Programmation Android

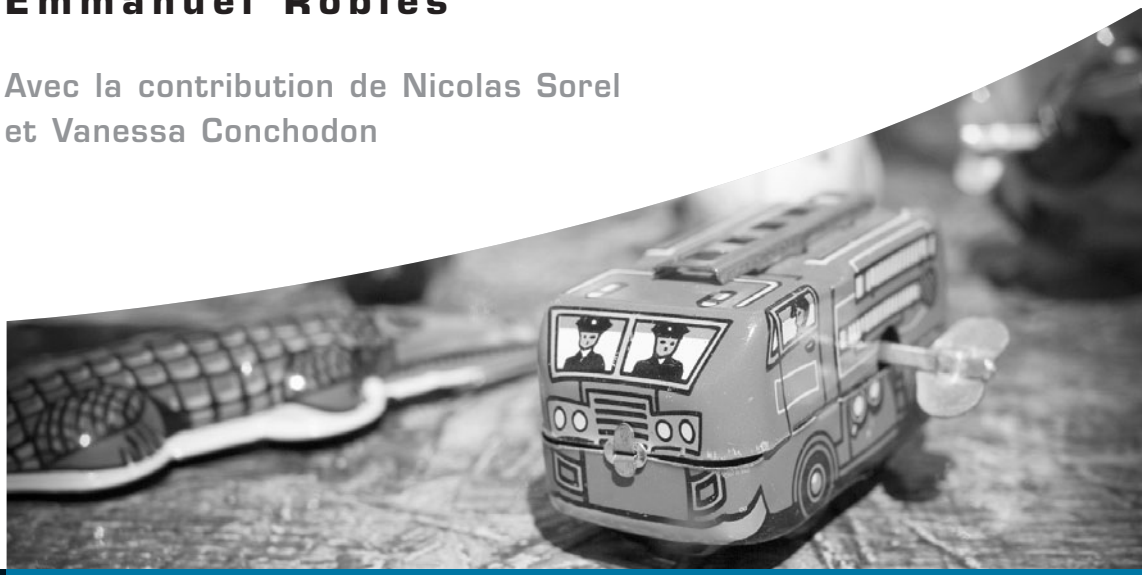
De la conception au déploiement avec le SDK Google Android 2

Damien Guignard

Julien Chable

Emmanuel Robles

Avec la contribution de Nicolas Sorel
et Vanessa Conchodon



EYROLLES

Programmation Android

Par son ouverture et ses possibilités de déploiement, la plate-forme Google Android basée sur Linux offre un socle et un environnement de développement puissants pour créer des applications mobiles robustes et ergonomiques. Elle met à la portée des professionnels et des particuliers la réalisation d'applications à la fois riches en fonctionnalités et adaptées aux contraintes de l'utilisation mobile.

Bien développer sous Android : ergonomie et robustesse

Écrit par des développeurs Android expérimentés et présents sur le marché depuis la sortie d'Android, cet ouvrage détaille les bonnes pratiques de conception, de développement et de publication avec le SDK Android (versions 1.5, 1.6 à 2.x), depuis la conception d'une interface utilisateur, l'accès aux données et au matériel (senseurs, etc.), le multimédia et l'utilisation de Google Maps, jusqu'à la publication sur l'Android Market, en passant par la création de services, la gestion des threads et les tests dans l'émulateur ou sur les appareils disponibles. L'ouvrage décrit les différentes API Android : géolocalisation et GPS, graphisme 2D et 3D OpenGL, audio et vidéo, WiFi, Bluetooth…

Au sommaire

À qui s'adresse cet ouvrage ?

– Aux développeurs (Java/.NET, PHP, Python, Ruby, etc.) souhaitant créer des applications mobiles Android.

– Aux professionnels de la conception d'applications mobiles et aux agences web souhaitant être présents sur le marché Android.

Damien Guignard est développeur et formateur sur Java ME, Java EE et Android. Il intervient auprès d'entreprises pour fournir son expertise sur l'ensemble des technologies Java.

Julien Chable est développeur Java, .NET et Android. Consultant auprès de grands groupes en développement sur les plates-formes collaboratives et mobiles (notamment SharePoint) il participe au réseau Codes Sources en tant qu'administrateur.

