Versions de débogage et de déploiement

Objectif

Jusque-là nous avons testé les exemples dans ADL, le simulateur AIR inclus dans Flex SDK.

Ce nouveau chapitre décrit les étapes principales à mettre en œuvre pour générer des applications multi-plateformes déployables sur les mobiles réels. Traitant d'abord des versions de débogage, nous aborderons ensuite le déploiement en décrivant la génération d'une version finale dite *release*.

FICHIERS DE DESCRIPTION

Le comportement de l'application n'est pas déterminé uniquement par du code MXML ou ActionScript. Il dépend également des réglages précisés dans les fichiers de description. Avant de déployer une application, même pour une première série de test, il convient d'étudier ces fichiers d'un peu plus près.

Introduction

Nous avons déjà abordé le sujet des fichiers de description dans le chapitre 2. La plupart des choix effectués lors de la création d'un nouveau projet sont enregistrés dans les fichiers de description. Ils seront incorporés au final dans le package destiné au déploiement de l'application sur la plateforme cible.

L'assistant de création de projets ne s'exécute qu'une fois ... lors de la création du projet. Il est nécessaire de savoir éditer ultérieurement les fichiers de description afin d'adapter les réglages de l'application au fur et à mesure de son évolution. La

documentation correspondante se trouve bien entendu dans le matériel fourni par Adobe, mais également dans la documentation propre à chaque plateforme.

Les déclarations contenues dans le fichier de description peuvent se décomposer en trois groupes, du plus général au plus spécifique :

- **des informations sur le comportement de l'application**, nécessaires pour le lecteur AIR et identiques à celles requises pour les applications « bureau » ;
- **des informations spécifiques au déploiement d'une application AIR** sur mobile ;
- **des informations propres à la plateforme ciblée** par le déploiement.

Par exemple, pour effectuer une requête vers un serveur distant, il faut disposer des droits d'accès à Internet :

- cela se configure pour Android et BlackBerry Playbook dans un fichier de description (*descriptor file*) situé dans le dossier **src** du projet.
- pour iOS, les droits sont demandés lors de l'exécution : ainsi également pour l'autorisation de géolocaliser l'appareil[1].

Génération automatique

Lorsque Flex génère un projet pour déploiement avec AIR, il crée dans le dossier **src**, un fichier XML de description {**nomapplication**}**-app.xml** où `{nomapplication}` est remplacé par le nom du projet courant. Pour une application nommée **MobileSample**, le fichier de description se nomme **MobileSample-app.xml**.

Ce fichier de description est requis pour toute application AIR, pas seulement pour les projets mobiles. Vous y trouvez à la fois des paramètres communs à tous les types de projets et des sections spécifiques aux mobiles Android et iOS.

Pour la tablette Blackberry Playbook, le fichier descripteur est distinct et se nomme toujours **blackberry-tablet.xml**. Il est également stocké dans le dossier **src**. Flash Builder le crée automatiquement si l'ardoise numérique de RIM figure dans les cibles du projet.

Les indispensables

J'ai inséré ci-après quelques commentaires sur le contenu du fichier de description.

Version d'AIR

Les déclarations débutant le fichier {**nomapplication**}**-app.xml** sont essentielles à la bonne exécution de l'application sur le mobile.

1. Le chapitre 2 traite également des options relatives aux permissions dans les fichiers de description.

```
<application xmlns="http://ns.adobe.com/air/application/2.6">
<!-- Adobe AIR Application Descriptor File Template.
Specifies parameters for identifying, installing, and launching AIR
applications.
xmlns - The Adobe AIR namespace: http://ns.adobe.com/air/application/2.6
The last segment of the namespace specifies the version
of the AIR runtime required for this application to run.
```

Il est obligatoire que le numéro de version à la fin de l'espace de noms corresponde au *runtime* AIR ciblé (dans la chaîne ci-dessus : 2.6 correspond à la première version commerciale de Flex 4.5 datant de mai 2011).

Notez que cette déclaration détermine les fonctionnalités accessibles même si un *runtime* AIR de version supérieure est présent sur le périphérique où s'exécute l'application. À l'inverse, si la version du mobile est antérieure à celle requise, un message propose à l'utilisateur de télécharger le *runtime* spécifié, comme dans la figure 1.

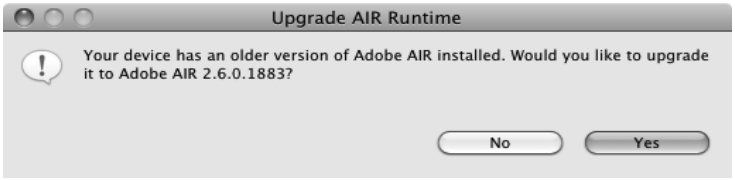

Figure 1 — Upgrade AIR Runtime

Dans les pré-versions, il était fréquent de devoir fixer cette valeur manuellement pour suivre les évolutions du SDK. Même si nous pouvons imaginer que dorénavant Flash Builder s'en charge, il est préférable de connaître le rôle de cette chaîne.

Identifiant de l'application

```
minimumPatchLevel - The minimum patch level of the AIR runtime required to run
the application. Optional.
-->
<!-- A universally unique application identifier. Must be unique across all
AIR applications.
Using a reverse DNS-style name as the id is recommended. (Eg.
com.example.ExampleApplication.) Required. -->
<id>MobileSample</id>
```

Dans la valeur de l'élément `<id>`, il est conseillé d'utiliser un identifiant unique du style `<id>com.ckti.mobileSample</id>`, c'est-à-dire préfixant le nom de l'application par votre nom de domaine inversé, ce que l'on nomme **reverse DNS-style**. Cette chaîne permettra d'identifier de manière unique votre application sur le périphérique, mais aussi dans les différents magasins en ligne.

Sur Android, le contenu de la propriété ID est préfixé lors de la génération par **air** afin de déterminer le nom de package. L'exemple précédent fournira au final le package **air.com.ckti.mobileSample**.

Pour iOS, il faut que l'identifiant spécifié ici corresponde à la deuxième partie de l'**app ID**. Cet app ID, renseigné dans le portail d'approvisionnement d'Apple iOS, se compose de deux parties : une racine et un identifiant de *bundle*.

Prenons une racine valant `SD7MMAD9FM6` et considérons les trois cas qui peuvent se présenter pour l'app ID :

- `SD7MMAD9FM6.com.ckti.Mobilesample`
  - il faut utiliser `com.ckti.MobileSample` comme valeur de l'élément `<id>`
- `SD7MMAD9FM6.com.ckti.*`
  - toute chaîne commençant par `com.ckti.` conviendra
- `SD7MMAD9FM6.*`
  - la valeur de `<id>` est libre

Il est ainsi tout à fait possible de déclarer le même identifiant sur iOS et Android. Vérifiez soigneusement l'ID avant de soumettre une application sur un magasin. Vous ne pourrez plus le modifier ultérieurement sans supprimer au préalable votre application et soumettre une **nouvelle** application. De plus cette information apparaît à plusieurs endroits à l'utilisateur.

Nom de l'application

Les noms des options suivantes sont employés pour désigner l'application dans l'interface, restez cohérents.

Attention, si vous renommez le projet dans Flash Builder, le nom de l'application n'est pas modifié dans le fichier descripteur.

```
<!-- Used as the filename for the application. Required. -->
<filename>MobileSample</filename>
<!-- The name that is displayed in the AIR application installer.
May have multiple values for each language. See samples or xsd schema file.
Optional. -->
<name>MobileSample</name>
```

Numéro de version

Prenez soin de bien renseigner et incrémenter le numéro de version, en particulier, si vous déployez l'application au travers d'un magasin en ligne (App Store, Android Market...).

```
<!-- A string value of the format <0-999>.<0-999>.<0-999> that represents
application version which can be used to check for application upgrade.
Values can also be 1-part or 2-part. It is not necessary to have a 3-part
value.
An updated version of application must have a versionNumber value higher than
the previous version. Required for namespace >= 2.5 . -->
<versionNumber>0.0.1</versionNumber>
```

Quelques remarques :

- Le numéro de version comprend trois parties, chacune d'elle peut comprendre trois chiffres. Vous n'êtes pas obligé de fournir un numéro composé de trois parties : 1 est valide tout comme 2.2.
- Pour une application distribuée au travers de l'Android Market, il faut également spécifier le champ `versionLabel`.
- Comme mentionné dans les commentaires XML, en cas de mise à jour, la nouvelle valeur de `versionNumber` doit être supérieure à l'ancienne.
- Cette dernière précaution est également à respecter pour l'installation sur iOS, même dans le cas de version de débogage. Incrémenter le numéro de version aide à s'assurer que la nouvelle version remplace bien l'ancienne.

