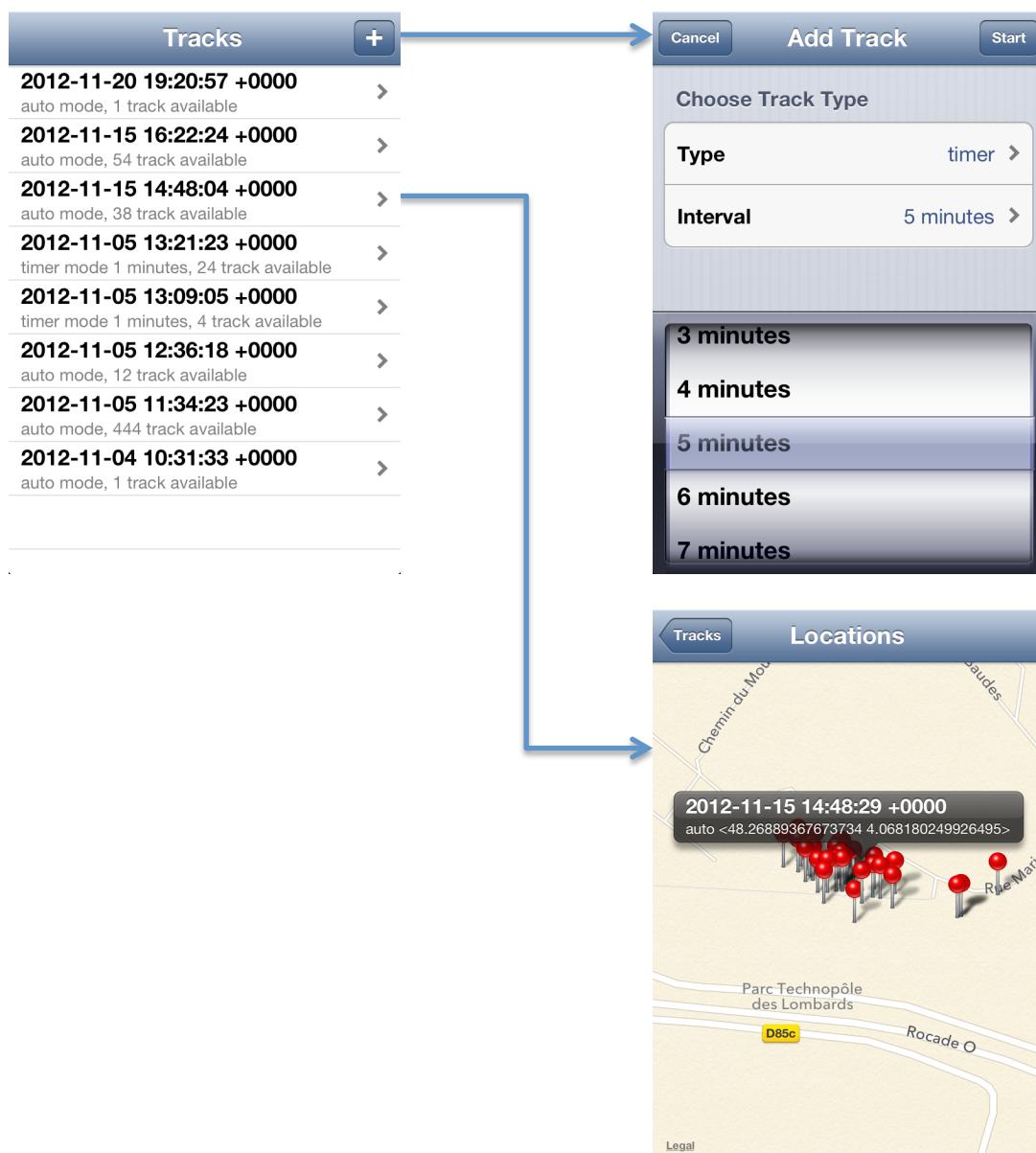


Image VI.4.d : Interface de l'application « SpyGéo »

5. Applications

Dans cette partie, on s'intéresse à étudier les applications d'iPhone. L'objectif est d'identifier les applications installées dans le mobile et d'analyser l'utilisation des applications. Apple ne fournit aucun APIs public concernant ces informations, mais il n'interdit pas à chercher dans les documents privés. On peut facilement consulter les fichiers "header" avec Xcode.

1) Processus

La premier essaie concerne les processus parce qu'ils ont une relation explicite dans tous les systèmes d'exploitation. Nous voudrons identifier les applications à partir de ce lien.

a) Récupérer les processus

Apple ne fournit pas d'API publique permettant d'accéder à la liste des processus. Par contre, IOS, comme Mac OS, est un système d'exploitation de type UNIX. Donc, la plupart des fonctions de système UNIX sont disponibles, y compris sysctl. Selon le BSD System Manager's Manual de Mac OS, l'utilitaire sysctl récupère l'état du noyau et permet aux processus avec les priviléges appropriés de définir l'état du noyau [2012, Processus]. Vus qu'il n'y a pas de documentations publiques pour cette fonction en IOS, nous repérons les libraires de Mac OS et nous supposons la même utilisation dans le système embarqué. L'utilisation de la fonction sysctl est comme suivant :

```
int sysctl(int *name, u_int namelen, void *oldp, size_t  
          *oldlenp, void *newp, size_t newlen);
```

Code VI.5.a : Utilisation de la fonction « sysctl »

En appelant cette méthode, on obtient des données de l'instant demandé. Les informations sont copiées dans la mémoire tampon spécifiée par oldp. La taille de la mémoire tampon est proposée par l'emplacement spécifié par oldlenp avant l'appel. oldlenp indique la quantité de données copiées après un appel réussi. Après un appel d'échec, il renvoie ENOMEM comme code d'erreur. [2012, sysctl(3)]

Nous pouvons appeler cette fonction par la synthèse suivante :

```
sysctl( (int *) name, (sizeof(name) / sizeof(*name)) - 1,  
        NULL, &length, NULL, 0);
```

Code VI.5.b : Appel de fonction « sysctl »

« name » dans le premier champ est composé d'une liste de noms de type "Management Information Base" (MIB) de plusieurs niveaux. Dans notre cas, le chemin est :

```
static const int name[] = { CTL_KERN, KERN_PROC,  
                           KERN_PROC_ALL, 0};
```

Code IV.5.c : MIB pour récupérer tous les processus [2002, processus]

En consultant le fichier sys/sysctl.h, nous avons vérifié que ces constantes existent également dans l'IOS. Afin de récupérer tous les processus, on commence par appeler la commande sysctl avec les pointeurs NULL et la longueur 0 :

```
int st = sysctl(mib, miblen, NULL, &size, NULL, 0);
```

Code IV.5.d : Initiation de buffer

Cela va réussir, et régler la longueur à la longueur appropriée. Dans une boucle, on va ensuite allouer des buffers de taille incrémentée et d'appeler à nouveau sysctl avec ce buffer.

```
st = sysctl(mib, miblen, process, &size, NULL, 0);
```

Code IV.5.e : Affectation de buffer [2010, iphone process]

b) Analyser les processus

Avec la liste des processus, nous pouvons observer les attributs des processus et les utiliser à réaliser nos objectifs. Annexe 2 montre un exemple de tous les attributs trouvés dans la structure d'un processus.

attribut	chemin
nom	kp_proc.p_comm
pid	kp_proc.p_pid
stat	kp_proc.p_stat
uid	kp_eproc.e_ucred.cr_uid [2011 process uid]
starttime	kp_proc.p_un._p_starttime tv_sec ou tv_usec

Image IV.5.a : Les informations basiques

Par la suite, nous allons utiliser ces attributs de base et d'autres informations à analyser les processus et voir ce que l'on peut trouver comme données personnelles d'utilisateurs.

2) Identifier les applications

a) Techniques

- Le premier problème nous intéresse est de séparer les processus d'applications et ceux du système. Dans un système UNIX ordinaire, les deux types de processus sont différenciés souvent par leurs uid : les processus du système sont exécutés par root, ceux des applications sont déclenchés par d'autres user. On essaie d'imprimer tous les processus avec leurs uid afin de vérifier si notre supposition est correcte (voir image ci-dessous). On constate qu'il y a plusieurs uid apparaissant dont les plus utilisés sont 501 et 0. Vu que notre application procSpy est déclenchée par 501, on suppose que 501 est l'identifiant de l'utilisateur et 0 est le root. Par contre, à part procSpy, il y a encore plusieurs processus liés avec

uid 501. Dans ce cas là, nous ne pouvons pas conclure que ceux qui est avec uid 501 sont les processus d'application, mais il faut trouver plus de différences que ce simple critère.

```

name=SystemExpert, uid=501
name=MobileTimer, uid=501
name=ptpd, uid=501
name=notification_pro, uid=501
name=afcd, uid=501
name=syslogd, uid=0
name=lsd, uid=501
name=springboardservi, uid=501
name=securityd, uid=64
name=lockbot, uid=0
name=syncdefaultsd, uid=501
name=keybagd, uid=0
name=debugserver, uid=501
name=ProcSpy, uid=501

```

Image IV.5.b : Liste de processus avec uid

- Deuxième essai concerne des fonctions spéciales dans IOS : openURL et canOpenURL. L'url ici non seulement contient les urls d'un page web, mais également peut être les schèmes vers une application. Par exemple, mailto:// pour ouvrir la page "nouvelle mail" de l'application Mail. De cette manière, on peut vérifier si un processus est une application en essayant d'ouvrir cette application avec le nom de processus. On applique la fonction canOpenURL comme un filtre sur la liste des processus afin de savoir quels sont les processus d'application. Cette technique fonctionne avec limite parce que le nom du processus d'une application n'est pas forcément le schème URL de cette application. Les schèmes peuvent être définis d'après le choix de développeurs des apps [2012, handleopenurl]. La liste de schèmes disponibles peut être trouvée dans le site <http://handleopenurl.com/scheme>. On peut constater que cette technique fonctionne dans certains cas, mais ses inconvénients sont également évidents.

