



Dotnet France
Technologies Sharepoint, SQL Server & .NET

Association Dotnet France

Débugger et Emuler Windows Mobile

Sommaire

1	Introduction.....	3
2	Détecter et corriger des erreurs de programmation	4
2.1	Utilisation de la liste des erreurs.....	4
2.2	Utilisation du débogueur.....	5
2.2.1	Entrer en mode débogage.....	5
2.2.2	Différences entre le .Net Compact Framework et le full .Net Framework	5
2.3	Utilisations des modes pas à pas.....	6
2.4	Utilisation du point d'arrêt (breakpoint).....	6
2.4.1	Création d'un point d'arrêt.....	6
2.4.2	Comment conditionner un point d'arrêt.....	6
2.5	Utilisation des points de trace.....	7
2.6	Utilisation des espions.....	7
2.7	Repérer la version en cours de développement (Controlling Release Code Size)	7
2.8	Obtenir des messages d'erreur complets	7
3	Utilisation de l'émulateur.....	8
3.1	Configuration de l'émulateur	8
3.2	Démarrer manuellement l'émulateur	8
3.3	Utiliser et créer des skins pour l'émulateur	9
3.4	Les différents émulateurs disponibles	11
3.5	Fonctionnalités avancés d'émulation.....	11
3.5.1	Les simulateurs.....	11
3.5.2	Tests de sécurité.....	12
3.5.3	Utilitaires	13
4	Politique de test et débogage à adopter.....	14
5	Conclusion	15

1 Introduction

Tout d'abord nous verrons dans ce chapitre ce en quoi consistent le débogage et l'émulation dans le cas de développement pour Windows Mobile.

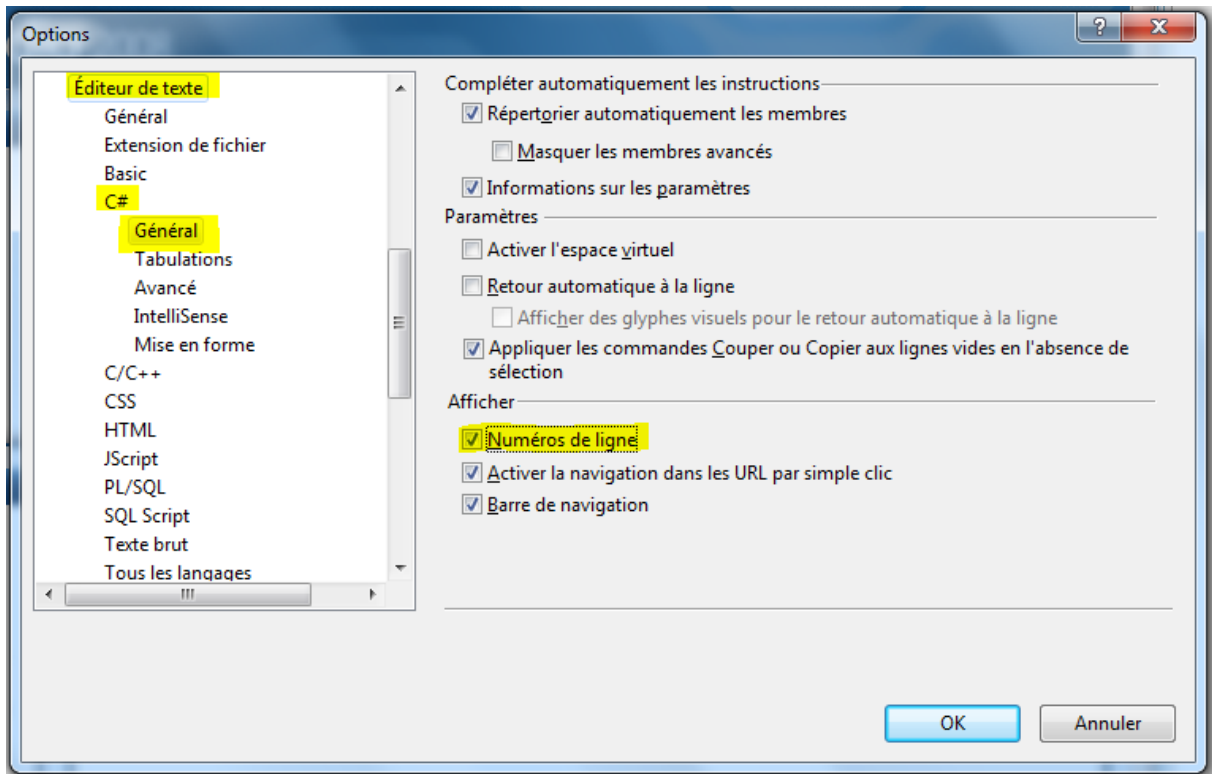
Le débogage consiste à utiliser différentes techniques pour trouver quelles erreurs sont présentes au niveau du code. Pour cela nous pouvons utiliser l'exécution pas à pas, les points d'arrêts, les points de trace, les assertions et les espions. Ces outils (dont certains apparus avec Visual studio 2008) sont en fait toute la force de cet environnement de développement. En effet, ils représentent un gain de productivité évident, grâce à eux nous pourrions analyser plus rapidement la cause des erreurs dans nos programmes et ainsi procéder à leurs corrections.