Exemple :

```
<versionNumber>1.2.5<versionNumber>
<versionLabel>Version 2012 : 1.2 release 5</versionLabel>
```

Paramètres spécifiques des projets mobiles

Plus loin dans le fichier, vous trouvez d'autres paramètres spécifiques des projets mobiles :

```
<!-- The initial aspect ratio of the app when launched (either "portrait" or
"landscape"). Optional. Mobile only. Default is the natural orientation of the
device -->
<!-- <aspectRatio></aspectRatio> -->
<!-- Whether the app will begin auto-orienting on launch. Optional. Mobile
only. Default false -->
<!-- <autoOrients></autoOrients> -->
<!-- Whether the app launches in full screen. Optional. Mobile only. Default
false -->
<!-- <fullScreen></fullScreen> -->
<!-- The render mode for the app (either auto, cpu, or gpu). Optional. Mobile
only. Default auto -->
<!-- <renderMode></renderMode> →
```

Voici les propriétés :

- `aspectRatio` : affichage initial en mode portrait ou paysage.
- `autoOrients` : l'application doit-elle automatiquement basculer si l'utilisateur change l'orientation du périphérique ?

- `fullScreen` : détermine si l'affichage s'effectue plein écran ou sous la barre d'état du système qui occupera dans ce cas le haut de l'écran.
- `renderMode` : choix entre **GPU** (*Graphics Processing Unit*) et **CPU** (*Central Processing Unit*). La valeur par défaut est `auto` qui correspond pour le moment à CPU. Je vous conseille de réserver GPU à un usage avancé où vous maîtrisez la plateforme d'exécution. Si GPU ne peut effectuer le rendu d'un objet, il ne sera pas affiché du tout et comme certaines tablettes ne supportent pas le rendu GPU...

Exemple :

```
<autoOrients>true</autoOrients>
<fullScreen>true</fullScreen>
<visible>true</visible>
```

Certaines propriétés sont ignorées sur les mobiles, il en est ainsi des options concernant la taille et la transparence des fenêtres.

supportedProfiles

L'élément optionnel `supportedProfiles` déclare la liste des classes de périphériques sur lesquelles l'application peut être déployée. En son absence, le déploiement peut s'effectuer sur tout appareil prenant en charge AIR.

Les valeurs permises sont les suivantes :

- `desktop` – ordinateur de bureau ou portables.
- `extendedDesktop` – applications pour les ordinateurs de bureau packagées avec un installateur natif à la plateforme.
- `mobileDevice` – smartphones et tablettes, Android, BlackBerry Tablet OS et iOS.
- `extended mobile device` – non utilisé pour le moment.
- `tv` – déploiement sur les télévisions, périphériques tels que lecteurs Blu-ray, *set-top boxes*...
- `extendedTV` – identique au précédent, mais incluant des fonctionnalités supplémentaires.

Vous pouvez indiquer plusieurs valeurs séparées par un espace :

```
<supportedProfiles> desktop extendedDesktop mobileDevice </supportedProfiles>
```

Flash Builder tente de respecter ces réglages au moment de la génération de l'application, vous empêchant par exemple de générer un package `.apk` si vous n'avez pas inclus le profil `mobileDevice`.

Notez que pour les tests, ADL utilise le premier profil listé (quoique vous puissiez spécifier un autre profil en option de ligne de commande).

Emplacement de l'installation

À la différence des périphériques Apple, les mobiles Android sont souvent assez limités en capacité de stockage. Afin de ne pas saturer la mémoire interne, il est important de spécifier dans le fichier de description que l'application peut s'installer sur la mémoire externe (généralement la SDCard).

Pour cela, définissez l'attribut `installLocation` de l'élément `manifest` d'Android à `auto` ou `preferExternal`. Utiliser `internalOnly` empêche l'utilisateur de déplacer l'application sur la SDCard.

```
<android>
      <manifestAdditions><![CDATA[
            <manifest android:installLocation="preferExternal">
      ...
            </manifest>
      ]]></manifestAdditions>
```

Icônes pour l'application

Le tableau 1 récapitule les tailles d'icônes utilisées pour chaque plateforme mobile[1].

Tableau 1 — Taille des icônes selon la plateforme

Taille d'icône en pixels	Plateforme
29x29	iOS
36x36	Android
48x48	Android, iOS
57x57	iOS
72x72	Android, iOS
76x76	BlackBerry Playbook
114x114	iOS
512x512	Android, iOS

Pour iOS et Android, le chemin d'accès aux fichiers d'icônes doit s'indiquer dans l'élément `icon` du fichier de description :

```
<icon>
    <image36x36>assets/icon36.png</image36x36>
    <image48x48>assets/icon48.png</image48x48>
    <image72x72>assets/icon72.png</image72x72>
</icon>
```

Pour BlackBerry Tablet OS, la spécification de l'icône de l'application s'effectue dans le fichier **blackberry-app.xml** :

1. Cf. l'excellent article sur le site *savagelook* : http://savagelook.com/blog/android/mobile-developers-icon-image-checklist

```
<qnx>
   <initialWindow>
      <systemChrome>none</systemChrome>
      <transparent>false</transparent>
   </initialWindow>
   <icon>
      <image>icons/icon_86.png</image>
   </icon>
```

Si une taille requise pour une plateforme n'est pas référencée, l'icône de plus grande taille approchante sera mise à l'échelle pour être utilisée en remplacement.

Une bonne technique consiste à partir de la version 512 × 512 pixels puis de la mettre à l'échelle pour les autres tailles requises tant que l'aspect demeure acceptable. Pour les plus petites dimensions, il est en général nécessaire de redessiner une version spécifique.

Filtrage de compatibilité (Android uniquement)

Les sous-éléments `<uses-permission>` de l'élément `<manifest>` offrent la possibilité de vérifier les pré-requis de l'application avant l'installation sur le périphérique.

Si vous exploitez dans l'application l'interface tactile en mode *multi-touch*, il est par exemple conseillé de vérifier sa disponibilité sur le périphérique cible à l'aide de la ligne suivante :

```
<uses-feature android:required="true"
android:name="android.hardware.touchscreen.multitouch"/>
```

Cette déclaration s'utilise également pour éviter un filtrage trop drastique sur l'Android Market. Si votre application déclare utiliser l'appareil photo comme dans la déclaration ci-dessous, le magasin Android estimera que vous avez besoin de TOUTES les options relatives à la caméra :

```
<manifest><uses-permission android:name="android.permission.CAMERA" />
```

Un appareil disposant d'un appareil photo, mais pas de l'autofocus, ne pourra pas installer l'application.

Pour assouplir ce filtrage, il faut déclarer les sous-ensembles de fonctionnalités comme optionnels :

```
<uses-feature android:name="android.hardware.camera"
android:required="false"/>
<uses-feature android:name="android.hardware.camera.autofocus"
android:required="false"/>
<uses-feature android:name="android.hardware.camera.flash"
android:required="false"/>
```

Suivant la plateforme

AndroidManifest.xml

Les autorisations pour la plateforme Android se déclarent comme sous-éléments d'un élément optionnel `<android>`.

Les autorisations exprimées seront transmises par l'outil de génération ADT vers le fichier **AndroidManifest.xml**. Ce dernier est requis par les spécifications de Google afin de construire le package `.apk` qui sera installé sur le périphérique.

Flex remplit un sous-ensemble des entrées possibles, le minimum requis pour que l'application puisse s'exécuter. Vous pouvez ajouter d'autres entrées dans cette section, ADT les transmettra également au fichier **AndroidManifest.xml**[1].

Autorisations pour Android

Lors de l'installation sur un périphérique Android, chaque application doit demander à l'utilisateur de confirmer les autorisations d'exécution requises. La liste des autorisations à soumettre à l'utilisateur est déclarée dans le fichier de description, section `<android><manifestAdditions>`.