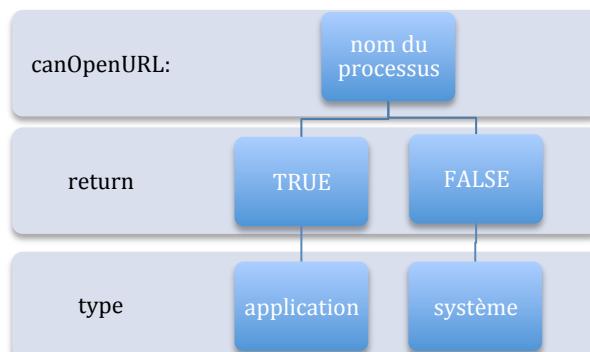


Image IV.5.c : Filtrage de processus par canOpenURL

- La méthode dernière a des difficultés à établir un lien entre les processus et les applications via url. Nous retournons vers la liste des processus et essayons de comparer les attributs d'un processus d'application et ceux d'un processus de système. Nous souhaitons de trouver une règle qui décrit la différence. Malheureusement, nous n'y avons pas réussi pour l'instant.
- Malgré que les trois dernières techniques ne répondent pas suffisamment notre situation, nous voulons quand même essayé une dernière méthode. Elle est la méthode la plus initiale, mais peut être la plus efficace dans notre cas. Supposons que le nombre de processus de système est limité, voir environ centaines, nous allons construire une petite base de données de tous les processus de système avec leurs noms. Quand on regarde un processus, si son nom n'apparait pas dans cette liste, on considère qu'il est un processus d'une application. On peut améliorer cette technique en une combinaison avec les deux premières qui consistent à réduire le nombre de processus à vérifier.

Après avoir séparé les processus d'applications de ceux de systèmes, il reste encore un problème pour identifier les applications exécutantes. Il faut connaître le nom réel des applications, voir leurs icônes et d'autres informations. En effet, les noms de processus ne sont pas identiques de ceux des applications. Ils sont les noms de projet lors de développement.

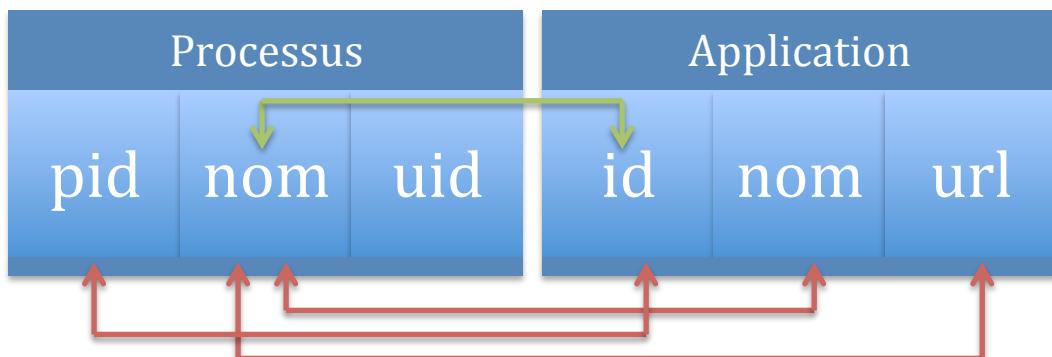


Image IV.5.d : Correspondance des attributs entre processus et application

Après une vérification de correspondances de toutes les paires des attributs entre processus et application, nous constatons que les liens en rouge dans le schéma en dessous ne peuvent pas établir.

De ce fait, si on veut connaître la relation entre un processus et une application, il faut avoir un tableau qui décrit la correspondance entre eux. Ce tableau a besoin 2 colonnes fondamentaux (qui sont reliés avec le lien vert) : nom des processus et id des applications. Voici un exemple :

nom de processus	id d'application
Facebook	284882215
Evernote	281796108
Twitter	333903271

Image IV.5.e : Exemple de tableau de relation entre processus et application

Pid n'est pas utilisé dans ce cas parce qu'il change pour une application lors 2 fois d'exécution. Le nom d'application est également variable selon la préférence du système (langage...) Par la suite, on peut repérer les noms d'application en interrogeant l'app store avec l'api de recherche :

```
http://itunes.apple.com/lookup?id=284882215
```

Code IV.5.f : API de recherche d'iTunes [2012, Search iTunes API]

b) Apps existantes

Pour réaliser ce travail, nous avons également étudié certaines apps existantes dans app Store.

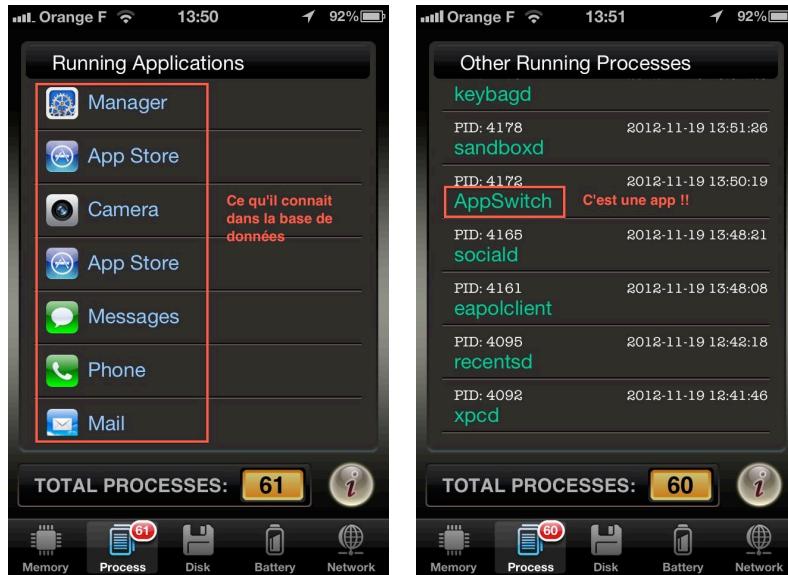


Image IV.5.f : Capture d'écran de l'app « SYS-Activity »



Image IV.5.g : Capture d'écran de l'app « AppSwitch »

Même si certains entre eux déclarent qu'ils peuvent détecter les processus d'applications, mais le résultat semble plus ou moins imprécis. Il y a toujours des

apps qu'ils n'arrivent pas à identifier. Selon nos études, on pense que c'est à cause de la base de données incomplète. Bien évidemment, la base de données est de grande dimension avec 700 milles applications en ligne. Certaines applications visent à utiliser la puissance de communauté à enrichir leurs bases de données. Ils demandent aux utilisateurs d'identifier les applications installées dans leurs mobiles.

Pour conclure, les applications et les processus ont des relations internes. Par contre, cette relation est difficile d'être exploitée car ils ont différentes représentations. La structure d'un processus ne reflète directement les propriétés de son application correspondante, il faut donc établir manuellement les liens entre eux avec une base de données.

3) Description de l'application "iHasApp"



Image IV.5.h : La relation des propriétés de laquelle « iHasApp » exploite

Avec tout ceux que l'on possède jusqu'à l'instant, nous pouvons réaliser un programme qui détecte des applications installées dans un mobile. iHasApp, un framework opensource propose des APIs pour connaître les identifiants des applications installées [2011, Amitay]. Ce framework n'exploite pas la relation entre processus et application, mais que les propriétés d'application.

```

12301   "xing2": [297625850],
12302   "xkcd": [303688284],
12303   "xlr8": [506486124],
12304   "xmagmicbdrx": [324160308],
12305   "xmagmicbigkahunawords": [345313219, 388452002],
12306   "xmagmickaglomx": [322002209],
12307   "xmagmicknytx": [307569751],
12308   "xmagmictphase10x": [302049354],
12309   "xmas2011": [473419372],
12310   "xmasbooth": [405787010],
12311   "xmasbooth-url-scheme": [405787010],
12312   "xmaspinball": [337425607],
12313   "xmpp": [327603487],
12314   "xms": [425154540],
12315   "xmslive": [425154540],
12316   "xx-ymsgn": [309219097],

```

Image IV.5.g : Exemple de base de données « iHasApp »

Dans le code source de ce framework, on localise, un objet Json qui décrit la relation entre les urls et les appId dans iTunes des applications. Il y a actuellement 12430 enregistrements. Afin de détecter les applications installées, il itère la liste des urls qu'il connaît et essaie d'ouvrir chacun avec canOpenURL. Si la fonction retourne TRUE, c'est-à-dire que l'application est installée, le framework récupère son id avec la base de données ci-dessus. Avec l'identifiant de l'application dans iTunes, on obtient toutes ses informations, y compris le nom et l'icône.