Ensuite il y a l'émulation dont l'intérêt est de tester l'application développée sans devoir la transférer et réinstaller à chaque modification, il s'agit d'une interface reproduisant celle d'un Windows Mobile et lors de chaque test, le fichier .cab généré à la volée par Visual Studio est automatiquement installé, ce qui nous permet de tester instantanément le programme en évitant des clics répétitifs.

Enfin, il va de soi qu'il faut tester les applications avec ces outils même si elles ne semblent rencontrer aucun souci, cela afin de permettre de bien analyser leur fonctionnement et repérer les actions imprévues (exemple d'une lettre saisie par l'utilisateur alors qu'un chiffre est attendu) ou bien de devancer d'éventuelles évolutions du programme pour éviter de recréer tout le code au moindre ajout de fonctions (maintenabilité du programme).

2 Détecter et corriger des erreurs de programmation

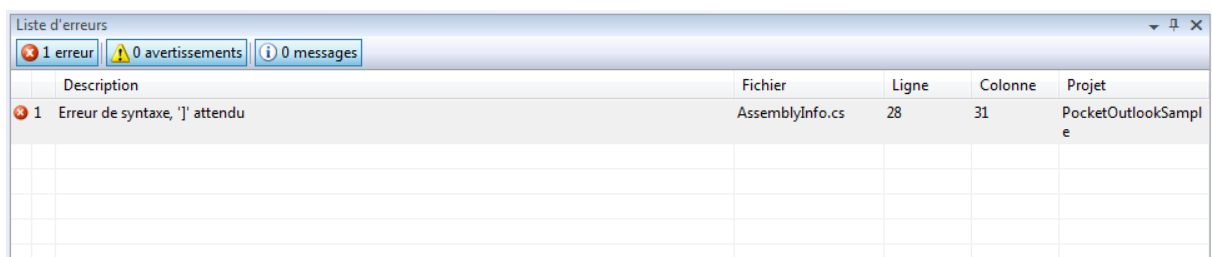
Le point commun entre toutes ces techniques de débogage c'est qu'elles s'appuient sur le numéro de ligne. Pour nous repérer il existe deux méthodes : le numéro de ligne apparaissant en bas à droite de la fenêtre de Visual Studio variant selon la position du curseur, ou bien afficher à gauche du code les numéros de ligne (paramétrage spécifique à chaque langage et pour le C# par exemple il faut se rendre dans : Outils > Options > Editeur de texte > C# > Général > Afficher > Numéros de ligne).



2.1 Utilisation de la liste des erreurs

Le mécanisme listant les erreurs fonctionne en temps réel et permet de détecter les erreurs de syntaxe sans avoir à compiler le code, par défaut elle est située en dessous de la fenêtre d'écriture du code. Etant donné cette détection instantanée d'erreurs il arrivera fréquemment que des « erreurs » apparaissent alors que nous n'avons pas fini d'écrire notre ligne de code.

De plus nous pouvons afficher ce que nous voulons en cliquant sur 'erreur', 'avertissements' ou 'messages' et en cas de détection de problème la fenêtre indique le numéro de ligne, de colonne, le fichier concerné et le projet le contenant, ainsi on peut rapidement retrouver la source de notre erreur.