Section Apple iOS

Pour les applications à destination d'iOS, un autre élément optionnel `<iPhone>` remplit le même objectif. Les informations déclarées ici sont ajoutées au fichier **Info.plist** de l'application générée pour iOS.

```
<iPhone>
       <!-- A list of plist key/value pairs to be added to the application
Info.plist -->
       <InfoAdditions>
           <![CDATA[
                <key>UIDeviceFamily</key>
                <array>
                    <string>1</string>
                    <string>2</string>
                </array>
                <key>UIStatusBarStyle</key>
                <string>UIStatusBarStyleBlackOpaque</string>
                <key>UIRequiresPersistentWiFi</key>
                <string>YES</string>
            ]]>
        </InfoAdditions>
       <requestedDisplayResolution>high</requestedDisplayResolution>
</iPhone>
```

Le tableau `<array>` sous l'entrée `<key>UIDeviceFamily</key>` indique les gammes de périphériques ciblées : **1** correspond à l'iPhone/iPod Touch, tandis que **2** désigne l'iPad. La déclaration ci-dessus générera une application compatible iPhone et iPad.

1. Certaines propriétés sont réservées, vous trouverez leur liste dans la documentation d'Adobe, http://help.adobe.com/en_US/air/build/WSfffb011ac560372f7e64a7f12cd2dd1867-8000.html.

Pour ne cibler que l'iPad, déclarez :

```
<array><string>2</string></array>
```

Afin de déterminer le mode d'affichage sur les périphériques dotés d'écran à haute résolution (iPhone4 et son écran *Retina Display*), renseignez l'élément `requestedDisplayResolution` :

- `high`, chaque pixel de l'écran peut être adressé. Les dimensions de la scène sont de 640 × 490.
- `standard`, à un pixel dessiné correspondent quatre pixels sur l'écran. Même sur un écran 640 × 490, les dimensions de la scène demeurent 320 × 480.

Pour une compatibilité avec la haute résolution, une icône de l'application en 114 × 114 pixels doit être fournie et son chemin d'accès indiqué dans le fichier de description :

```
<icon>
    <icon114x114>icons/icon114.png</icon114x114>
</icon>
```

Lorsque votre application est renvoyée à l'arrière-plan, iOS peut lui demander de rester active (fonctionnement par défaut depuis que le multitâche a été introduit dans le système d'Apple) ou de quitter. Ce fonctionnement peut se paramétrer par la chaîne suivante :

```
<key>UIApplicationExitsOnSuspend</key>
<true/>
```

BlackBerry TabletOS

Le déploiement vers les tablettes BlackBerry Playbook requiert la déclaration du fichier : **blackberry-tablet.xml**. Voici un exemple d'un tel fichier de description :

```
<qnx>
    <initialWindow>
        <systemChrome>none</systemChrome>
        <transparent>false</transparent>
    </initialWindow>
    <icon>
        <image>icons/icon_86.png</image>
    </icon>
    <publisher>CKTI</publisher>
    <category>core.internet</category>
    <splashscreen>assets/splash.png</splashscreen>
   <permission>access_internet</permission>
</qnx>
```

Le document *Development Guide* de RIM décrit les différentes informations gérables dans ce fichier.

Dans l'exemple précédent :

- `category` : la catégorie dans laquelle sera rangée l'application (exemple core.games, core.media...)
- `icon` : l'icône à utiliser sur la page d'accueil de la tablette (peut être du type png ou jpg et doit faire au maximum 86 × 86 pixels)
- `splashscreen` : image à utiliser au chargement de l'application. Peut se déclarer sous la forme `<splashscreen>splashLandscape.png:splahPortrait.png</splashscreen>`. Dans ce cas, le premier fichier avant les " :" sera utilisé en mode paysage, tandis que le second fichier sera utilisé en mode portrait.

GÉNÉRATION D'UNE VERSION DE TEST

Flash Builder offre la possibilité de générer :

- une version de débogage exécutée directement dans ADL sur l'ordinateur de développement ;
- une version de débogage exécutée sur le périphérique ;
- une version finale (*release*) à déployer sur le périphérique.

Dans ce paragraphe, nous considérons la génération d'une version de débogage. Le paragraphe *Version finale* décrira le cas de la version *release*.

Génération d'une version

Processus

La génération d'une version suit le déroulement suivant :

- compilation du swf ;
- rassemblement des différentes ressources (icônes, contenu du dossier assets...) ;
- traitement du fichier de description ;
- assemblage (*Package*) d'une archive pour la plateforme cible.

En effet, quoique les suffixes diffèrent suivant la plateforme, le fichier final est toujours une archive. Remplacer le suffixe par .zip rend possible la décompression de l'archive pour en vérifier le contenu.

Attention aux chargements de données

Si votre application charge des données depuis un serveur et que dans la phase de développement vous avez utilisé votre serveur local sur localhost, vous ne chargerez plus rien en passant sur le périphérique. En effet, localhost désignera alors l'adresse du périphérique lui-même et non plus celle de la machine de développement.

Évident ? « *Il y en a qui ont essayé : ils ont eu des problèmes...* »

Options de Packaging

La génération d'une version s'effectue en fonction des options spécifiées dans le groupe Flex Build Packaging. On accède à ce dialogue par les propriétés du projet (clic droit sur le nom du projet dans le Package Explorer).

Figure 2 — Flex Build Packaging

Pour chaque plateforme supportée par Flex, s'affiche une case à cocher indiquant si la cible correspondante doit être retenue pour ce projet.

Les différentes étapes de paramétrage correspondent globalement à celles proposées par Flash Builder lors de la génération de la version *release*. Reportez-vous au paragraphe *Version finale* pour une description des différentes options.

Un point spécifique cependant pour BlackBerry Tablet OS : la case à cocher « Add platform specific librairies to library path ». Elle indique s'il faut référencer les bibliothèques de code QNX spécifiques à la plateforme ou non. Décochée par défaut[1].

Mise au point et simulateurs

Automatisation suivant la plateforme

En fonction de la plateforme ciblée, l'installation d'une version, qu'elle soit de débogage ou finale, pourra être automatisée ou devra s'effectuer manuellement. Le tableau 2 reprend les différentes combinaisons.

Collaboration avec les simulateurs

- Seul l'émulateur du Playbook permet une installation directe depuis Flash Builder. De plus, l'émulateur propose une exécution très fluide et agréable.
- Le transfert vers l'émulateur Android fourni dans le SDK de Google est possible en ligne de commande. Cependant, la lenteur d'exécution de l'émulateur rend son utilisation assez pénible.

1. http://blog.everythingflex.com/2011/05/26/qnx-components-within-flex-for-blackberry/

Tableau 2 — Combinaison d'installations suivant la plateforme

	Débogage	Release	Émulateur/simulateur
iOS	Manuel	Manuel	Pas de possibilité d'installer un .ipa dans le simulateur iOS
Android	Automatique	Automatique	Installation manuelle possible d'un .apk spécialement généré en ligne de commande
Playbook	Automatique	Automatique	Automatique

- Enfin, dans le monde Apple seul l'outil maison XCode peut compiler une application pour le simulateur iOS.

Cycle de mise au point

En conclusion, à part dans le cas du Playbook, les tests applicatifs se feront :

- dans un premier temps dans ADL ;
- dans un second temps, une fois la version ADL validée, directement sur un périphérique réel.

Il faut être conscient que le cycle *Génération d'une version de déploiement, installation sur le périphérique* peut ralentir très sensiblement le processus de mise au point, en particulier lorsque la cible est un périphérique Apple.

Malheureusement, pour tester certaines fonctionnalités (en particulier les capacités natives comme la géolocalisation ou les interactions *multi-touch*), ce sera le seul chemin possible. Vous devrez donc prendre votre mal en patience. À prévoir néanmoins dans vos estimations de temps de développement !

Dialogue Run Configurations

La maîtrise des différents éléments du dialogue **Run Configurations**, *cf.* la figure 3, est essentielle pour la mise au point des applications mobiles.

Ce dialogue s'obtient par :

- Un clic droit sur le projet, puis sélection de Run As et du sous-menu Run Configurations...
- Sélection du menu Run > Run Configurations ...
- Le menu déroulant associé à l'icône d'exécution dans la barre d'outils (flèche verte), puis sélection de l'option Run Configurations...