Dans l'application SpyUTT, ce framework est intégré. Dans le résultat de détection, on peut observer que la liste d'application n'est pas complète. C'est normal parce que le résultat dépend la base de données qui ne contient que les applications populaires.

4) Analyser l'utilisation des applications

a) Addiction aux smartphones

Plus en plus de monde fait attention à leurs dépendances aux smartphones. On croit que l'on dépense trop de temps sur les smartphones et éventuellement a eu l'addiction. Les phénomènes que l'on éteint jamais le mobile, on tweet ou visite facebook régulièrement avec le mobile, on jette un coup d'oeil au mobile avant de dormir chaque soir et après de se lever chaque matin montrent que l'on est probablement un "iphonedépendant". [2012, grobubu] Nombreux d'utilisateurs se rendent compte de ce type d'addiction. Prenons l'exemple du jeu Angry Birds, un jeu très populaire sur les smartphones, il y a plus que 3 millions résultats quand on recherche "angry bird addiction" sur google. Les gens s'inquiètent du fait qu'ils ne peuvent pas arrêter de jouer le jeu et ils étendent le son même si ils le ferment [2012, Damian]. Étonnant par le chiffre de 5 million heure mondiale sur ce jeu par jour [2011, Noah], nous sommes préoccupés par les problèmes qu'il provoque. On perd concentration pendant le temps de travail à cause des ouvertures et des consultations fréquentes des mobiles. A part des jeux, les apps qui sont publiées comme productivité et utilité sont également critiquées pour le formalisme.

Pendant ces 20 ans récents, l'addiction à la télé, à Internet, aux jeux vidéo et maintenant au mobile et à ses applications sont discutés sans arrêts. Par contre, il y a des gens qui insistent que nos inquiétudes sont excessives parce que par rapport les addictions aux drogues, les addictions comportementales semblent moins graves. Tout le monde a des addictions comportementales qui indiquent une partie de nos passions. [2012, Frances]

Peu importe si l'addiction aux smartphones est grave, il faut que nous prenons en compte ce que l'on fait avec le smartphones et prenons contrôles si nécessaire. Pour cette raison, nous voudrons proposer un outil qui permet de consulter l'utilisation de mobile. Le temps total passé sur des applications et leurs taux d'utilisation pendant une période est des alertes pour les gens.

Pour savoir quel application est en avant-plan à un instant donnée, il faut connaître le changement d'état des applications : soit on regarde régulièrement le processus en avant-plan, soit on écoute aux événements. De toute façon, il faut d'abord connaître les différents états d'application définie par Apple.

b) Cycle de vie d'une application

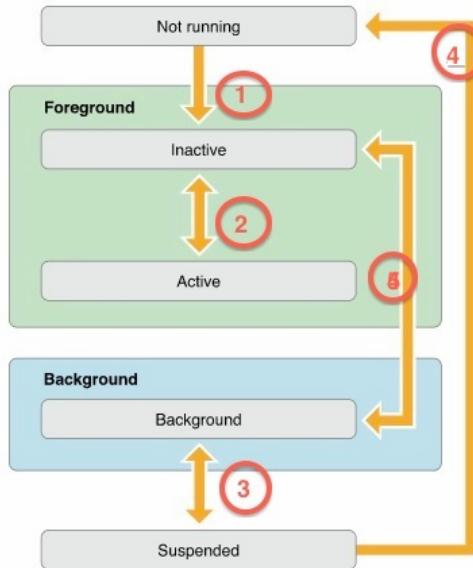


Image IV.5.i : Cycle de vie d'une application

Le cycle de vie d'une application est décrit par 5 états et 5 transitions potentielles entre eux. Une application est exécutée par cliquer sur son icône. Cette application sera exécutée au premier écran en état inactif, c'est-à-dire qu'elle exécute certains codes mais elle n'écoute pas aux événements. Ensuite, il passe à l'état actif et reçoit les événements. C'est l'état le plus commun pour les applications exécutantes en avant-plan. Quand l'usager touche le "home" bouton, et envoie cette application au fond, l'état passe à "daemon". Elle continue à tourner pendant un certain temps. Pour la plupart des applications, le temps sera très court voir 10 minute d'après expérience avant qu'elles sont suspendues par le système. Il y a des exceptions qui concernent des applications ou des services comme musique, VoIP, téléchargement, gps et qui a besoin d'échanger les données régulièrement avec un accessoire externe. Dans ces cas, l'application peut être exécutée en mode "daemon" pendant longtemps et normalement le système ne la suspend pas. Sinon, l'application passe vite à l'état "suspendue". En ce moment, même si les applications apparaissent dans la barre "multitasks", aucun processus n'est en cours. Cette application revient au "non exécutant" si on le ferme manuellement. D'un point de vue fonctionnel, les deux états sont identiques, mais ils ne sont pas pareil au niveau de l'apparence sur l'interface. Le système peut suspend les applications "daemon" sans les notifier au cas où le mémoire est faible. La transition 5 explique le cas où on lance une application en arrière-plan de la barre "multitask".

c) Techniques

- Nous continuons dans la voix d'analyser les processus. Par contre, les états d'application et de processus n'ont pas de correspondance bijective. Les états inactif, actif, daemon sont tous représentés par l'état "currently runnable" de processus. De plus, l'heure démarrage que l'on lit de la liste des attributs processus n'est enregistré qu'au moment quand l'application est lancée de l'état "non exécutant". Il ne se rafraîchit pas lors de la transition de l'arrière-plan vers l'avant-plan. A cause de ces

deux raisons, on ne peut pas estimer le temps d'utilisation réelle. En prenant compte de la conclusion précédente, nous pensons que l'étude sur les processus n'est pas le bon chemin pour analyser les utilisations des applications des utilisateurs.

- Lorsque nous travaillons sur les processus, nous trouvons plusieurs possibilités intéressantes. L'une est le fichier event.h. Dans ce fichier, les constantes des états d'application sont définies. Nous pensons qu'il y a peut-être une méthode à écouter aux changements d'états, par exemple avec kevent, une autre fonction BSD. L'autre possibilité se met à consulter le log du système afin de regarder les actions d'utilisateur. La difficulté ici est de filtrer les informations. A cause du manque de connaissances spécifiques sur ces deux parties, nous n'avons pas pu aller plus loin avec ces deux méthodes.
- Après les études précédentes, une direction devient plus en plus claire. C'est que l'utilisation des applications a un lien très fort avec l'interface graphique. On définit d'une manière avant-plan et arrière-plan. Alors pourquoi on n'essaie pas d'exploiter l'écran d'iPhone. Dans IOS, l'écran où s'affiche les applications s'appelle SpringBoard [2012 WikiPedia].

```
#define SBSERVPATH
"/System/Library/PrivateFrameworks/SpringBoardServices.framework/SpringBoardServices"

mach_port_t *port;
port = (mach_port_t *)SBSSpringBoardServerPort();
void *sbserv = dlopen(SBSERVPATH, RTLD_LAZY);
void*
(*SBFrontmostApplicationDisplayIdentifier)(mach_port_t*
port,char * result) =
dlsym(sbserv, "SBFrontmostApplicationDisplayIdentifier");
char frontmostAppS[256];
memset(frontmostAppS,sizeof(frontmostAppS),0);
SBFrontmostApplicationDisplayIdentifier(port,frontmostAppS);
NSString * frontmostApp=[NSString
stringWithFormat:@"%@",frontmostAppS];
//NSLog(@"%@",frontmostApp);
dlclose(sbserv);
```

Code IV.5.g : Utilisation de « SpringBoard »

Avec les documents privés, on trouve la fonction qui permet de récupérer l'identifiant de l'application en avant-plan. [2011, Vikarti]

```
http://itunes.apple.com/lookup?bundleId=com.skype.skype
```

Code IV.5.h : Exemple d'identifiant d'une application

Cet identifiant est le bundleId d'app sous format : com.com.skype. Nous allons ensuite faire la recherche avec API d'iTunes et identifier l'application avec son nom et icône. Jusqu'à ici, nos problèmes techniques sont presque résolus. Il reste à enregistrer et analyser les données.

5) Description de l'application "Addictions"

Après avoir réussi à détecter les applications en cours à n'importe quel moment, nous voulons ensuite collecter ces données et essayer de les analyser afin d'étudier l'addiction d'utilisateur aux apps. Avec le big bang de technologies

Une fois l'application est lancée, elle exécute comme un daemon en enregistrant l'application en cours après chaque Δt ($\Delta t = 10$ seconde). Nous utilisons une astuce qui est d'écouter aux événements de GPS pour qu'elle puisse exécuter plus que 10 minutes après entrer dans dernier-écran.

Dans la base de données, il y a une seule table qui est constitué des traces. Une trace contient son bundleid et l'heure d'enregistrement. Pour faciliter la recherche, on divise l'heure dans 2 parties : le jour et l'heure. Le jour est un chiffre d'1 à 7 qui représente lundi à dimanche. Cette structure nous permet de filtrer les données par ces 2 critères. On garde également le format complet de l'heure pour appliquer d'autres analyses éventuelles.