Remarque : Si pour une raison quelconque elle n'apparaît pas nous pouvons l'afficher en allant dans Affichage > Liste d'erreurs.

2.2 Utilisation du débogueur

2.2.1 Entrer en mode débogage

Il est utile de rappeler qu'il faut compiler en mode debug, sans quoi quasiment aucune fonctionnalité ne sera accessible. Si nous utilisons des bibliothèques il faudra utiliser les versions debug de celles-ci.

2.2.2 Différences entre le .Net Compact Framework et le full .Net Framework

Les débogueurs du Compact Framework peuvent pratiquement effectuer les mêmes opérations que ceux du Full Framework excepté ce qui est listé ci-dessous :

- Fonctionnalité Modifier & Continuer : pas de modification de code durant le débogage, il faut d'abord l'arrêter, modifier le code, puis le relancer. Si nous tentons de modifier le code, le débogueur nous avertira.

- L'évaluation de fonction : Le débogueur d'appareil natif ne peut pas effectuer l'évaluation de fonction : nous ne pouvons pas taper d'expression qui contient une fonction et qui soit évaluée avec les résultats retournés, alors qu'avec le débogueur d'appareil managé nous pouvons le faire.

- Limitations du débogage d'interopérabilité : Nous ne pouvons pas déboguer à la fois code natif et code managé dans une même instance du débogueur. Pour pouvoir le faire il faut définir des points d'arrêt dans chaque section à partir desquels nous exécutons notre code pas à pas. Associons à chaque section de code le débogueur approprié, ainsi il changera automatiquement à chaque point d'arrêt. Actuellement, l'utilisation simultanée de deux instances de débogage sur le même processus n'est pas possible.

- Le débogage d'attribut : Le .NET Compact Framework ne peut pas effectuer de débogage basé sur les attributs, ce qui fait que nous ne pourrions pas définir d'attributs pour les visualisateurs.

- Le débogage de projets bureautiques : L'utilisation du débogueur Smart Device pour déboguer les applications écrites pour le bureau est impossible.

- Le débogage de noyau : L'utilisation du débogueur Smart Device pour le débogage de noyau est impossible.

- Mode de débogage Uniquement mon code : Ce mode est aussi impossible.

2.3 Utilisations des modes pas à pas

Il existe plusieurs modes pas à pas. En fait, il s'agit des différentes façons d'explorer le code : soit en pas à pas détaillé (F11), soit en pas à pas principal (F10). Le premier mode est comme son nom l'indique détaillé, il va rentrer par exemple dans les appels de fonctions, tandis que le second ne quittera pas la fonction courante même s'il voit un appel de fonction.

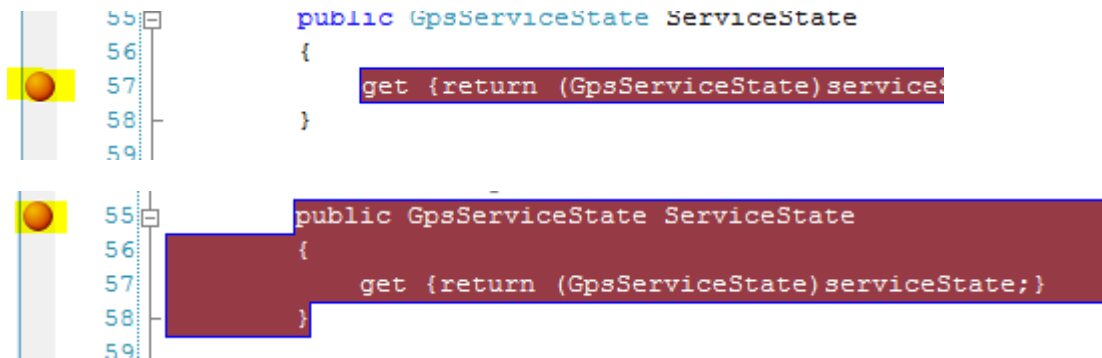
2.4 Utilisation du point d'arrêt (breakpoint)

Le point d'arrêt permet d'indiquer au débogueur de se mettre en mode arrêt, ainsi l'exécution du programme et son débogage est stoppée, cela permet notamment l'utilisation des espions (que nous verrons plus bas) qui ne peuvent être utilisés que durant l'arrêt.