Note : les configurations définies pour l'exécution (*run*) sont également appliquées pour le débogage (*debug*).

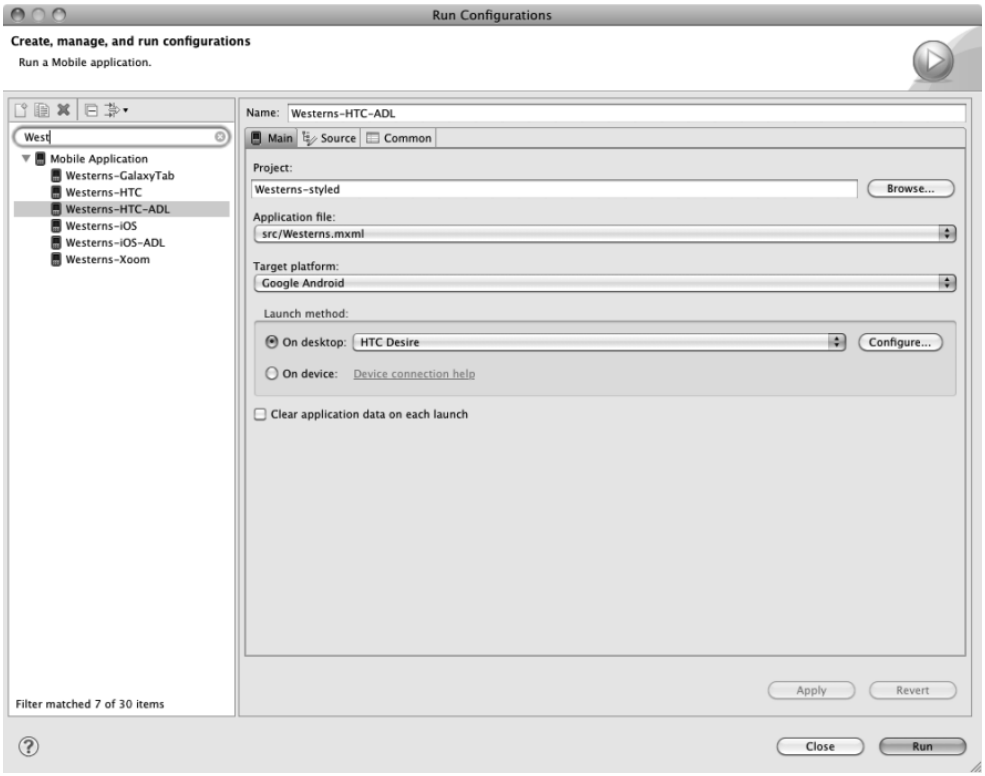

Figure 3 — Run Configurations

Gestion des configurations

Chaque configuration doit être dotée d'un nom unique permettant de la manipuler. Les boutons en haut à gauche du dialogue proposent des actions de base sur les fichiers de configuration. Ils sont repris dans la figure 4.

Dans l'ordre de gauche à droite :

- Ajout d'une configuration
- Duplication d'une configuration
- Suppression d'une configuration
- Déploiement, repli des configurations
- Filtrage des configurations

Paramétrages dépendant de la plateforme

Les options du haut de la fenêtre déterminent le projet et le fichier principal à lancer. Rien de particulier à signaler sinon de bien vérifier avant de cliquer sur le bouton Run que le projet souhaité est bien sélectionné. Lorsque plusieurs projets sont ouverts

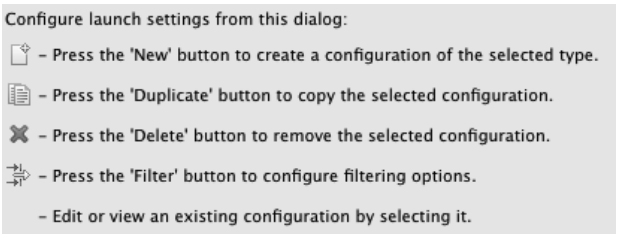

Figure 4 — Gestion des configurations

simultanément, il est assez fréquent d'obtenir le lancement du premier projet de la liste déroulante.

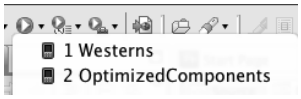

Figure 5 — Liste des projets ouverts

Les particularités du développement mobile constituent les options suivantes de la fenêtre.

Tout d'abord le menu Target platform propose de choisir la plateforme cible parmi celles définies pour le projet sélectionné. Vous ne pouvez tester une version de débogage que sur une plateforme à la fois. En revanche, nous verrons un peu plus loin que vous pouvez générer en une seule passe tous les fichiers de déploiement d'une version *release*.

Choix entre ADL et périphérique

Le contenu du groupe **Launch method** dépend de la plateforme sélectionnée.

Lorsque vous éditez une configuration d'exécution, Flash Builder vous propose le choix entre l'exécution dans ADL ou la génération d'une version pour un périphérique réel.

Pour toutes les plateformes, le choix par défaut est fixé à O*n desktop* ce qui correspond à une exécution au sein du simulateur ADL.

Lancement sur BlackBerry Tablet OS

La gamme BlackBerry Tablet OS sera peut-être amenée à s'agrandir, mais pour l'instant un seul périphérique est supporté : le BlackBerry Playbook. La figure 6 montre le menu déroulant des configurations par défaut proposées pour cet OS.

En cas de choix du bouton-radio On device, la figure 7 montre l'aspect obtenu.

En l'absence de connexion d'un périphérique, le choix se résume à l'exécution dans le simulateur proposé par RIM. Il s'agit déjà d'une intégration remarquable : nous

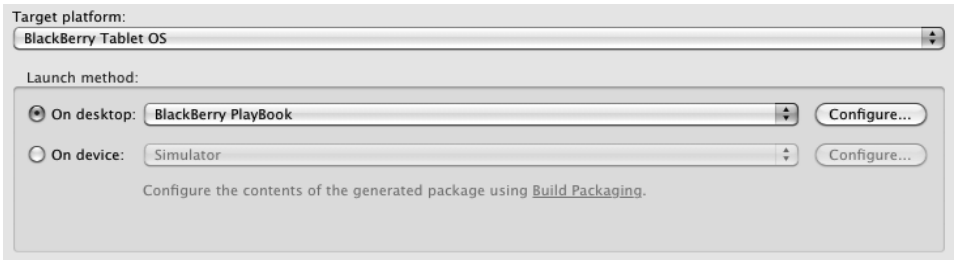

Figure 6 — Lancement pour Tablet OS

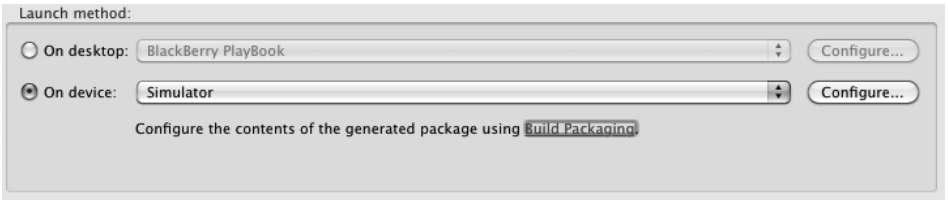

Figure 7 — Lancement pour Tablet OS sur périphérique

avons vu auparavant que le test direct vers le simulateur de la plateforme n'est pas possible pour les autres SDK.

Le simulateur doit être configuré avant une première exécution en cliquant sur le bouton **Configure**. Les différents réglages sont affichés dans la figure 8. En réalité, il n'est pas fait de différence entre la configuration du simulateur et d'un autre périphérique.

Le réglage essentiel consiste à définir le nom attribué à ce périphérique et l'adresse IP qui permettra la communication.

L'accès à la tablette est protégé par mot de passe. Il est possible d'enregistrer le mot de passe dans ce profil.

À définir également l'adresse IP utilisée pour le mode debug.

Enfin, est proposée la gestion du *debug token*, élément signé numériquement qui permet de déployer et déboguer une application non signée. Cette gestion est propre au Playbook et n'est pas nécessaire lorsque l'on utilise le simulateur. Pour plus de détail reportez-vous à la documentation de BlackBerry.

Lancement sur Google Android

Le choix de l'exécution d'une application pour la plateforme Android *On desktop*, c'est-à-dire dans ADL, remplit une liste déroulante déjà préconfigurée de nombreux périphériques. La figure 9 présente une partie de cette liste.