Emmanuel Robles développe très tôt des applications pour ATARI, PC, puis pour tous types de plates-formes. Auteur de plusieurs applications commercialisées sur l'Android Market, il crée en 2009 avec Nicolas Sorel *Androlib.com*.

Nicolas Sorel, crée en 1999 le réseau *Codes-Sources* (plus de 1,5 million de membres) et co-fonde *Androlib.com* avec Emmanuel Robles.

Conception : Nord Compo © Photo de couverture, Fotolia

www.editions-eyrolles.com
Groupe Eyrolles | Diffusion Geodif

EYROLLES

Avant-propos

La téléphonie mobile a connu une explosion dans les années 2000 mais aucune révolution n'a semblé arriver depuis : les appareils tendaient à tous se ressembler, les innovations n'avaient plus vraiment de saveur ; les applications étaient difficiles d'accès de par leur mode de distribution et souvent peu performantes à cause des faibles capacités des appareils.

Depuis quelques mois, les smartphones sont dotés d'une puissance plus importante et d'espaces de stockage conséquents. Les téléphones tendent à devenir des objets artistiques, presque de reconnaissance sociale, et possèdent des fonctionnalités qu'aucun téléphone ne pouvait espérer auparavant : connexion haut débit, localisation GPS, boussole, accéléromètre, écran tactile souvent multipoint, marché d'applications en ligne… Autant de qualités permettant de créer des applications innovantes et de les distribuer en toute simplicité.

La plate-forme Android apporte tout cela au consommateur, mais surtout, elle affranchit le développeur de nombreuses contraintes par son ouverture ; elle permet à n'importe quel développeur de créer ses applications avec un ticket d'entrée quasi nul. Le framework et le système d'exploitation et outils associés ont un code source ouvert, leur accès est gratuit et illimité. Plus besoin de négocier avec le constructeur du téléphone pour qu'il vous laisse développer sur sa plate-forme.

Tous les développeurs sont ainsi sur un même pied d'égalité, qu'ils soient une grande entreprise ou quelques jeunes dans un garage ; tous peuvent ajouter de la mobilité à des applications existantes.

À qui est destiné cet ouvrage ?

Cet ouvrage se veut accessible à toute personne qui souhaite créer des applications mobiles sur la plate-forme Android. Que vous soyez un développeur confirmé ou une personne débutant tout juste dans la programmation informatique, nous espérons que ce livre vous donnera l'envie et les informations nécessaires pour vous permettre de créer les applications de demain.

Cet ouvrage ne traite pas du langage ou de la plate-forme Java. Une première expérience en Java est conseillée, la plate-forme Android étant basée sur ce langage.

Achat d'un téléphone de test

Avant d'investir dans l'achat d'un téléphone Android de développement ou de vous inscrire sur le marché Android, lisez attentivement les premiers chapitres et réalisez les exemples nécessaires pour bien démarrer. Bien évidemment si vous possédez déjà un téléphone s'exécutant sous Android, cela représente déjà un avantage pour tester vos applications. Vous trouverez en annexe une partie sur la manière de configurer votre téléphone pour développer et tester directement vos applications sur ce dernier.

Versions d'Android liées à ce livre

L'évolution de la plate-forme Android est rapide : lors du projet initial de cet ouvrage, Android était en version 1.5, avant de passer rapidement en version 1.6. À l'heure de l'écriture de ce livre, Android 2.0 est le standard qui tend déjà à se répandre auprès des développeurs.

Tous les exemples de ce livre ont été créés avec Android 1.5 et la plupart vérifiés avec Android 2.0. Cependant, le rythme élevé des évolutions du SDK et les modifications réalisées, qui sont parfois non compatibles avec les versions émises précédemment, pourront nécessiter des adaptations du code. L'utilisation des exemples de ce livre ne nécessite pas l'achat d'un appareil Android : tous les développements peuvent être réalisés sur l'émulateur, exception faite des exemples du chapitre 15 sur le matériel.

Mises à jour et errata

Vous trouverez des ressources complémentaires et éventuels errata sur la fiche du livre sur le site des éditions Eyrolles et sur le site dédié au livre :

- www.android-le-livre.fr
- http://www.editions-eyrolles.com

Structure de l'ouvrage

La première partie de cet ouvrage présente la plate-forme Android et vous guide à travers l'installation de l'ensemble de l'environnement logiciel nécessaire à la mise en pratique des concepts et des exemples proposés dans ce livre.