Nous pouvons afficher la fenêtre 'Points d'arrêt' en allant dans Débuguer > Fenêtres > points d'arrêt, ce qui nous permet de lister l'ensemble des points d'arrêts créés et de les éditer plus rapidement.

2.4.1 Création d'un point d'arrêt

Pour créer un point d'arrêt il suffit de presser F9 (après avoir placé le curseur dans la ligne concernée) ou cliquer à gauche de la ligne (dans la bande grise) que l'on ne veut pas exécuter avant la fin de la « pause ». Un rond rouge apparaît pour indiquer sa création et la(les) ligne(s) de code concernée(s) se colorent en rouge.



Nous pouvons ôter rapidement tous les points d'arrêt en allant dans Débuguer > Supprimer tous les points d'arrêt ou en pressant Ctrl + Maj. + F9.

2.4.2 Comment conditionner un point d'arrêt

Les points d'arrêts étant déjà très pratiques, nous pouvons également leur associer une condition qui, si elle n'est pas remplie, ne rend pas actif le point d'arrêt. Prenons comme exemple le cas de boucles où seul le 3^{ème} « tour » nous intéresse : d'abord, en cliquant droit sur le rond rouge créé, puis en cliquant sur 'Condition...' ou directement sur un des points d'arrêts apparaissant dans la fenêtre 'Points d'arrêts'. Puis, dans la fenêtre 'Condition de point d'arrêt' écrivons une expression valide et sélectionnons « est true » pour que le point d'arrêt soit effectif lorsque l'expression est vérifiée, ou « est modifiée » pour qu'il soit effectif lorsque l'expression est modifiée.

2.5 Utilisation des points de trace

Les points de trace ont été ajoutés à Visual Studio dans sa dernière version, ils fonctionnent comme les points d'arrêt : ils stoppent l'exécution et lancent une action personnalisée.

Ils sont souvent utilisés pour afficher un message informant quel point a été atteint. Nous pouvons également les utiliser pour grande partie d'actions permises par trace mais cette fois sans avoir à modifier le code ; ces actions ne seront actives qu'en mode débogage.

2.6 Utilisation des espions

Les espions permettent de saisir une expression (variable, calcul de variables) et de visualiser dans la fenêtre « espions » les modifications de celle-ci durant l'exécution du programme.

Pour afficher la fenêtre il faut être en mode arrêt et aller dans Déboguer > Espion express, ensuite pour définir un nouvel espion cliquons sur un nom de variable dans la source, celle-ci se place automatiquement dans la boîte de dialogue, et pour confirmer cliquons alors sur 'ajouter un espion'.

2.7 Repérer la version en cours de développement (Controlling Release Code Size)

Cette fonction permet de mettre en évidence qu'un programme est en cours de développement. En effet lors de la compilation pour le débogage, Visual Studio définit la variable DEBUG, ce qu'il ne fait pas lors de la compilation pour la version finale. Il suffit alors d'intégrer du code pour montrer que l'application est en mode débogage :

```
//C#  
#if DEBUG  
MessageBox.Show("WARNING - this is the debug build");  
#endif
```

Ainsi le code sera compilé seulement lors de la création de la version debug.

2.8 Obtenir des messages d'erreur complets

Dans le Framework complet .Net, un message descriptif est associé à chaque exception. Pour des raisons de gain de place, les DLL de la version compacte du Framework ne contiennent pas l'ensemble de ces messages. Par contre, les messages d'erreur sont contenus dans des ressources téléchargeables si besoin.

La DLL est cependant volumineuse (environ 80 KB) donc à utiliser dans le cadre du développement et des tests. En mode production, celle-ci sera supprimée.