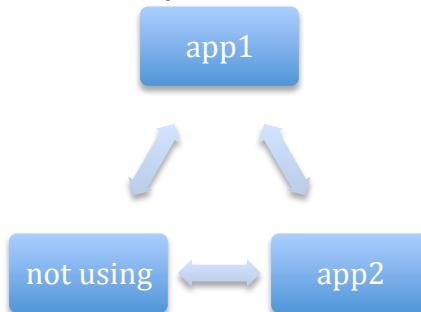


Image IV.5.j: transition d'état possible des apps dans une période

La méthode nous permet d'obtenir les snapshots de l'utilisation. Par contre, les changements d'états sont compliqués, pendant une période, l'application en cours peut être fermée, une autre application peut être lancée, ou encore, la dernière est fermée et on relance la première. Ce n'est pas grave si on connaît pas du tout ce qui se passe pendant cette période. Vu que la période est relativement courte (qui est beaucoup plus courte que le temps moyen que l'on passe sur des applications), on peut considérer que pendant une période, l'utilisateur reste toujours dans la dernière application détectée. De cette manière, on peut estimer le temps d'utilisation en comptant le nombre d'enregistrements d'une application pendant une période.

La première fonctionnalité de notre app "Addiction" est d'afficher le temps d'utilisation de chaque app. Dans une liste, nous pouvons voir les applications et leurs temps d'utilisation dans l'ordre croissant.

La deuxième fonctionnalité fournit une interface de recherche le taux d'utilisation par 2 dimensions : le jour et l'heure. Par exemple, on peut voir dans les jours de travail (lundi à vendredi), on dépense la plupart du temps à consulter les mails le matin ; les soirs de chaque vendredi, on joue des jeux sans arrêts ; et tous les weekends, le mobile se repose tranquillement comme nous.

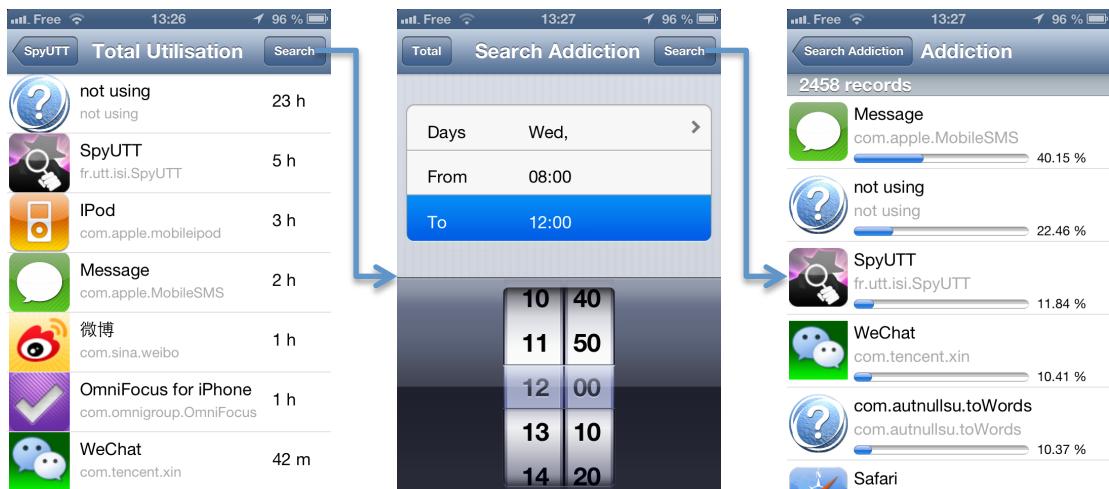


Image IV.5.k: Maquette de l'app "Addiction"

6. Device

1) Objectif

Dans cette partie, nous allons essayer de récupérer les informations liées au dispositif.

2) Etude des APIs

Access au dispositif est géré par la classe « `UIDevice` ». Cette classe nous fournit une instance de type singleton qui représente le dispositif. A partir de cette instance, nous pouvons récupérer les informations sur le nom d'utilisateur, le modèle du dispositif, le nom et la version du système d'exploitation.

Access aux informations :

```
UIDevice *device = [UIDevice currentDevice];
NSString *name = [device name];
NSString *systemName = [device systemName];
NSUUID *uuid = [device identifierForVendor];
...
...
```

Code IV.6.a : Access aux informations du dispositif

Nous pouvons également accéder aux données des deux capteurs sur iPhone : le capteur pour la proximité et le capteur pour l'orientation de l'écran. De plus, le niveau de batterie est également accessible à l'aide de la classe « `UIDevice` ».

Access aux capteurs est réalisé par le code suivant:

```
UIDevice *device = [UIDevice currentDevice];
NSInteger *orientation = [device orientation];
NSInteger *proximityState = [device proximityState];
Float *batteryLevel = [device batteryLevel];
BOOL *isBatteryMonitoringEnabled = [device isBatteryMonitoringEnabled];
BOOL *isProximityMonitoringEnabled = [device isProximityMonitoringEnabled];
```

Code IV.6.b : Access aux capteurs

Pour obtenir le changement d'état des capteurs, nous avons utilisé la technique de notification dans iOS :

```
[ [NSNotificationCenter defaultCenter]
    addObserver:self
        selector:@selector(orientationChanged)
        name:UIDeviceOrientationDidChangeNotification
        object:nil];
```

Code IV.6.c : Changement d'état d'un capteur

Lorsqu'un changement d'état, la méthode « orientationChanged » va être appelée.

3) Description de l'application

Nous avons crée une application simple pour présenter les informations liées au dispositif. La vue principale de cette application est une liste de type « UITableView » où chaque cellule représente un paire de clé-valeur. Un bouton en haut sur la droite nous permet d'activer et de désactiver les capteurs.



Image IV.6.a : Interface de l'application « SpyDevice »

7. Autres

1) Données non accessibles

a) Notes

L'app Notes dans IOS est un outil pratique. On enregistre souvent des pensees, des numéros, des plannings même des numéro de cartes bancaires dedans. Est-ce que les autres applications peuvent accéder aux textes dans Notes ? Heureusement, la réponse est non. Apple n'a publie aucun APIs pour Notes. C'est-à-dire qu'aucun tiers app ne peut accéder aux informations dans Notes

b) Safari

Comme tous les autres navigateurs existants, on trouve dans la base de données de safari, les historiques, les téléchargements et les cookies. Par contre, ces informations sont exclusives pour Safari. Sans APIs, ces données ne sont pas visibles aux autres applications.

Dans la préférence du système, utilisateurs peuvent voir ce qu'il y en a dans la base de données (Safari > Advanced > Website Data) et exécuter les opérations pour supprimer toutes les traces.

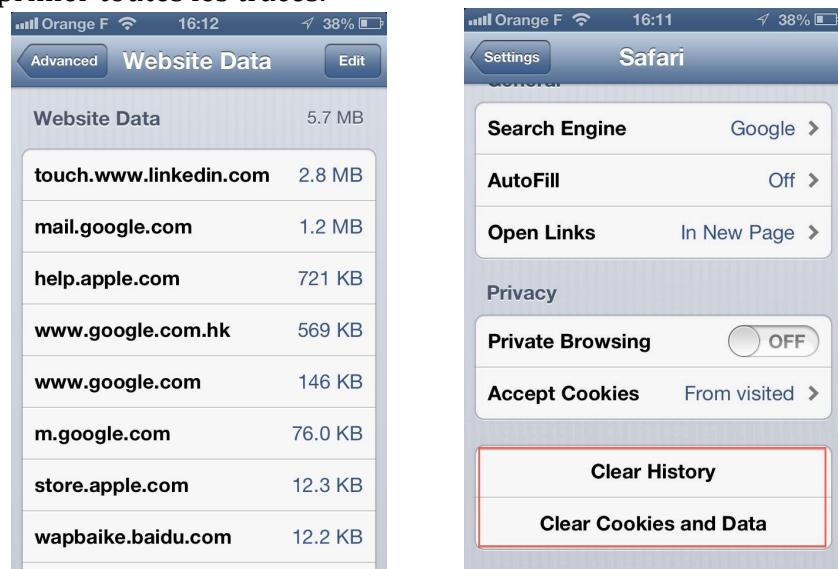


Image IV.7.a: Gestion des données de Safari

A partir de IOS 5.0, Apple fournit la possibilité de visiter les pages sans enregistrer les données. Cela peut être configuré dans la préférence [2012, Orgera].