La deuxième aborde ensuite les thèmes fondamentaux indispensables à la conception d'applications Android : composition des applications, conception et réalisation d'une première application, création d'interfaces utilisateur et enfin, présentation du mécanisme de communication entre applications (les Intents).

La troisème partie regroupe les problèmatiques qui permettront de maîtriser les techniques qui rendront votre application interactive et communicante : interfaces utilisateur avancées, persistance et exposition des données, multimédia, graphismes 3D, réseau, géolocalisation et gestion du matériel.

Enfin, la quatrième partie de ce livre vous accompagne jusqu'à la publication, sur l'Android Market, de l'application que vous aurez conçue.

À propos des auteurs

Damien Guignard est développeur Java et également formateur Java ME et Android. Il est le fondateur d'une jeune société, Neimad, au travers de laquelle il intervient auprès des sociétés qui souhaitent partager ses 10 ans de fidélité au langage Java sous toutes ses formes.

Julien Chable est développeur et consultant depuis de nombreuses années auprès de PME et de grands groupes. Spécialisé dans le développement et le conseil sur les plates-formes collaboratives et mobiles, il aide les entreprises à se lancer en leur communiquant son expertise.

Emmanuel Robles se passionne dès l'enfance pour les technologies de l'informatique. Très vite, il commence à développer des applications pour ATARI, PC et maintenant pour tous types de plates-formes. Principalement dévoué à la création sur le système d'exploitation Android sur lequel Emmanuel a déjà réalisé plusieurs applications commercialisées sur l'Android Market, il crée avec Nicolas Sorel *Androlib.com* en juillet 2009

Nicolas Sorel, passionné par la programmation informatique, crée Codes-Sources en 1999 afin de permettre à tous les développeurs francophones de partager leurs connaissances en informatique. Cette communauté qui regroupe aujourd'hui plus de 1,5 million de membres offre, 10 ans après sa création, plus de 40 000 sources de code. Dès 2008, Nicolas s'intéresse de près au développement Mobile et crée avec Emmanuel Robles *Androlib.com* en juillet 2009.

Remerciements

Damien Guignard – Merci à celles et ceux qui m'ont donné mes premières ou mes secondes chances (Chrystel, Fabienne, Serge, Laurent, Sébastien, Hervé, Xavier et Christophe). Un grand merci également à tous ceux qui n'ont pas compté leurs heures sur ce livre. Et puis, c'est quand on est totalement absorbé par l'écriture ou la relecture finale qu'on s'aperçoit de l'importance de certains remerciements. Merci donc à tous ceux qui n'ont pas eu beaucoup de nouvelles et qui ne m'en tiennent pas rigueur. Et enfin, merci mon Ange, maintenant que ce livre est terminé, il est temps d'écrire les plus belles pages de notre livre de souvenirs.

Julien Chable – J'adresse mes remerciements à Damien, Nicolas et Emmanuel pour m'avoir accepté dans l'aventure. Je tiens également à remercier ma compagne sans qui ma participation n'aurait pu voir le jour. Pour terminer, je remercie bien sûr l'équipe Eyrolles : Muriel Shan Sei Fan et Vanessa Conchodon pour leur travail et leur confiance qui ont donné vie à ce livre.

Emmanuel Robles - Je remercie ma femme, ma famille et mes associés pour ce qu'ils sont : formidables ! Myriam Longuet, experte en psychologie du « Geek » et d'une patience inébranlable ainsi que toute l'équipe de Video2Brain. Reto Meier, Android Developer Advocate chez Google pour sa sympathie et sans qui Android ne serait pas ce qu'il est. Alain Herry, toujours au taquet, pour m'aider comme si sa vie en dépendait. Enfin, un remerciement spécial au « groupe ».

Nicolas Sorel - Je remercie Aude Sorel, Alice Sorel et Maxime Sorel pour la patience qu'ils ont avec moi. Grégory Renard et la société Wygwam pour leur compétence et leur aide inestimable. Étienne Jambou, directeur Marketing chez Magma Mobile pour son flegme et sa clairvoyance de chaque instant. Eclipse, l'émulateur Android et surtout l'adb toujours aussi taquins. Enfin, une dédicace spéciale à mon chien Végas.