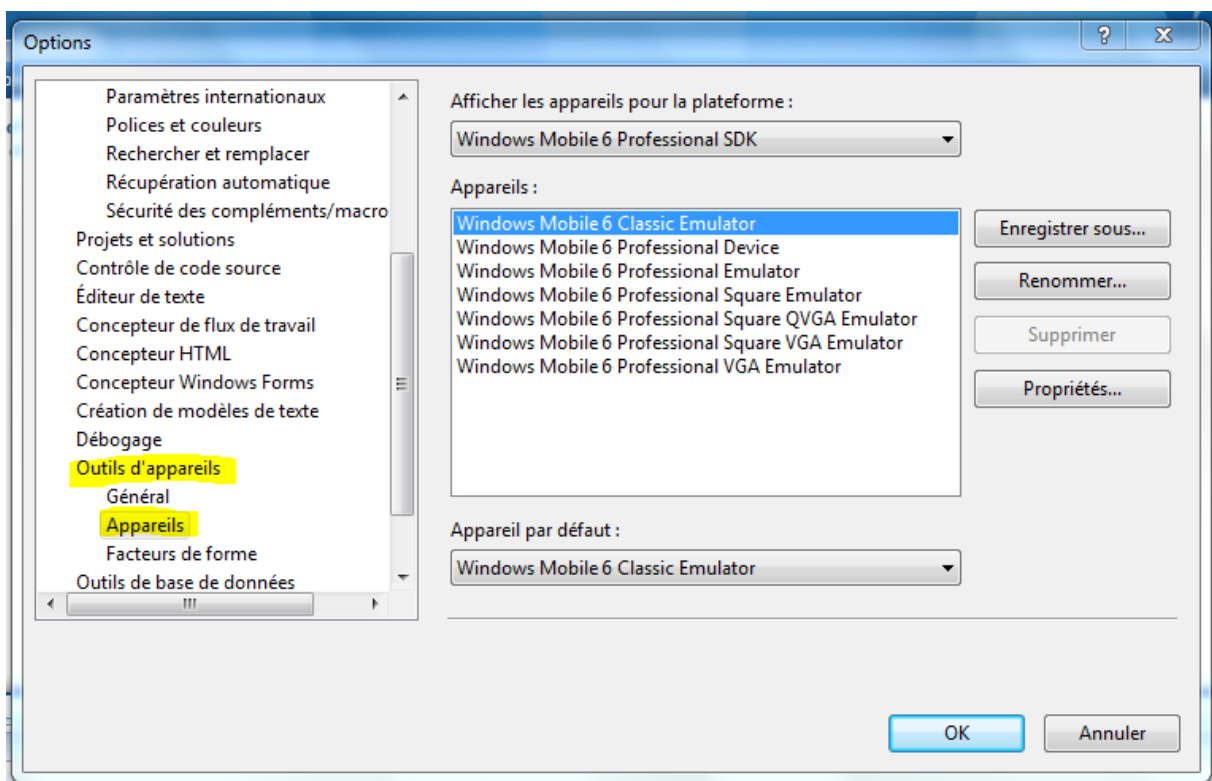
3 Utilisation de l'émulateur

L'émulateur simule un terminal Windows Mobile. L'intérêt principal réside dans le fait qu'il permet d'éviter d'incessants allers-retours entre l'ordinateur et le terminal. En effet, Visual Studio envoie à la machine virtuelle le programme développé pour ensuite l'utiliser directement et voir si par exemple il y a des incohérences dans l'affichage.

De plus, de nombreux outils sont fournis et grâce à eux nous pouvons simuler la réception d'un appel, d'un sms, d'une connexion wifi, d'appuis intempestifs sur l'écran, (cas de terminal à écran tactile), de GPS, etc. En somme, tout une gamme d'évènements qu'il serait contraignants d'utiliser sur un terminal physique.

3.1 Configuration de l'émulateur

Pour accéder au panneau de configuration des émulateurs il faut aller dans Outils > Options > Outils d'appareils > Appareils :



Différentes options sont ainsi accessibles, laissons les configurations par défaut qui conviendront dans la plupart des cas.

3.2 Démarrer manuellement l'émulateur

Il est possible également de démarrer l'émulateur de Windows Mobile séparément sans passer par Visual Studio : pour cela, il faut exécuter DeviceEmulator.exe (ou Emulator.exe) situé dans C:\Program Files\Microsoft Device Emulator\1.0 en lui associant des paramètres. Pour connaître leurs détails, il faut simplement lancer l'application qui fera la liste des possibilités offertes.

Il ne nous reste plus qu'à lancer l'application en ajoutant les paramètres voulus et nous obtenons un Mobile Windows émulé.