Image IV.7.b: Confidentialité de Safari

c) *Messages/ Téléphone/Mail*

Les fonctionnalités de base dans un mobile sont messages, téléphone et mail. Si les historiques d'appels, des messages ou des mails peuvent être récupérés par d'autres personnes, cela peut être un grand risque. Afin de permettre aux développeurs d'exploiter ces fonctionnalités en garant la sécurité des données concernées, Apple ne fournit que certaines méthodes pour accéder à l'interface graphique. C'est-à-dire que les opérations et les transmis des données sont encapsulés et on ne les voit pas. Apple propose deux façons :

Premièrement, comme nous avons discutés dans les parties précédentes, les applications sont accessibles par leurs urls. Par exemple, on peut ouvrir l'application "message" par sms: ou encore créer un nouveau message en mettant un numéro portable suivant cet url. L'application "mail" fonctionne de la même façon avec l'url mailto: sauf que les attributs sont plus riches en contenant le sujet, les destinataires en copie, etc. [2012, Apple Link]. L'inconvénient de cette méthode est qu'il ouvre une autre application et par conséquent les utilisateurs quittent forcément l'application. Cela ne convient pas la situation où on veut retourner à l'App au cours, par exemple, on voudrait seulement envoyer un mail pour signaler un bug et continuer utiliser l'App par la suite.

Depuis IOS4.0, Apple propose la deuxième méthode qui répond à ce besoin. Les développeurs peuvent intégrer des scènes de MessageUI framework dans leurs applications. Il faut simplement créer une instance de MFMailComposeViewController ou de MFMessageComposeViewController et configurer ses attributs. Toutes ces deux méthodes ne permettent qu'utiliser l'interface graphique sans accéder directement aux données et opérations essentielles.

```

MFMailComposeViewController *controller = [[MFMailComposeViewController
alloc] init];
[controller setSubject:@"In app email..."];
[controller setMessageBody:@"...a tutorial from mobileorchard.com"
isHTML:NO];

```

Code IV.7.a: Utilisation de MessageUI [2009, Grigsby]

Quand il s'agit de la téléphonie, nous avons repéré le Core Telephony Framework qui gère ces informations [2010, Apple Telephony]. Ce framework vise à fournir les utilités aux fournisseurs de services téléphonies. Nous n'avons pas eu du temps à approfondir dans ce domaine. Par contre, des possibilités intéressantes avec ce framework sont de bloquer certains appels ou de collecter les appels en écoutant aux événements [2011, Vikarti].

2) SandBox

Pour des raisons de sécurité, iOS place chaque application (y compris ses préférences et données) dans un SandBox au moment de l'installation. Un SandBox est un ensemble de contrôles qui limitent l'accès de l'application à des fichiers, les préférences, les ressources réseau, le matériel, et ainsi de suite. Le système installe chaque application dans son propre répertoire SandBox, qui agit comme le /home pour l'application et ses données.

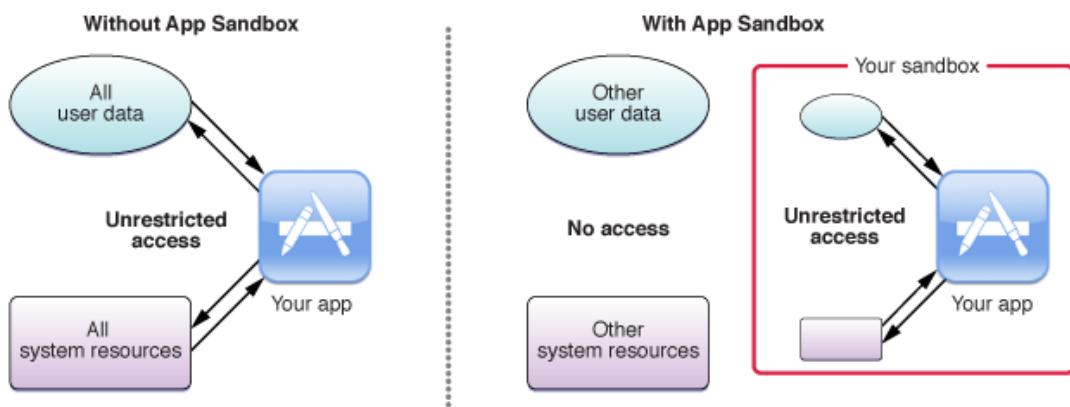


Image IV.7.c: SandBox [2012, Apple SandBox]

Le but de ce mécanisme est de séparer les applications et les rendre individuels. De cette façon, il empêche l'application détournée d'affecter d'autres applications et d'autres parties du système. D'un autre côté, les communications entre applications sont interdites et les données spécifiques à une App sont sécurisées. Ce n'est pas possible de gérer les bases de données ou les fichiers d'une autre App.

Lors de la programmation, on peut manipuler les données dans le SandBox pour faire le teste. Avec "Organizer" dans Xcode, le SandBox est téléchargeable sous extension .xcappdata. Le fichier exporté peut être exploité comme un répertoire normal dont la base de données sqlite se situe dans AppData/Documents/Default\ Track\ Database/StoreContent/persistentStore

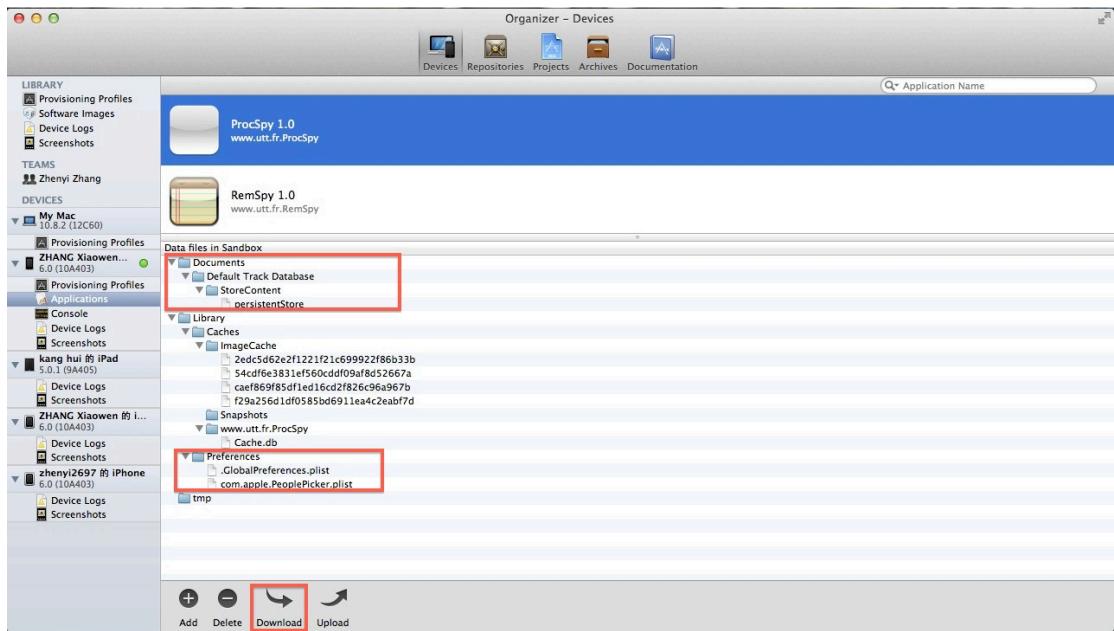


Image IV.7.d : SandBox d'une App

V. Bilan de protection

1. Confidentialité dans IOS 6.0

A partir d'iOS 6.0, une nouvelle rubrique est apparue dans le « Réglages » : Confidentialité. Dans cette rubrique, nous pouvons gérer tous les accès aux données privées dans notre Smartphone : Contacts, Calendriers, Rappels, Photos, Localisation et les comptes de réseaux sociaux.



Image V.1.a : Gestion de confidentialité

Si on entre dans un type de données, les applications qui ont demandé l'accès à ce type de données vont être listé sur l'écran : non seulement les apps auxquelles vous avez donné l'accès, les apps que vous avez refusées font partie aussi de la liste. Si vous trouvez qu'une application utilise les données qui n'a pas de lien avec sa fonctionnalité, vous pouvez désactiver tout de suite son accès à fin de protéger nos données privées.

Dans le champ « Service de localisation », nous pouvons également désactiver entièrement le service ou bien application par application. Les petits icônes devant chaque application nous permettent de distinguer les apps qui viennent d'utiliser le service de localisation, les apps qui ont utilisé lors des dernières 24 heures et les apps qui utilisent une barrière virtuelle.