Les sources de ce livre

Tous les codes sources des exemples de ce livre sont disponibles sous licence Apache 2.0 si vous souhaitez les réutiliser ailleurs que dans le cadre de votre formation avec cet ouvrage.

Vous trouverez les sources à télécharger sur le site des éditions Eyrolles, sur la fiche du livre, et sur le site dédié au livre :

- http://www.android-le-livre.fr
- http://www.editions-eyrolles.com

Table des matières

CHAPITRE 4
Communication entre applications : la classe Intent 101

CHAPITRE 5
Création d'interfaces utilisateur avancées 123

CHAPITRE 13
Géolocalisation, Google Maps et GPS 379

ANNEXE A

1

La plate-forme Android

Débuter sur une nouvelle plate-forme n'est jamais évident. Ce chapitre vous permettra de bien commencer.

Que vous débutiez en développement ou que vous soyez déjà un développeur Java aguerri, Android est une excellente opportunité pour appréhender le développement d'applications mobiles ambitieuses.

Rappelons les points clés d'Android en tant que plate-forme :

- elle est innovante car toutes les dernières technologies de téléphonie y sont intégrées : écran tactile, accéléromètre, GPS, appareil photo numérique etc. ;
- elle est accessible car en tant que développeur vous n'avez pas à acheter de matériel spécifique (si vous voulez aller plus loin que l'utilisation d'un émulateur, un téléphone Android pour effectuer vos tests vous sera toutefois nécessaire), ni à connaître un langage peu utilisé ou spécifique : le développement sur la plate-forme Android est en effet réalisé en langage Java, un des langages de programmation les plus répandus ;
- elle est ouverte parce la plate-forme Android est fournie sous licence open source, permettant à tous les développeurs – et constructeurs – de consulter les sources et d'effectuer les modifications qu'ils souhaitent.

Ce chapitre est une introduction à la plate-forme, aux outils et à la configuration de votre environnement de travail. Si cet ouvrage est votre premier contact avec la programmation Android, commencez par lire ce chapitre attentivement.

La plate-forme Android

De par son ergonomie et les capacités du matériel, l'arrivée de l'iPhone d'Apple a bouleversé le paysage des systèmes d'exploitation mobiles, tant par les usages proposés que par les possibilités offertes aux utilisateurs avec le marché d'applications Apple Store.

La firme de Cupertino a su en quelques mois instaurer sa suprématie dans le monde de la téléphonie et surtout initier de nouveaux usages, qui ont conduit au raz-de-marée que l'on connaît. Aujourd'hui, les utilisateurs ne cessent de comparer leur téléphone à l'iPhone : ergonomie, écran tactile, simplicité, connexion Internet quasi permanente, multitude d'applications, etc. De créateur de nouveaux usages, l'iPhone est devenu un véritable standard pour les utilisateurs.

Devant les lacunes des offres des constructeurs historiques, une coalition s'est créée pour mettre au point et promouvoir un système d'exploitation mobile concurrent. Ce rassemblement, qui a vu le jour fin 2007, se nomme l'*Open Handset Alliance* et se compose aujourd'hui de 47 acteurs qui ont pour objectif de créer et de promouvoir un système complet, ouvert et gratuit dans le monde du mobile : Android.

> **ALLEZ PLUS LOIN** **Où se renseigner sur l'Open Handset Alliance**
>
> Vous pouvez vous rendre sur le site officiel de l'alliance. Notez que l'Open Handset Alliance et Android sont des marques déposées.
>
> ▸ http://www.openhandsetalliance.com

Tordons cependant le cou aux simplifications et notamment à celle trop souvent faite : « Android = Google phone ». Google est certes l'acteur majeur de ce rassemblement autour d'Android, mais les téléphones Android disponibles sur le marché n'ont rien à voir avec cette firme, mis à part le « Nexus One », lancé tout début 2010 et qui est fabriqué spécialement pour Google. L'offre des constructeurs s'élargit d'ailleurs chaque jour davantage.