Essentiellement, les configurations sont groupées par plateforme et spécifient pour chaque élément une définition et une résolution d'écran.

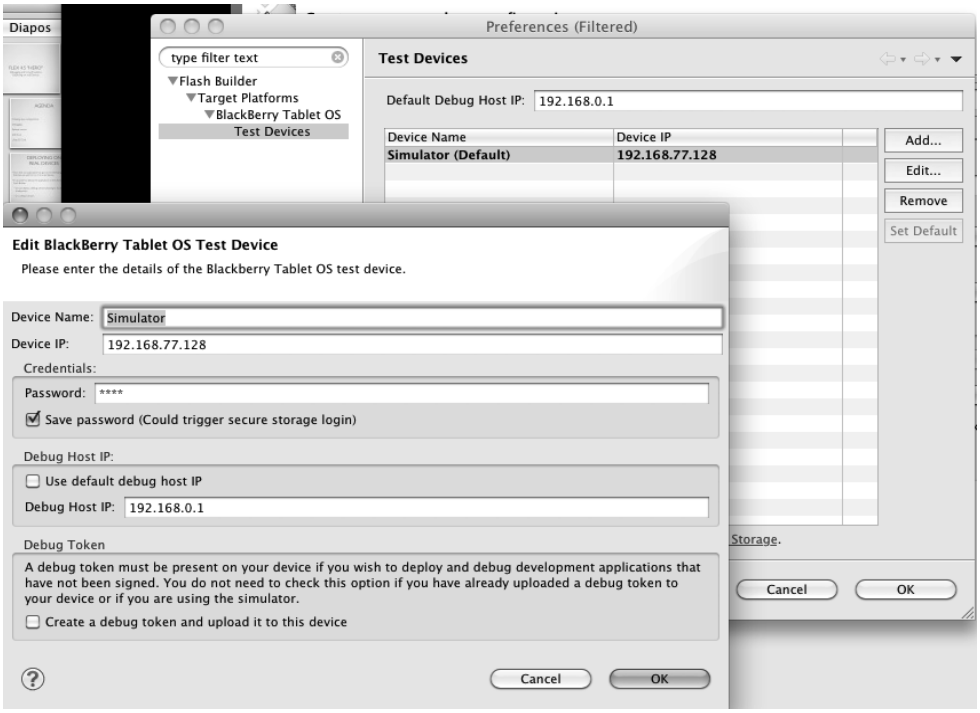

Figure 8 — Configuration du simulateur

Figure 9 — Lancement pour Android

Dans le chapitre 16, vous ajoutez un profil à cette liste afin de tester le comportement de l'application Westerns avec différentes résolutions.

Le choix *On device* ne propose aucune option supplémentaire. Si plusieurs périphériques sont connectés à la machine de développement, un dialogue demandera de choisir celui sur lequel doit s'effectuer le déploiement. Ensuite, le processus se déroule sans message final. L'application est simplement installée et lancée sur le périphérique.

Aucune installation n'est requise sur MacOS pour connecter un mobile Android. En revanche, sur Windows, même si Adobe intègre une série de pilotes USB pour les mobiles les plus courants, il se peut que votre appareil ne figure pas dans cette liste. Reportez-vous à la documentation pour savoir comment installer un pilote USB dans ce cas.

Lancement sur Apple iOS

Le choix de l'exécution d'une application pour la plateforme iOS *On desktop*, c'est-à-dire dans ADL, remplit une liste déroulante de trois types de périphériques possibles[1]. La figure 10 permet de visualiser cette liste constituée de iPad, iPhone3GS, iPhone 4.

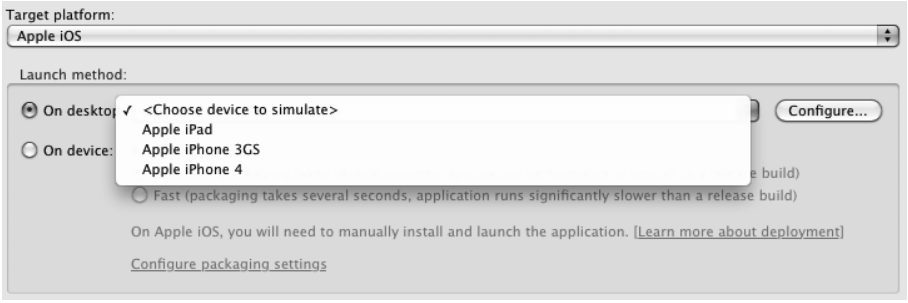

Figure 10 — Lancement pour iOS

Le choix *On device* propose deux options supplémentaires consultables dans la figure 11.

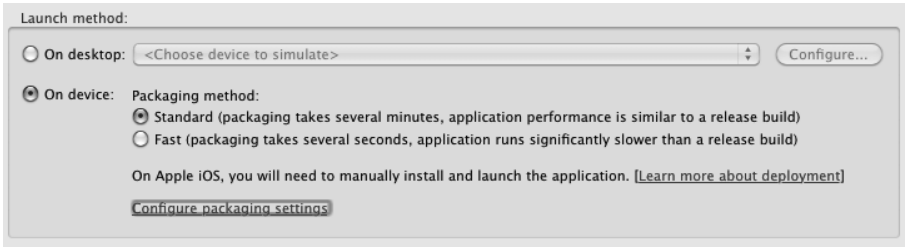

Figure 11 — Lancement pour iOS sur périphérique

Le mode *Fast packaging* génère rapidement une version non optimisée. À l'inverse, le mode *Standard packaging* demande plus de temps de construction, mais offre au final de meilleures performances.

Avec une cible iOS, pas d'installation automatique. En fin de génération, vous obtenez un fichier **.ipa** qu'il vous faut installer manuellement sur le périphérique.

1. Pas d'iPhone 5 à la date de rédaction !

La figure 12 montre le dialogue d'alerte affiché par Flash Builder afin de vous prévenir de la disponibilité du fichier .ipa à installer.

L'installation que vous effectuerez ensuite ne comprend plus aucun lien avec Flash Builder. Reportez-vous au paragraphe *Processus d'installation/Déploiement sur iOS*.

Figure 12 — Packaging Completed

Le multi-plateforme en récompense

Tout cela peut paraître un peu fastidieux à initialiser, mais permet ensuite de générer rapidement les différentes versions suivant la plateforme en cours de test.

La récompense de tous ces efforts s'obtient lorsque l'on contemple le dossier contenant un package par plateforme, comme dans la figure 13 :

- .apk : pour Android
- .bar : pour Tablet OS
- .ipa : pour iOS

Figure 13 — Packages multi-plateformes

Lancement sur le périphérique

Le processus de constitution du package prend un certain temps et en ce qui concerne iOS un temps certain ! (environ une dizaine de minutes, durée pouvant varier en fonction de l'application et de la machine de développement).

Un indicateur de progression affiche l'avancée du traitement comme le montre la figure 14.

Comme toute vue d'Eclipse, cette fenêtre peut être déplacée vers un endroit plus adapté. La figure 15 en donne un exemple affiché dans le bas du *workbench*.

Figure 14 — Progression du lancement

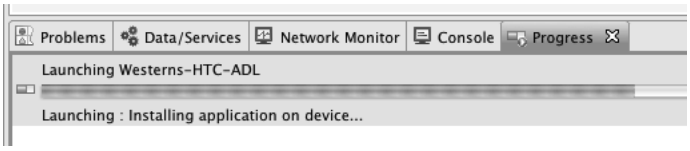

Figure 15 — Déplacement de la vue Progress

PROCESSUS D'INSTALLATION

Quelle que soit la plateforme ciblée, la fourniture d'un certificat d'identification sera nécessaire afin de :

- valider l'identité de l'auteur de l'application ;
- garantir l'intégrité de l'application après sa génération (pas de modification pour y introduire un *malware*).

Le processus de certification et les éléments requis varient suivant l'OS. L'accès au paramétrage s'effectue par les propriétés du projet, puis sélection du thème Flex Build Packaging.