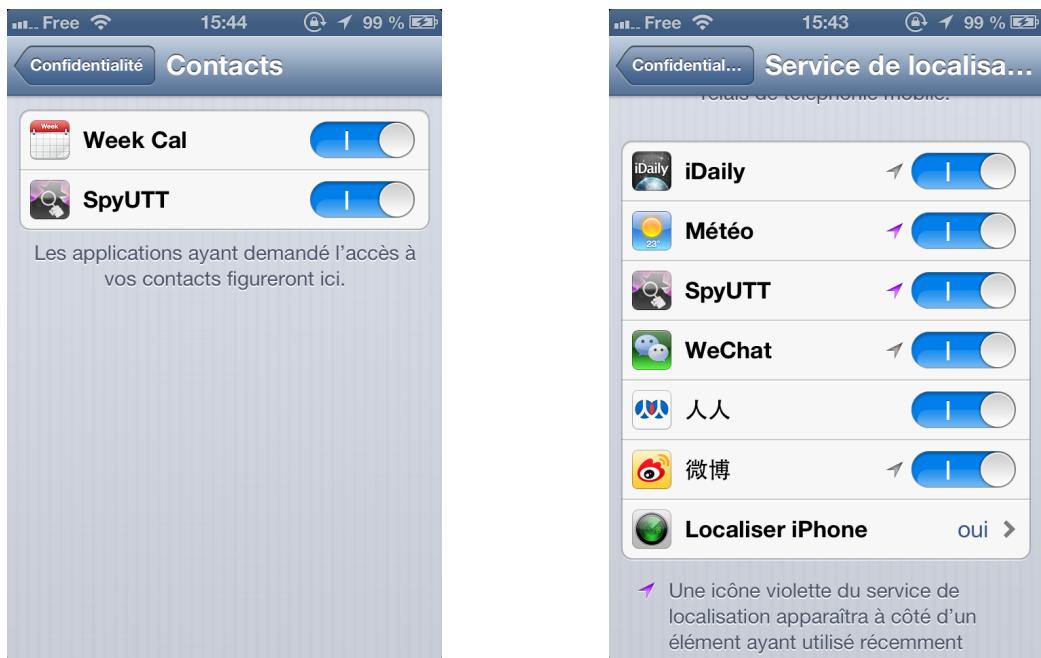


Image V.1.b : Liste d'applications qui ont demandé l'accès aux données

Si vous voulez désactiver tous les accès aux données privées et recommencer la phase d'autorisation, vous pouvez réinitialiser la gestion de confidentialité en allant dans la rubrique « Général – Réinitialiser – Réin. Localisation et confidentialité ».



Image V.1.c : Réinitialisation des accès aux données privées

2. Guide directive d'Apple

1) Extraction des règles

A fin de contrôler la qualité des applications dans l'App Store et d'éliminer les applications malicieuses, un guide officiel de la validation a été mis en disposition par Apple à partir du Septembre 2010. [2012, Apple store review guideline] Nous avons étudié ce guide et listé les règles liées à la protection des données personnelles:

2. Fonctionnement

2.5 Les Apps qui utilisent des API non-publiques seront rejetées.

2.16 Les applications multitâches ne doivent exploiter les services d'arrière-plan que pour leur propre besoin: VoIP, jeu audio, géolocalisation, fin de tâches, notifications locales, etc.

4. Géolocalisation

4.1 Les Apps qui ne demandent pas et n'obtiennent pas le consentement de l'utilisateur avant de collecter, transmettre ou exploiter le système de géolocalisation seront rejetées.

4.4 Les données de localisation peuvent être utilisés que lorsqu'ils sont directement en rapport avec les fonctions et services fournis par l'application pour l'utilisateur ou pour soutenir la publicité utilisations approuvées.

17. Vie Privée

17.1 Les Apps ne peuvent transmettre des données à propos d'un utilisateur sans avoir obtenu de lui la permission et sans l'informer de la façon dont les données seront exploitées.

22 Obligations légales

22.7 Les développeurs qui créent des applications pour récupérer insidieusement les identifiants ou mots de passe ou toute autre donnée privée seront bannis du programme des développeurs iOS.

Image V.2.a : Extraction des règles dans « Apple Store Review Guideline »

2) Analyse des termes

Pour le terme 2.5, les APIs qui ne sont pas listés dans la documentation officielle font partie de APIs non-publique. Dans notre application, la détection des applications en premier plan se fait par un API non-publique (SpringBoard). C'est pour cela que ce type d'application n'existe pas dans l'App Store. Le test des APIs non-publique est réalisé par des outils automatiques [Outils Automatiques de Test]. Avant de soumettre l'application, il est recommandé de vérifier l'application avec ces outils.

Le terme 2.16 limite l'utilisation des services en arrière-plan. Comme nous avons expliqué dans les parties précédentes, Apple autorisent que trois types de

services en arrière-plan : le téléphone, la géolocalisation et le music. Dans notre application, nous avons utilisé la géolocalisation pour détecter le taux d'utilisation des autres applications. Cette fonctionnalité n'est forcément pas autorisée dans l'App Store.

Les autres termes listés dessus consistent à protéger les données privées: l'application n'est pas autorisé à envoyer les données sans obtenu la permission de l'utilisateur. [2012 Apple Approval Processus]

3. Conseils de protection

Nos études sur le vol de données nous permettent de conclure quelques points pour sécuriser les données dans les smartphones. Nous constatons que la plupart des risques viennent des applications installées.

- Jailbreak est dangereux parce que vous donnez le droit de root aux apps. Dans ce cas, toutes les manipulations sur votre mobile sont possibles.
- Téléchargez et installez que les apps d'App Store parce qu'elles passent les examens rigoureux d'Apple.

Même si Apple prend des mesures pour protéger les utilisateurs, c'est possible que les apps faites "plus" que la façon qu'elle est décrite. Quand une application vous demande certaines permissions, c'est vous qui prenez la responsabilité.

- Avant de donner à une application les permissions d'accès aux données personnelles, vérifiez bien ce qu'elle va faire avec ces données.
- Méfiez les applications qui demandent d'utilisation de géolocalisation, même si elles ne sont pas des daemons, géolocalisation consomme beaucoup de batterie.

Apple ne peut pas connaître les traitements qu'une app fait quand elle communique avec un serveur à distance.

- Quand le symbole de connexion d'Internet tourne sans une action de votre part, vérifiez ce que l'application transmet.
- Méfiez les applications qui vous permettent de faire sauvegarder des contacts, photos, calendriers, etc. Ils enregistrent les données sur leurs serveurs et vous ne connaissez jamais ce qu'ils font avec.
- Consultez vos trafics d'internet. Si le trafic est trop grand, une cause possible est qu'une app communique vos données avec son serveur.

D'autres mesures de sécurité qu'un utilisateur peut faire :

- Verrouillez le mobile avec un mot de passe. Même si c'est tout simple, c'est une première protection de vos données.
- Fermez régulièrement les applications en dernier-plan afin d'éviter les daemons.
- Si c'est possible, n'enregistrez pas les informations essentielles dans le mobile. La perte physique de l'appareil et le vol de données par des apps menace ces données.
- Si vous avez besoin d'utiliser une app pour mémoriser vos informations essentielles, par exemple, les mots de passe, utilisez Notes parce qu'elle est relativement sécurisée.

VI. Conclusion

Dans le cadre de ce TPEX, un ensemble des applications sont été conçues, analysées et développées pour analyser la vulnérabilité des données personnelles sur iPhone. Le principe était de récupérer les données personnelles en prétendant des logiciels malveillants. Les données confidentielles contiennent les contacts, les calendriers (y compris les rappels), les localisations, les photos et les propriétés du dispositif. Ces données sont accessibles avec des frameworks proposés par Apple. Nous avons récupéré ces données et les réorganisé d'une manière plus susceptible. La situation d'utilisations des Apps ainsi nous intéresse non seulement parce que c'est une information essentielle, mais également parce que l'on peut l'utiliser afin d'étudier l'addiction éventuelle aux Smartphones. Nous constatons que les autres données ne sont pas accessibles à cause du mécanisme de SandBox que l'on peut faire confiance. A la fin, nous avons analysé les efforts qu'Apple a fait et ce que les utilisateurs peuvent faire attention afin de protéger leurs données personnelles.

Ce projet TPEX nous a permis de développer les connaissances sur la sécurité des données personnelles sur mobile, plus concrètement, sur iPhone. En tant qu'un plateforme enrichit par les Apps de n'importe quel origine, le système risque de révéler les données personnelles d'utilisateurs. Les données intéressent les malveillants pour différentes utilisations. La conception des applications, notamment sur la maquettage nous permet d'approfondir la compréhension sur l'ergonomie d'interface humain-machine. La spécialité de l'écran iPhone est prise en compte lors de la conception. L'étude sur les frameworks et le développement nous offre une opportunité d'apprendre le développement mobile de A à Z. Nous avons réussi à comprendre et pratiquer la langage Objective-C, le pattern MVC et à approfondir la compréhension sur programmation orientée objet. Nous avons mise en place la méthodologie d'ingénieur pour trouver des solutions pour une question : identification des problématiques, proposition des solutions, mise en oeuvre des solutions, validation des solutions.

Il reste encore des améliorations auxquelles l'on peut penser. D'un côté, on peut changer de point de vue à répondre à la question. Nous avons commencé par regarder ce que l'on peut "voler". Par contre, même si l'on a réussi à développer les applications, on se rend compte que ces Apps peuvent rencontrer des restrictions lards d'entrer dans App Store. C'est donc également intéressant de connaître toutes les stratégies de protection qu'Apple met en place. D'autre côté, après la récupération des données, nous n'en avons pas encore exploité. Par exemple, on peut essayer d'établir une base de données de contacts de plusieurs utilisateurs et d'envoyer les SPAM. On peut également écrire un programme afin de filtrer les informations utiles dans ce que l'on a récupéré.