3.3 Utiliser et créer des skins pour l'émulateur

L'interface peut être personnalisée en créant des skins, cela nous permet de donner à l'émulateur l'apparence du terminal voulu. Par exemple, pour le faire ressembler au terminal réel.

Pour cela, nous utilisons un fichier XML ainsi que 3 images au format bitmap. Voici par exemple le code d'un thème par défaut :

```
//XML

<?xml version="1.0" encoding="utf-8" ?>
<skin>
  <view
    titleBar ="Windows Mobile 6 Classic"
    displayPosX="55"
    displayPosY="67"
    displayWidth="240"
    displayHeight="320"
    displayDepth="16"
    mappingImage="pocket_pc_emulator_mask.png"
    normalImage="pocket_pc_emulator_up.png"
    downImage="pocket_pc_emulator_down.png">
    <button
      tooltip="Power"
      onPressAndHold="0x75"
      mappingColor="0x487710"
    />
    .
    .
    .
    .
    <button
      tooltip="Soft Key 1"
      onClick="0x3B"
      mappingColor="0xDC2C1E"
    />
  </view>
</skin>
```

Attributs de l'élément <view>

Attribut	Signification
titleBar	texte qui apparaît dans le titre de la fenêtre
displayPosX, displayPosY	coordonnées du bord droit supérieur de l'affichage de l'émulateur exprimé en pixels
displayWidth, displayHeight	taille de l'affichage de l'émulateur exprimée en pixels. Pour la largeur, choisir une valeur comprise entre 80 et 1024. Pour la hauteur, choisir une valeur entre 64 et 768. Les tailles doivent être un multiple de 8.
displayDepth	profondeur de couleurs (8, 16, ou 32 bits par pixel).
normalImage	fichier contenant une image avec les boutons relevés (unselected).
mappingImage	fichier contenant une image dont les couleurs définissent les positions de boutons
downImage	fichier contenant une image avec les boutons enfoncés (selected).

Remarque : Chaque élément <button> définit un bouton physique simulé.

Attributs de l'élément <button>

Attribut	Signification
toolTip	texte qui apparaît lorsque le bouton est survolé
onClick	définit ce qui arrive lorsque le bouton est enfoncé
onPressAndHold	définit ce qui arrive lorsque le bouton est enfoncé durant un moment
mappingColor	couleur utilisée dans « mappingImage » pour définir le bouton. Tous les pixels de cette couleur dans l'image seront considérés comme faisant partie du bouton.

Images exemples de skins



normalImage



mappingImage



downImage

Chacune des images ci-dessus a une fonction bien précise : "normalImage" représente le terminal à l'état neutre ; dans "mappingImage" chaque couleur représente la zone du bouton (en lien avec mappingColor vu ci-dessus) ; et enfin "downImage" qui représente chaque bouton une fois enfoncé.

La façon la plus simple d'arriver à créer votre propre skin est de démarrer à partir d'un modèle préexistant et de l'adapter afin d'obtenir le résultat escompté.

3.4 Les différents émulateurs disponibles

Il a été prévu une grande variété d'émulateurs permettant de couvrir à peu près tous les types de terminaux disponibles en fonction du format et de la résolution de l'écran, et du fait que nombre d'entre eux ont des fonctions spécifiques. Voici une liste d'entre eux :

- Classic emulator
- Professional device (qui est en fait un faux émulateur et permet d'utiliser un terminal Windows Mobile réel comme plateforme de tests)
- Professional emulator
- Professional Square emulator
- Professional Square QVGA emulator
- Professional Square VGA emulator
- Professional VGA emulator

3.5 Fonctionnalités avancés d'émulation

3.5.1 Les simulateurs

3.5.1.1 Pour les fonctions liées au téléphone

Cellular Emulator est un utilitaire qui permet de simuler des appels, des sms reçus, le passage de la 2G à la 3G, la perte de connexion GPRS, la réception de commandes AT.

3.5.1.2 Pour les fonctions liées au GPS

FakeGPS est un utilitaire qui permet de recevoir des signaux GPS simulés quand il n'y a pas de récepteur GPS réel connecté. Les données sont lues à partir de fichiers NMEA de type .txt tels que dixies.txt : envoi immédiat de trames NMEA avec une position GPS.