Certificats

La gestion des certificats diffère suivant la cible. Faute de place suffisante, je me contenterai d'une présentation générale. Je vous invite à compléter les informations données ici par la lecture de trois tutoriels disponibles sur le site d'Adobe :
Packaging Adobe AIR applications
http://www.adobe.com/devnet/air/articles/packaging-air-apps.html

Android

Il suffit pour déployer sur Android de fournir le chemin d'accès vers un certificat au format (.p12) comme vous le présente la figure 16.

Pour les tests, Flash Builder propose la création d'un certificat auto-signé.

Cliquez sur le bouton Create... et renseignez le formulaire d'informations qui s'affiche comme dans la figure 17.

iOS

Pour les mobiles Apple, c'est un peu plus compliqué. L'inscription préalable auprès d'Apple comme développeur est un pré-requis. Elle s'effectue moyennant une redevance dépendant de votre mode de déploiement.

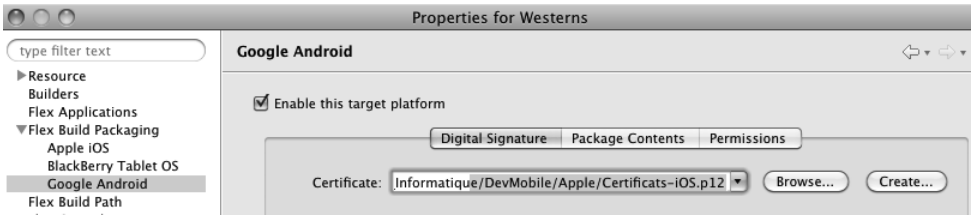

Figure 16 — Certificat Android

Figure 17 — Certificat auto-signé

Suite à cette formalité, vous pourrez télécharger un certificat de développeur (.cer) ainsi qu'un fichier de profil d'approvisionnement (.mobileprovision).

Le premier fichier sera à convertir au format p12. Une fois le certificat converti, vous fournirez le mot de passe associé comme dans la figure 18.

Figure 18 — IOS password

Blackerry Tablet OS

Mode opératoire encore différent avec le constructeur canadien : sur l'onglet Digital Signature figure une seule case à cocher Enable digital signing. Pour tester dans le simulateur, il n'est pas nécessaire de la cocher.

En revanche, pour déployer sur une vraie tablette, vous cocherez cette option, puis vous vous rendrez sur la page spéciale de réglages des informations de signature pour Tablet OS. La figure 19 illustre cette description.

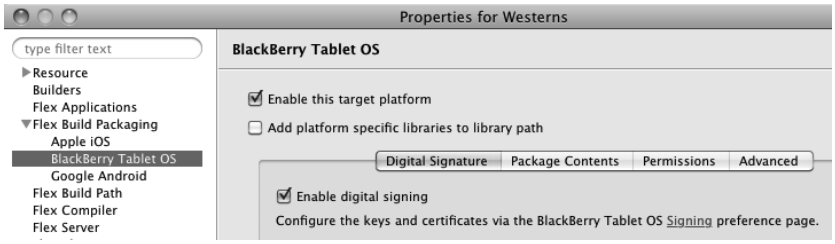

Figure 19 — Signature BlackBerry

> Il faut se souvenir que BlackBerry fournit un plugin complémentaire à Flash Builder ainsi qu'un SDK pour permettre le développement pour sa tablette. Certaines manipulations passent ainsi par les compléments de RIM et non les options natives à Flash Builder.

Cliquez maintenant sur le lien Signing pour accéder à la page spécifique pour Tablet OS. Cette page s'obtient également par les préférences de Flash Builder, puis choix du thème Flash Builder/Target Platforms/BlackBerry tablet OS/Signing. La page Signing est représentée dans la figure 20.

Comme pour Android, vous pouvez sélectionner un certificat au format p12 déjà existant ou créer votre propre certificat auto-signé.

Ensuite, enregistrez-vous auprès de RIM afin de pouvoir signer l'application. Ce processus est décrit sur le site du constructeur canadien. Vous trouverez la référence dans la page Web liée à ce chapitre.

Déploiement sur iOS

Autant il est facile de déployer une application sur Android, une simple URL dans un mail et le tour est joué, autant le processus est contraignant pour iOS.

iTunes

De base, le déploiement d'une application .ipa sur un périphérique iOS s'effectue grâce à iTunes. L'application se référence dans l'onglet Apps de la bibliothèque d'iTunes, puis une synchronisation la transfère sur l'appareil connecté.

Comme le montre la figure 21, lorsqu'un périphérique est connecté, il est possible de sélectionner les applications qui doivent être synchronisées.

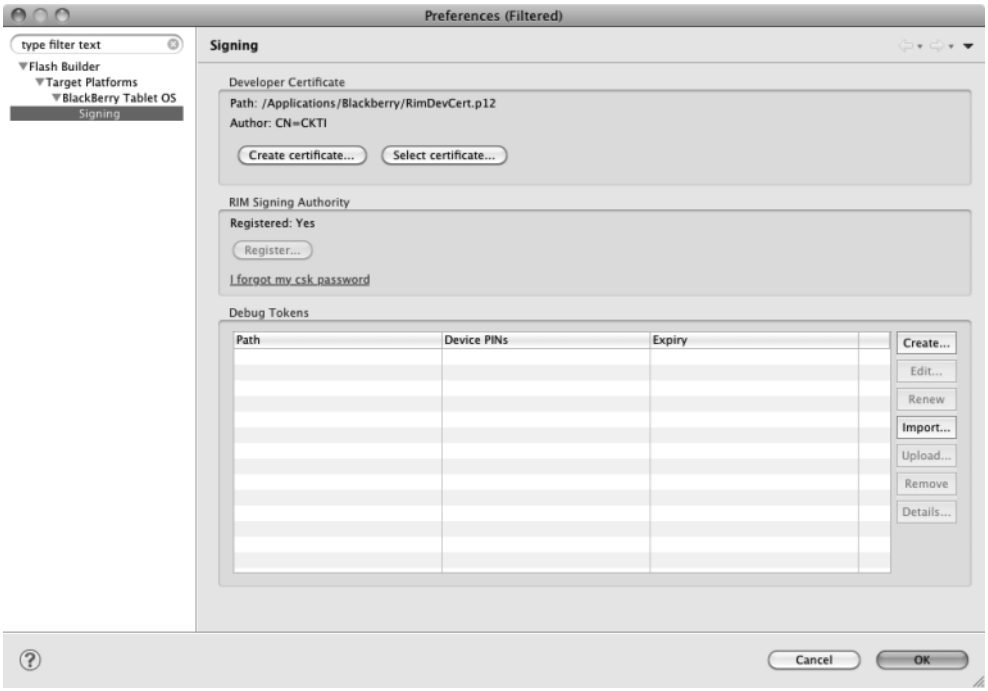

Figure 20 — Signing

Figure 21 — Synchroniser les apps

Il est préférable de désinstaller l'application du périphérique avant d'installer une nouvelle version par iTunes. Sinon, il semble que la version la plus récente ne soit pas toujours disponible après synchronisation.

Tout cela est un peu fastidieux, d'autant plus que la synchronisation par iTunes prend en charge tous les autres fichiers communs à l'ordinateur et au périphérique. Le temps requis est donc conséquent !

Solutions alternatives

Apple fournit un utilitaire pour faciliter la vie des administrateurs : vous pouvez télécharger sur http://support.apple.com/kb/dl851 iPhone Configuration Utility.

Disponible en version MacOS et Windows. D'après les retours des utilisateurs Windows, il apparaît que la version MacOS est de meilleure facture.

Une fois l'utilitaire lancé, la première étape consiste à ajouter l'application à déployer dans la liste des applications. Le bouton "ajouter" en haut à gauche ou le glisser-déposer fonctionnent très bien pour cela.

Vous obtenez une liste d'applications installables sur vos différents périphériques comme dans la figure 22.

Figure 22 — Applications Installables

En cliquant ensuite sur l'un des périphériques connectés, vous retrouvez dans l'onglet "Applications" la liste des applications installables sur cet appareil, comme dans la figure 23.

Figure 23 — Applications iPhone

Le bouton tout à droite de chaque application permet de basculer entre installation et désinstallation de l'application sur le mobile.

Une autre possibilité réservée aux possesseurs d'un Macintosh consiste à utiliser l'outil de développement d'Apple XCode. Le glisser-déposer d'un .ipa vers la fenêtre Organizer installe directement le package sur le périphérique sélectionné.