VII. Bibliographie

- [2012 CNIL] CNIL, Avec la CNIL, « *apprenez à sécuriser votre smartphone!* », <http://www.cnil.fr/la-cnil/actualite/article/article/avec-la-cnil-apprenez-a-securiser-votre-smartphone/>, visité le 21/11/2012
- [2012 MCLOV] ARUN THAMPI, « *Path uploads your entire iPhone address book to its servers* », <http://mclov.in/2012/02/08/path-uploads-your-entire-address-book-to-their-servers.html>, visité le 21/11/2012
- [2012, Apple A] Apple Inc., « *CLLocationManager Class Reference* », https://developer.apple.com/library/mac/#documentation/CoreLocation/Reference/CLLocationManager_Class/CLLocationManager/CLLocationManager.html, visité le 21/11/2012
- [2012, Apple B] Apple Inc., « *Assets Library Framework Reference* », http://developer.apple.com/library/ios/#documentation/AssetsLibrary/Reference/AssetsLibraryFramework/_index.html, visité le 21/11/2012
- [2012, Apple C] Apple Inc., « *Introduction to Calendars and Reminders* », <http://developer.apple.com/library/ios/#documentation/DataManagement/Conceptual/EventKitProgGuide/Introduction/Introduction.html>, visité le 21/11/2012
- [2012, Apple D] Apple Inc., « *Advanced App Tricks* », <http://developer.apple.com/library/ios/#documentation/iphone/conceptual/iphonesosprogrammingguide/AdvancedAppTricks/AdvancedAppTricks.html>, visité le 21/11/2012
- [2012, Apple E] Apple Inc., « *App States and Multitasking* », <http://developer.apple.com/library/ios/#documentation/iphone/conceptual/iphonesosprogrammingguide/ManagingYourApplicationsFlow/ManagingYourApplicationsFlow.html>, visité le 21/11/2012
- [2012, Apple F] Apple Inc., A Short Practical Guide to Blocks, http://developer.apple.com/library/ios/#featuredarticles/Short_Practical_Guide_Blocks/_index.html, visité le 21/11/2012
- [2012, sysctl(8)] Apple Inc., « *sysctl(8) OS X Manual Page* », <https://developer.apple.com/library/mac/#documentation/Darwin/Reference/ManPages/man8/sysctl.8.html>, visité le 21/11/2012
- [2012, sysctl(3)] Apple Inc., « *sysctl(3) OS X Developer Tools Manual Page* », <http://developer.apple.com/library/Mac/#documentation/Darwin/Reference/ManPages/man3/sysctl.3.html>, visité le 21/11/2012
- [2002, Processus] Apple Inc., « *Getting List of All Processes on Mac OS X* », <http://developer.apple.com/legacy/mac/library/#qa/qa2001/qa1123.html>, visité le 21/11/2012

[2010, iPhone Process] Stackoverflow, « *Can we retrieve the applications currently running in iPhone and iPad* »,
<http://stackoverflow.com/questions/4312613/can-we-retrieve-the-applications-currently-running-in-iphone-and-ipad>, visité le 21/11/2012

[2012, iHasApp] Danielamitay, iHasApp,
<https://github.com/danielamitay/iHasApp/blob/master/iHasApp/schemeApps.json>, visité le 21/11/2012

[2012, Proocss Uid] Stackoverflow, « *Can I use `sysctl` to retrieve a process list with the user?* », <http://stackoverflow.com/questions/7729245/can-i-use-sysctl-to-retrieve-a-process-list-with-the-user>, visité le 21/11/2012

[2012, handleopenurl] Magnatron, « *URL Schemes* »,
<http://handleopenurl.com/scheme>, visité le 21/11/2012

[2011, Amitay] Daniel Amitay, « *How To Detect Installed iOS Apps* »,
<http://danielamitay.com/blog/2011/2/16/how-to-detect-installed-ios-apps>, visité le 21/11/2012

[2012, iHasApp] iHasApp, <http://www.ihasapp.com/>, visité le 21/11/2012,
visité le 21/11/2012

[2012, iTunes API] Apple Inc., « *Affiliate Ressources* »,
<http://www.apple.com/itunes/affiliates/resources/documentation/itunes-store-web-service-search-api.html>, visité le 21/11/2012

[2012 WikiPedia] WikiPedia, « *SpringBoard* »,
<http://en.wikipedia.org/wiki/SpringBoard>, visité le 21/11/2012

[2012, Grobubu] Grobubu, « *Humour : 8 questions pour faire le point sur votre addiction (ou non) à l'iPhone* », <http://www.iphon.fr/post/8-points-addiction-iPhone>, visité le 06/12/2012

[2012, Damian] Damian Thompson, The new global addiction: smartphones,
<http://www.telegraph.co.uk/technology/9334058/The-new-global-addiction-smartphones.html>, visité le 06/12/2012

[2011, Noah] Noah Davis, « *The World Plays 5 Million Hours Of Angry Birds Per Day* », http://articles.businessinsider.com/2011-10-19/tech/30296725_1_cartoon-series-telegraph-rovio, visité le 06/12/2012

[2010, Oulasvirta] Antti Oulasvirta, Tye Rattenbury, Lingyi Ma, Eeva Raita, « *Habits make smartphone use more pervasive* »,
http://delivery.acm.org/10.1145/2130000/2124486/779_2011_Article_412.pdf?ip=193.49.161.211&acc=ACTIVE%20SERVICE&CFID=153910026&CFTOKEN=24496052&_acm_=1354877792_96a0f3038b3683b974804648bd1042e6, visité le 06/12/2012

[2012, Frances] Allen Frances, « *Do We All Have Behavioral Addictions?* »,
http://www.huffingtonpost.com/allen-frances/behavioral-addiction_b_1215967.html, visité le 06/12/2012

[2011, Vikarti] Vikarti Anatra, « *how to determine which apps are background and which app is foreground on iOS by application id* »,
<http://stackoverflow.com/questions/8252396/how-to-determine-which-apps-are-background-and-which-app-is-foreground-on-ios-by>, visité le 06/12/2012

[2012, Orgera] Scott Orgera, « *How to Enable Private Browsing in Safari for iPhone* »,
<http://browsers.about.com/od/howtousemobilebrowser1/ss/How-To-Enable-Private-Browsing-In-Safari-For-Iphone.htm>, visité le 06/12/2012

[2012 Apple AddressBook Reference] Apple Inc., « *ABAddressBook Reference* »,
http://developer.apple.com/library/ios/#documentation/AddressBook/Reference/ABAddressBookRef_iPhoneOS/Reference/reference.html, visité le 06/12/2012

[2010 Apple Store Review Guideline] Apple Inc., « *App Review* »,
<https://developer.apple.com/appstore/guidelines.html>, visité le 06/12/2012

[Apple Approval Processus] Wikipedia, « *Approval of iOS apps* »,
http://en.wikipedia.org/wiki/Approval_of_iOS_apps, visité le 06/12/2012

[2010 Private API Check] Jakob Borg, « *Check for private API “usage” yourself?* »,
<http://stackoverflow.com/questions/3546046/check-for-private-api-usage-yourself>, visité le 06/12/2012

[2012, Apple Link] Apple Inc, « *Apple URL Scheme Reference* »,
http://developer.apple.com/library/ios/#featuredarticles/iPhoneURLScheme_Reference/Introduction/Introduction.html#/apple_ref/doc/uid/TP40007891-SW1, visité le 07/12/2012

[2009, Grigsby] Dan Grigsby, « *In App Email, MessageUI* »,
<http://mobileorchard.com/new-in-iphone-30-tutorial-series-part-2-in-app-email-messageui/>, visité le 07/12/2012

[2010, Apple Telephony] Apple Inc, « *Core Telephony Framework Reference* »,
http://developer.apple.com/library/ios/#documentation/NetworkingInternet/Reference/CoreTelephonyFrameworkReference/_index.html#/apple_ref/doc/uid/TP40009603, visité le 07/12/2012

[2011, Vikarti] Vikarti, « *How can I use private APIs to block incoming calls in an iOS application?* »,
<http://stackoverflow.com/questions/7326338/how-can-i-use-private-apis-to-block-incoming-calls-in-an-ios-application>, visité le 07/12/2012

[2012, Apple SandBox] Apple Inc, « *App Sandbox Design Guide* »,
<http://developer.apple.com/library/mac/#documentation/Security/Conceptual/AppSandboxDesignGuide/AboutAppSandbox/AboutAppSandbox.html>, visité le
07/12/2012

VIII. ANNEXES

1. Annexe 1 : Métadonnées d'une photo

```
{Exif} : {  
    ApertureValue = "2.970854";  
    ColorSpace = 1;  
    ComponentsConfiguration = ( 1,  
        2,  
        3,  
        0  
    );  
    DateTimeDigitized = "2011:04:25 17:50:18";  
    DateTimeOriginal = "2011:04:25 17:50:18";  
    ExifVersion = ( 2,  
        2,  
        1  
    );  
    ExposureMode = 0;  
    ExposureProgram = 2;  
    ExposureTime = "0.008333334";  
    FNumber = "2.8";  
    Flash = 32;  
    FlashPixVersion = ( 1,  
        0  
    );  
    FocalLength = "3.85";  
    ISOSpeedRatings = ( 100  
    );  
    MeteringMode = 3;  
    PixelXDimension = 2048;  
    PixelYDimension = 1536;  
    SceneCaptureType = 0;  
    SensingMethod = 2;  
    Sharpness = 1;  
    ShutterSpeedValue = "6.906612";  
    SubjectArea = ( 1104,  
        657,  
        382,  
        384  
    );  
    WhiteBalance = 0;  
}
```

```
{GPS} : {
    ImgDirection = "201.4966";
    ImgDirectionRef = T;
    Latitude = "52.124";
    LatitudeRef = N;
    Longitude = "4.509333333333333";
    LongitudeRef = E;
    TimeStamp = "16:03:40.00";
}
Depth : 8
Orientation : 6
ColorModel : RGB
DPIWidth : 72
DPIHeight : 72
PixelHeight : 1536
PixelWidth : 2048
{TIFF} : {
    DateTime = "2011:04:25 17:50:18";
    Make = Apple;
    Model = "iPhone 3GS";
    Orientation = 6;
    ResolutionUnit = 2;
    Software = "4.3.2";
    XResolution = 72;
    YResolution = 72;
    "_YCbCrPositioning" = 1;
}
```