Les versions de la plate-forme

Au moment de l'écriture du livre, Android est disponible en version 2.1. Les versions se succèdent rapidement et les changements qui les accompagnent sont souvent conséquents en termes de nouvelles fonctionnalités et d'améliorations. Cependant, ces évolutions n'ont pas toujours été sans impact sur le bon fonctionnement et la compatibilité des applications entre versions.

Le tableau suivant présente l'historique des versions d'Android jusqu'à la rédaction de cet ouvrage. Vous le noterez certainement, les versions majeures récentes répondent toutes à de doux noms de desserts.

Tableau 1–1 Historique des versions d'Android

Versions	Quelques évolutions
Android 1.0 Septembre 2008	Début de l'aventure Android
Android 1.1 Février 2009	- Beaucoup de corrections de bogues (alarme, Gmail...) - Amélioration des fonctionnalités MMS - Introduction des applications payantes sur l'Android Market
Android 1.5 dit « Cupcake » Avril 2009	- Nouveau clavier avec autocomplétion - Support Bluetooth - Enregistrement de vidéos - Widgets et Live Folders - Animations changées
Android 1.6 dit « Donut » Septembre 2009	- Recherche dans les éléments améliorée - Recherche vocale améliorée - Android market remanié - Meilleure intégration entre la galerie multimédia et la prise de clichés ou de vidéos - Indicateur d'utilisation de la batterie - Apparition de fonctionnalités pour les réseaux privés virtuels (VPN)
Android 2.0 dit « Eclair » Octobre 2009	- Interface graphique remaniée - Gestion des contacts changée - Support d'HTML 5 - Support Bluetooth 2.1 - Clavier virtuel amélioré - Refonte de l'installation et des mises à jour du kit de développement
Android 2.0.1 Décembre 2009	Mise à jour mineure (corrige un ensemble de bogues comme pour l'autofocus, permet une réduction de la consommation de la batterie et quelques améliorations graphiques comme l'écran de verrouillage)
Android 2.1 Janvier 2010	Mise à jour encore mineure, même si elle apporte beaucoup de nouvelles choses : - le bureau se compose de 5 écrans virtuels - fonds d'écran animés (en fonction du toucher ou du son…) - nouveaux effets 3D (notamment sur la gestion des photos) - amélioration du client Gmail - interface graphique pour l'utilisation du téléphone en voiture (semblable à celle apparaissant au branchement sur la station pour les Motorola Droid)

Une architecture autour du noyau Linux

Android est conçue pour des appareils mobiles au sens large. Nullement restreinte aux téléphones, elle ouvre d'autres possibilités d'utilisation des tablettes, des ordinateurs portables, des bornes interactives, des baladeurs…

La plate-forme Android est composée de différentes couches :

- un noyau Linux qui lui confère notamment des caractéristiques multitâches ;
- des bibliothèques graphiques, multimédias ;
- une machine virtuelle Java adaptée : la *Dalvik Virtual Machine* ;
- un framework applicatif proposant des fonctionnalités de gestion de fenêtres, de téléphonie, de gestion de contenu... ;
- des applications dont un navigateur web, une gestion des contacts, un calendrier…

Les composants majeurs de la plate-forme Android sont résumés sur le schéma suivant (traduit de la documentation Google).