Fakegpsdata.txt : comme dixies.txt mais prend plus de temps avant de dévoiler un nouvel emplacement.

Les deux fichiers sont en fait constitués de la même façon, ce qui diffère c'est la façon dont ils sont lus. Ci-dessous un exemple de fichier type :

```
$GPGLL,4738.0173,N,12211.1874,W,191934.767,A*21
$GPGSA,A,3,08,27,10,28,13,19,,,,,,,,,2.6,1.4,2.3*3E
$GPGSV,3,1,9,8,71,307,43,27,78,59,41,3,21,47,0,10,26,283,40*77
$GPGSV,3,2,9,29,13,317,0,28,37,226,37,13,
 32,155,36,19,37,79,42*42
$GPGSV,3,3,9,134,0,0,0*46
$GPRMC,191934.767,A,4738.0173,N,12211.1874,W,0.109623,12.14,291004,,*21
$GPGGA,191935.767,4738.0172,N,12211.1874,W,1,06,1.4,32.9,M,-
17.2,M,0.0,0000*75 $GPGLL,4738.0172,N,
12211.1874,W,191935.767,A*21
$GPGSA,A,3,08,27,10,28,13,19,,,,,,,,,2.6,1.4,2.3*3E
$GPRMC,191935.767,A,4738.0172,N,12211.1874,W,
0.081611,15.81,291004,,*2A
```

3.5.2 Tests de sécurité

3.5.2.1 RapiSecurity

Tout d'abord, il faut savoir que RAPI (Remote API) permet à des applications lancées sur un ordinateur d'agir sur un périphérique Windows Mobile à distance. RAPI a la capacité de manipuler le système de fichier, incluant la création et la suppression de fichiers et de répertoires. Les fonctions RAPI peuvent être également utilisées pour créer et modifier des bases de données incluses dans le terminal. RAPI permet aussi de lire et modifier des clefs registres, autant que de lancer des applications et invoquer des méthodes sur le terminal distant.

Cela peut par contre provoquer des problèmes de sécurité car la machine dispose finalement d'un accès total au terminal, il existe pour cela un système de règles définissant la politique de sécurité à appliquer.

Au cours du développement, il peut être utile de tester différentes politiques afin de rendre le système plus ou moins permissif pour mettre en valeur la source du problème.

RapiSecurity permet ainsi de modifier la politique RAPI à la volée d'un terminal Windows Mobile en utilisant des fichiers type XML en .CPF contenant des informations de configuration.

3.5.2.2 SDK Development Certificates

Il s'agit de certificats qui sont installés sur le mobile et qui permettent de tester les applications durant leur développement. En effet pour qu'une application puisse être déployée sur un terminal Windows Mobile il faut qu'elle soit signée par un certificat reconnu par l'appareil, si ce n'est pas le cas un message signale le problème et empêche l'exécution de l'application. Pour contrer cela Visual Studio signe les applications avec son propre certificat, et installe sur le terminal les certificats correspondants.

3.5.2.3 Security Powertoy

Security Powertoy est un utilitaire de configuration à distance qui s'exécute sur une machine hôte et vous permet sur votre périphérique Windows Mobile de :

- Examiner la configuration de sécurité
- Installer une configuration standard de sécurité
- Sauvegarder la configuration de sécurité
- Ajouter un certificat de développement
- Signer un fichier avec un certificat
- Vérifier la signature d'un fichier

3.5.2.4 RapiConfig

RapiConfig.exe est une application pour l'ordinateur qui permet d'exécuter des fichiers XML sur un terminal Windows mobile émulé ou connecté par le biais d'ActiveSync. Il s'utilise en ligne de commande : `RAPICONFIG [/M] [/V] nomDuFichier`

Expliquons les paramètres :

`/M` : renvoie les Métadonnées

`/V` : valide le document. Aucun changement n'est effectué

`nomdufichier` : Indique le chemin du fichier XML

3.5.2.5 CabSignTool

CabSign Tool est une application qui permet de signer l'exécutable de l'application et les différents fichiers d'installation avec un certificat correspondant à votre société et ce pour des raisons de sécurité. Entre autres les fichiers *.dll, *.exe ainsi que le fichier .cab qui installe l'application doivent être certifiés ce qui peut être long si l'on devait le faire pour chaque fichier. Ce qui permet cet utilitaire c'est de traiter rapidement ces signatures.