2. Annexe 2: Liste des attributs des processus

```
(kinfo_proc) $1 = {
    (extern_proc) kp_proc = {
        (extern_proc::<anonymous union>) p_un = {
            (extern_proc::<anonymous struct>) p_st1 = {
                (proc *) __p_forw = 0x509802c6
                (proc *) __p_back = 0x0005238d
            }
            (timeval) __p_starttime = {
                (__darwin_time_t) tv_sec = 1352139462
                (__darwin_suseconds_t) tv_usec = 336781
            }
        }
        (vmspace *) p_vmspace = 0x00000000
        (sigacts *) p_sigacts = 0x00000000
        (int) p_flag = 16384
        (char) p_stat = '\x02'
        (pid_t) p_pid = 2003
        (pid_t) p_oppid = 0
        (int) p_dupfd = 0
        (caddr_t) user_stack = 0x2fd0d000
        (void *) exit_thread = 0x00000000
        (int) p_debugger = 0
        (boolean_t) sigwait = 0
        (u_int) p_estcpu = 0
        (int) p_cpticks = 0
        (fixpt_t) p_pctcpu = 0
        (void *) p_wchan = 0x00000000
        (char *) p_wmesg = 0x00000000
        (u_int) p_swtime = 0
        (u_int) p_slptime = 0
        (itimerval) p_realtimer = {
            (timeval) it_interval = {
                (__darwin_time_t) tv_sec = 0
                (__darwin_suseconds_t) tv_usec = 0
            }
            (timeval) it_value = {
                (__darwin_time_t) tv_sec = 0
                (__darwin_suseconds_t) tv_usec = 0
            }
        }
        (timeval) p_rttime = {
            (__darwin_time_t) tv_sec = 0
            (__darwin_suseconds_t) tv_usec = 0
        }
        (u_quad_t) p_uticks = 0
        (u_quad_t) p_sticks = 0
        (u_quad_t) p_iticks = 0
        (int) p_traceflag = 0
        (vnode *) p_tracep = 0x00000000
        (int) p_siglist = 0
        (vnode *) p_textvp = 0x00000000
        (int) p_holdcnt = 0
        (sigset_t) p_sigmask = 0
    }
}
```

```

(sigset_t) p_sigignore = 407404544
(sigset_t) p_sigcatch = 4097
(u_char) p_priority = '\x18'
(u_char) p_usrpri = '\0'
(char) p_nice = '\0'
(char [17]) p_comm = "debugserver" {
    (char) [0] = 'd'
    (char) [1] = 'e'
    (char) [2] = 'b'
    (char) [3] = 'u'
    (char) [4] = 'g'
    (char) [5] = 's'
    (char) [6] = 'e'
    (char) [7] = 'r'
    (char) [8] = 'v'
    (char) [9] = 'e'
    (char) [10] = 'r'
    (char) [11] = '\0'
    (char) [12] = '\0'
    (char) [13] = '\0'
    (char) [14] = '\0'
    (char) [15] = '\0'
    (char) [16] = '\0'
}
(pgrp *) p_pgrp = 0x00000000
(user *) p_addr = 0x00000000
(u_short) p_xstat = 0
(u_short) p_acflag = 0
(rusage *) p_ru = 0x00000000
}
(eproc) kp_eproc = {
    (proc *) e_paddr = 0x00000000
    (session *) e_sess = 0x00000000
    (_pcred) e_pcrcd = {
        (char [72]) pc_lock = "" {
            (char) [0] = '\0'
            (char) [1] = '\0'
            (char) [2] = '\0'
            (char) [3] = '\0'
            (char) [4] = '\0'
            (char) [5] = '\0'
            (char) [6] = '\0'
            (char) [7] = '\0'
            (char) [8] = '\0'
            (char) [9] = '\0'
            (char) [10] = '\0'
            (char) [11] = '\0'
            (char) [12] = '\0'
            (char) [13] = '\0'
            (char) [14] = '\0'
            (char) [15] = '\0'
            (char) [16] = '\0'
            (char) [17] = '\0'
            (char) [18] = '\0'
        }
    }
}

```

```
(char) [19] = '\0'
(char) [20] = '\0'
(char) [21] = '\0'
(char) [22] = '\0'
(char) [23] = '\0'
(char) [24] = '\0'
(char) [25] = '\0'
(char) [26] = '\0'
(char) [27] = '\0'
(char) [28] = '\0'
(char) [29] = '\0'
(char) [30] = '\0'
(char) [31] = '\0'
(char) [32] = '\0'
(char) [33] = '\0'
(char) [34] = '\0'
(char) [35] = '\0'
(char) [36] = '\0'
(char) [37] = '\0'
(char) [38] = '\0'
(char) [39] = '\0'
(char) [40] = '\0'
(char) [41] = '\0'
(char) [42] = '\0'
(char) [43] = '\0'
(char) [44] = '\0'
(char) [45] = '\0'
(char) [46] = '\0'
(char) [47] = '\0'
(char) [48] = '\0'
(char) [49] = '\0'
(char) [50] = '\0'
(char) [51] = '\0'
(char) [52] = '\0'
(char) [53] = '\0'
(char) [54] = '\0'
(char) [55] = '\0'
(char) [56] = '\0'
(char) [57] = '\0'
(char) [58] = '\0'
(char) [59] = '\0'
(char) [60] = '\0'
(char) [61] = '\0'
(char) [62] = '\0'
(char) [63] = '\0'
(char) [64] = '\0'
(char) [65] = '\0'
(char) [66] = '\0'
(char) [67] = '\0'
(char) [68] = '\0'
(char) [69] = '\0'
(char) [70] = '\0'
(char) [71] = '\0'
```

{

```

    (ucred *) pc_ucred = 0x00000000
    (uid_t) p_ruid = 501
    (uid_t) p_svuid = 501
    (gid_t) p_rgid = 501
    (gid_t) p_svgid = 501
    (int) p_refcnt = 0
}
({_ucred) e_ucred = {
    (int32_t) cr_ref = 24
    (uid_t) cr_uid = 501
    (short) cr_ngroups = 1
    (gid_t [16]) cr_groups = {
        (gid_t) [0] = 501
        (gid_t) [1] = 0
        (gid_t) [2] = 0
        (gid_t) [3] = 0
        (gid_t) [4] = 0
        (gid_t) [5] = 0
        (gid_t) [6] = 0
        (gid_t) [7] = 0
        (gid_t) [8] = 0
        (gid_t) [9] = 0
        (gid_t) [10] = 0
        (gid_t) [11] = 0
        (gid_t) [12] = 0
        (gid_t) [13] = 0
        (gid_t) [14] = 0
        (gid_t) [15] = 0
    }
}
({_vmspace) e_vm = {
    (int32_t) dummy = 0
    (caddr_t) dummy2 = 0x00000000
    (int32_t [5]) dummy3 = {
        (int32_t) [0] = 0
        (int32_t) [1] = 0
        (int32_t) [2] = 0
        (int32_t) [3] = 0
        (int32_t) [4] = 0
    }
    (caddr_t [3]) dummy4 = {
        (caddr_t) [0] = 0x00000000
        (caddr_t) [1] = 0x00000000
        (caddr_t) [2] = 0x00000000
    }
}
(pid_t) e_ppid = 1974
(pid_t) e_pgid = 2003
(short) e_jobc = 1
(dev_t) e_tdev = -1
(pid_t) e_tpgid = 0
(session *) e_tsess = 0x00000000
(char [8]) e_wmesg = "" {
    (char) [0] = '\0'

```

```

        (char) [1] = '\0'
        (char) [2] = '\0'
        (char) [3] = '\0'
        (char) [4] = '\0'
        (char) [5] = '\0'
        (char) [6] = '\0'
        (char) [7] = '\0'
    }
    (segsz_t) e_xsize = 0
    (short) e_xrssize = 0
    (short) e_xccount = 0
    (short) e_xswrss = 0
    (int32_t) e_flag = 0
    (char [12]) e_login = "" {
        (char) [0] = '\0'
        (char) [1] = '\0'
        (char) [2] = '\0'
        (char) [3] = '\0'
        (char) [4] = '\0'
        (char) [5] = '\0'
        (char) [6] = '\0'
        (char) [7] = '\0'
        (char) [8] = '\0'
        (char) [9] = '\0'
        (char) [10] = '\0'
        (char) [11] = '\0'
    }
    (int32_t [4]) e_spare = (0, 0, 0, 0)
}
}

```