Figure 24 — XCode Organizer

Ces deux solutions évitent de passer par l'étape Synchronisation : un gain de temps appréciable !

D'autres solutions prometteuses commencent à voir le jour comme *hockeykit* ou *testflightapp*. Sous forme de service en ligne, elles exploitent la distribution *ad hoc* pour vous affranchir des contraintes d'iTunes. Vous trouverez les références sur la page web de ce chapitre.

VERSION FINALE

Avant de se lancer dans la génération d'une version finale, tous les éléments sont-ils prêts ?

Checklist :

- Les différents réglages sont définis dans les fichiers de description.
- Les jeux d'icônes pour chaque plateforme sont prêts et rangés au bon endroit.
- L'application a été testée et déboguée dans AIR et sur les périphériques.

Le moment est venu de déployer une version finale sur le périphérique.

Version *release*

Pourquoi faire la différence entre la version de débogage et la version de déploiement que l'on nomme version *release* ?

La réponse est simple : parce que la version de débogage contient du code supplémentaire ... pour le débogage ! Ainsi du code de communication avec le débogueur ou du code d'envoi de messages vers la console. Tout cela est retiré de la version de déploiement finale.

La version *release* est donc plus compacte, plus rapide et il est vraiment recommandé de ne jamais mettre en exploitation une version de débogage.

Génération

Le processus se déclenche par la sélection du menu **Project > Export Release Build ...** Un assistant ouvre un dialogue dont le contenu dépend des plateformes ciblées par le projet. La figure 25 présente le contenu de la première page de ce dialogue.

Vérifiez dans les menus déroulants du haut de la fenêtre que sont bien sélectionnés le projet et l'application souhaités.

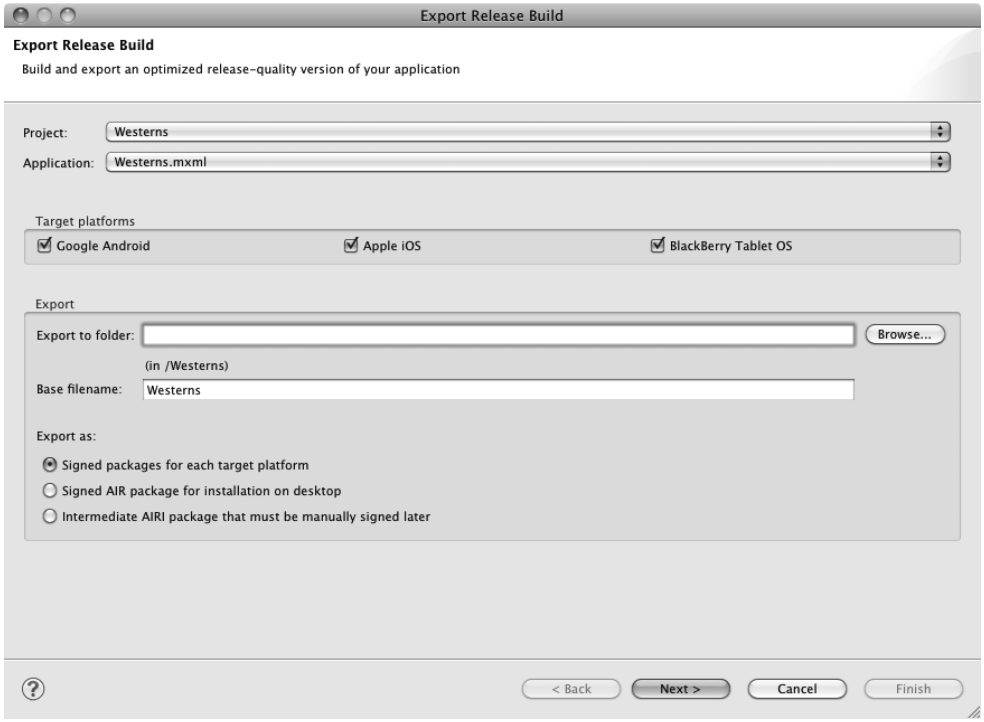

Figure 25 — Export Release Build

Target platforms

Sont ensuite présentées sous forme de cases à cocher, les différentes plateformes référencées dans votre projet. Pour la génération de la version de déploiement, vous avez la possibilité de n'en conserver qu'une sélection plus restreinte. Par exemple, si votre application n'est pas encore finalisée pour iOS, vous choisirez de décocher cette option.

Export

Vous pouvez ici définir un dossier pour l'export ainsi que le nom qui servira de racine pour le fichier généré. En fonction de la plateforme, seront ajoutés à cette racine les différents suffixes : ipa, apk, bar.

Pour une application mobile, vous conserverez généralement la sélection du bouton-radio Signed packages for each platform.

Écrans suivants

Cliquez ensuite sur le bouton "Next".

Sur la gauche de l'écran, vous retrouvez la sous-sélection des plateformes que vous ciblez dans cette *release*. Sur la droite de l'écran figurent plusieurs onglets dont le nombre et le contenu diffèrent suivant la plateforme.

Les deux premiers sont communs aux trois cibles, même si le contenu des options présentées n'est pas identique :

- **Digital Signature**, afin de compléter les différentes informations authentifiant l'auteur de l'application et permettant de signer numériquement l'application pour chaque cible.
- **Package Contents**, permet de définir un contenu différent suivant la plateforme cible.

L'écran **Digital Signature** correspond aux déclarations déjà explicitées dans le paragraphe Signatures.

La figure 26 affiche ces informations pour Android.

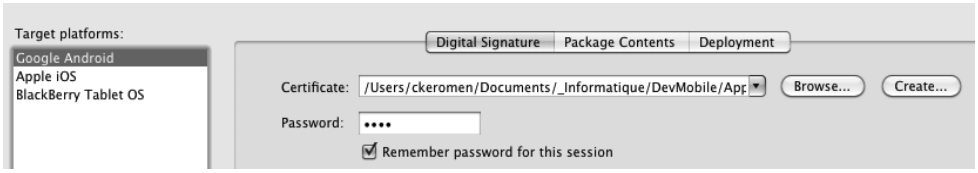

Figure 26 — Digital Signature pour Android

Digital Signature pour BlackBerry propose la version correspondante pour Blackberry. Cet écran se contente de proposer une case à cocher d'activation de la signature numérique, le paramétrage s'effectuant dans la page spécifique pour Tablet OS.

Figure 27 — Digital Signature pour BlackBerry

PackageContents

La figure 28 reprend le contenu de cet écran.

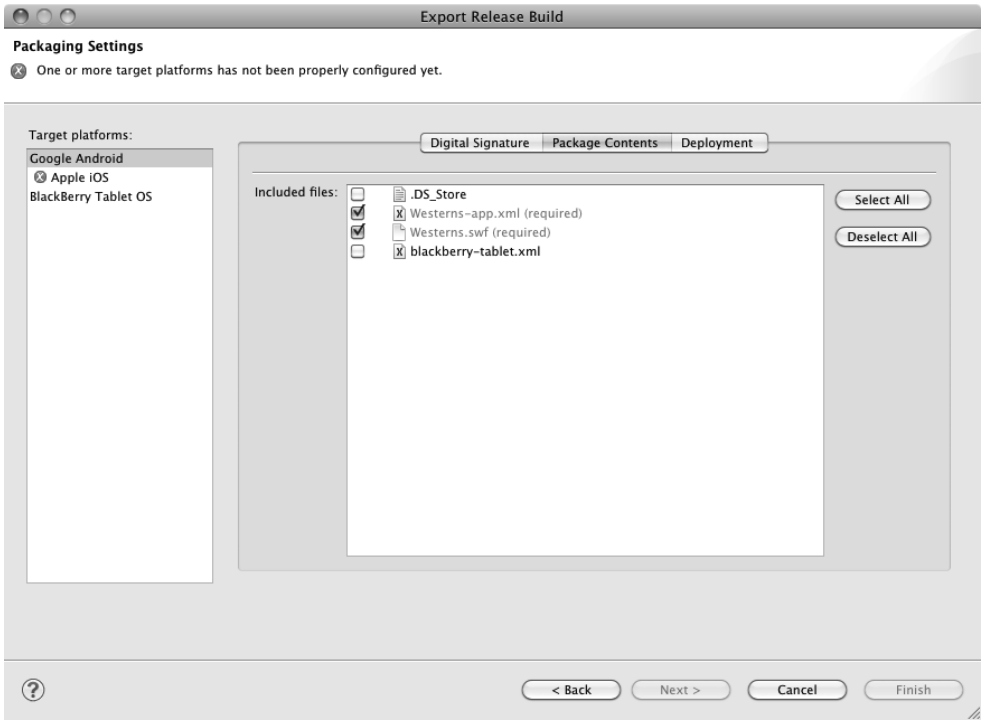

Figure 28 — PackageContents

Sur cet exemple, sont exclus du contenu pour Android, les fichiers .DS_Store, propres à MacOS, ainsi que le fichier descripteur **blackberry-tablet.xml** inutile pour cette cible.