L'utilitaire s'utilise de la façon suivante : nous créons notre application et son .cab d'installation comme habituellement sans nous occuper de la signature. Lors du traitement par CabSign Tool, il défait le package, signe chacun des fichiers, recrée le package et signe le fichier .cab contenant désormais la version signée des exécutables. Il permet également de signer tous les fichiers avec le même certificat ou séparer celui signant le .cab de celui des exécutables.

3.5.2.6 Revoke

Cet outil génère le hachage d'un certificat ou d'une application, créant ainsi un document XML le contenant, à disposition du périphérique pour révoquer l'application ou un certificat.

3.5.3 Utilitaires

3.5.3.1 Hopper

Les logiciels sont souvent lancés de façon continue sur les périphériques Windows Mobile. Leur ouverture prolongée provoque donc parfois des problèmes. Avec des tests classiques on ne peut que difficilement les détecter. Pour trouver l'origine de ces problèmes il faut recréer le même type de contrainte que lors d'un usage intensif. L'utilitaire permet cette mise en situation.

Son fonctionnement consiste à effectuer un nombre important de clics de façon aléatoire dans l'application, ainsi on retrouve les mêmes contraintes que pendant le déploiement. Nous pouvons aussi l'utiliser avec plusieurs applications et ainsi cela provoque des changements d'application au cours des clics créant une situation encore plus réaliste.

L'utilitaire permet non seulement de tester l'application mais également de ne pas créer une instabilité du terminal.

3.5.3.2 Windows Mobile Test Framework

Le Windows Mobile Test Framework contient une collection de classes, interfaces et valeurs type qui permettent à notre code de test d'accéder aux contrôles et ressources sur un terminal Windows Mobile.

4 Politique de test et débogage à adopter

L'environnement d'émulation est vraiment utile, particulièrement quand nous ne disposons pas d'une grande variété de matériels sur lesquels tester les programmes. Les différentes options des émulateurs permettent de tester les applications dans différents environnements (différences de mémoire vive, de périphériques, etc...).

Ce type d'utilitaires est particulièrement utile pour développer sur une grande variété de terminaux, pour tester les applications sur de nombreuses plateformes avant la sortie finale.

D'un autre côté, l'émulateur est assez lent comparé à une machine réelle. La différence entre les deux au niveau de la vitesse est difficile à quantifier : tout dépend du système hôte, de sa vitesse et si d'autres applications sont lancées simultanément. Dans certains cas les applications pourront fonctionner lentement alors qu'en production tout fonctionne correctement, cela se produira surtout avec les applications gourmandes en ressources comme celles basées sur le temps (type chronomètre).

De plus, l'émulateur n'est qu'un simulateur et n'égale pas le vrai matériel. La connexion des ports virtuels série et parallèle à des ports réels présents sur une machine hôte, ainsi qu'une connexion Ethernet peut être également émulée. Si d'autres éléments sont nécessaires, l'émulateur ne répondra pas à nos besoins. En conséquence, il sera utile seulement s'il est suffisamment proche de la plateforme finale.

Il peut être intéressant de créer séparément l'interface du reste de l'application. Cette façon de faire pourra nous faire gagner un temps précieux lors de la détection d'erreurs car le code s'en retrouvera simplifié.

Nous pourrions trouver également utile de créer l'interface avec le Framework complet .NET et par la suite porter l'application sous le Framework Compact .NET. Cela donnera toute la puissance et les avantages de l'environnement complet. Mais rappelons que le Framework Compact ne contient pas les mêmes possibilités que le Framework complet.

5 Conclusion

Comme nous avons pu le voir il est possible de façon relativement simple de tester une application et de s'assurer de son bon fonctionnement avant tout déploiement. Visual studio mettant à notre disposition tous les outils nécessaires.

Il faut bien dissocier les deux étapes, débbugger consiste à trouver des erreurs dans le code (de type syntaxique) alors qu'émuler permet principalement de tester le bon fonctionnement du logiciel en lien avec toutes les interactions qu'il aura avec l'utilisateur ou les échanges d'informations avec le gps, la puce gsm...