> **Attention** : pensez à exclure **blackberry-tablet.xml** de la configuration pour iOS avant de générer une version pour l'AppStore. Sa présence peut poser des problèmes.

Deployment

Pour la cible Android, on trouve un troisième onglet Deployment repris dans la figure 29.

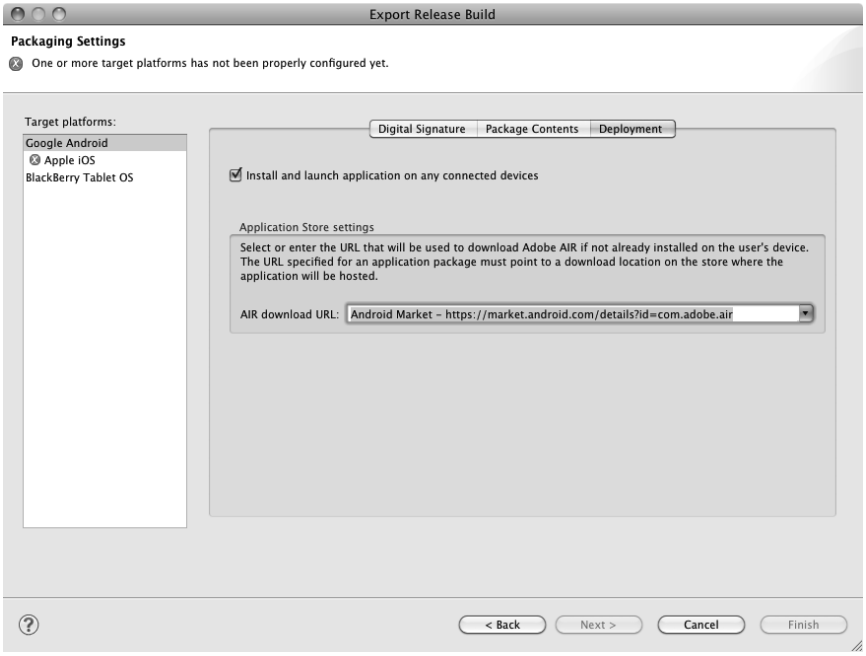

Figure 29 — Deployment

Seulement deux options dans cette fenêtre :

- Une case à cocher demandant à installer et lancer l'application générée sur tout périphérique Android connecté. Option classe confort que l'on aimerait tant retrouver sur la partie iOS !
- Une liste déroulante précisant de quel emplacement le *runtime* AIR doit être téléchargé s'il n'est pas déjà présent sur le périphérique lors de la première utilisation. Cette option a été rendue nécessaire par l'apparition du magasin d'Amazon : l'Amazon Appstore (http://www.amazon.com). Si l'application a été téléchargée depuis un magasin, il faut qu'AIR soit téléchargé depuis la même origine ! Attention, cela peut être une cause de rejet de l'application par Amazon.

Signature pour Apple iOS

Deux onglets seulement pour la plateforme d'Apple : Digital Signature (figure 30) et Package Contents qui correspond à l'écran de même nom pour Android.

En plus des différentes informations nécessaires à la signature numérique (certificat, mot de passe et fichier de provision Apple), figure dans ce dialogue le choix du mode de génération de la version iOS :

- Pour une distribution sur l'Apple App Store, dans ce cas un fichier de profil de provision de distribution est requis.
- Pour une distribution *ad hoc*.

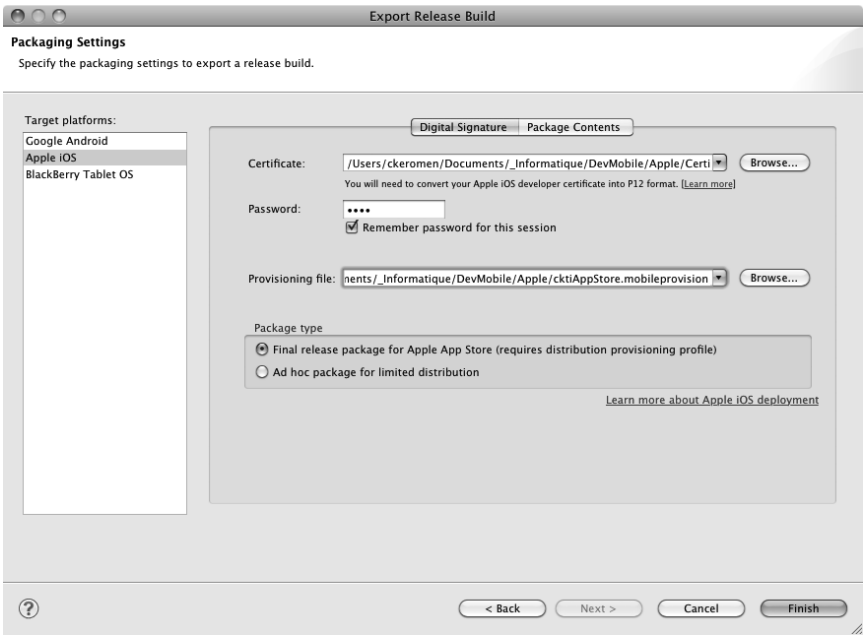

Figure 30 — Digital Signature

Écrans spécifiques BlackBerry

Pour la cible BlackBerry, on retrouve un onglet permettant de spécifier les différentes permissions qui seront conservées dans le fichier **blackberry-app.xml**.

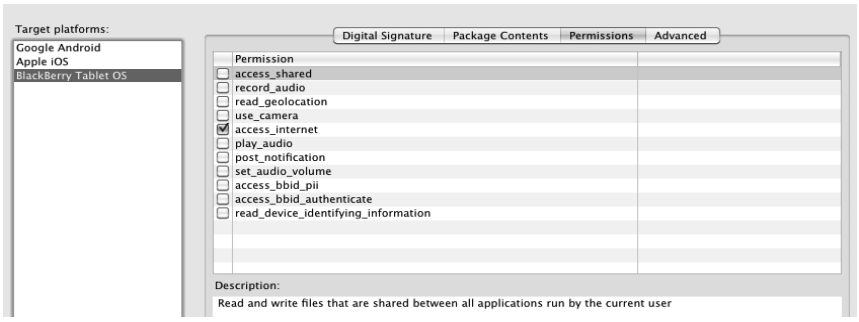

Figure 31 — Blackberry Permissions

Enfin un quatrième onglet, figure 32, propose des options très spécifiques à cette cible.

Dans la version utilisée pour ce livre, seule la possibilité de définir une option de packaging est proposée. Dans l'exemple suivant, elle est appliquée pour indiquer la version d'AIR à utiliser.

Figure 32 — BlackBerry Advanced

Périphériques iOS supportés

Encore une dernière précaution avant de lancer le déploiement sur iOS : vérifiez la compatibilité des périphériques. Au minimum, la version 4 du système doit être installée. Ce pré-requis élimine les modèles d'iPhone avant le 3GS et limite à la liste suivante les mobiles potentiels : iPhone 3GS, iPhone 4, iPad 1 & 2, iPod Touch 3e et 4e générations[1].

En résumé

Au terme de ce long chapitre, vous avez acquis une vue globale de la génération de versions de déploiement.

Il ne faut cependant pas masquer que nous sommes ici à la frontière entre l'outil multi-plateforme qu'est Flex et l'approche spécifique de chaque fournisseur. Glaner quelques compléments d'information sur les sites correspondants et consacrer du temps de mise en œuvre se révélera incontournable pour compléter cette lecture...

1. http://kb2.adobe.com/cps/891/cpsid_89107.html#main_System_requirements