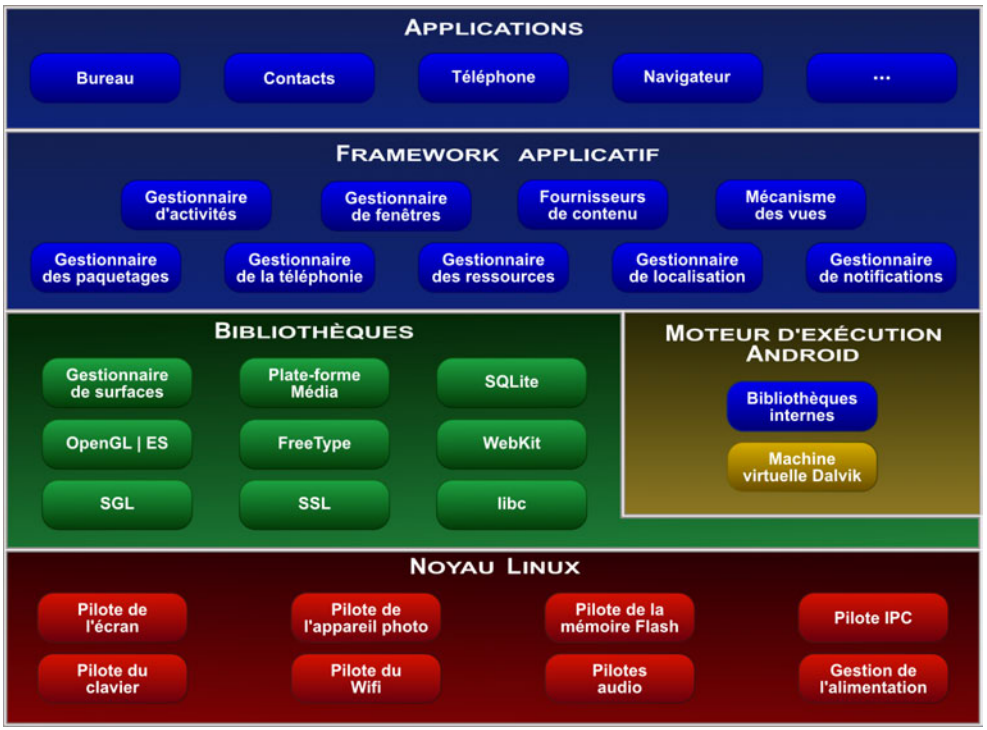


Figure 1–1 Les composants d'Android

La licence

Si vous développez sur la plate-forme Android, vous devrez accepter la licence associée qui vous sera présentée lorsque vous téléchargerez le kit de développement Android.

Celle-ci présente les modalités d'utilisation du kit de développement et de la plate-forme en général, avec un accent mis sur l'utilisation des API Google (la notion d'API et les API Google seront abordées ultérieurement).

La licence principale du projet Android est la licence Apache 2.0. Cependant, d'autres parties du projet peuvent être sujettes à d'autres types de licences : c'est notamment le cas des parties logicielles proches du noyau Linux qui, elles, sont concernées par la licence GPL.

Aller plus loin

Pour plus d'informations sur la licence GPL, n'hésitez pas à lire l'excellente biographie de celui qui formalisa les licences libres, Richard Stallman :

📖 *Richard Stallman et la révolution du logiciel libre – Une biographie autorisée*, Stallman *et al.*, Eyrolles 2010

La licence Apache autorise la modification et la distribution du code sous forme libre ou non, et permet, comme toute licence vraiment libre, d'en faire un usage commercial. Elle oblige le maintien du copyright lors de toute modification (et également du texte de la licence elle-même).

Le marché

Si début 2009 seul le téléphone de HTC (le G1) était disponible, le nombre de modèles d'appareils Android a augmenté depuis à un rythme soutenu. Du Motorola Droid aux HTC Hero, Nexus One, en passant par le Sony Ericsson Xperia X10 et autres, de plus en plus de modèles sont équipés d'Android.

À NOTER Android dans le monde

Au Mobile World Congress 2010 (le rendez-vous international de l'industrie du mobile), Eric Schmidt (Google) a indiqué qu'Android était présent sur 26 smartphones, traduit dans 19 langues et vendu dans 48 pays par 59 opérateurs.

▸ http://www.zdnet.fr/actualites/telecoms/0,39040748,39713101,00.htm

Selon certaines études optimistes, le nombre de téléphones devrait doubler sur l'année 2010 avec une estimation de 50 % du nombre de modèles de téléphones mobiles disponibles tournant sous Android.

Android ne profite pas seulement du bruit (voire « buzz ») qu'il a créé à sa parution mais suscite aussi un réel engouement de la part des constructeurs qui voient dans ce système une occasion de rattraper leur retard par rapport aux besoins créés par l'iPhone. Il ne se passe pas une journée sans avoir une annonce ou un nouveau test de téléphone Android.

Une nouvelle opportunité pour relancer le marché ? Pas seulement, car si l'iPhone a su changer les exigences des utilisateurs, Android devrait s'annoncer comme l'élément déclencheur d'un nouveau marché des systèmes d'exploitation mobiles : un marché où la standardisation et l'ouverture du système devraient permettre plus d'innovation et de flexibilité pour les utilisateurs.

Android et ses concurrents

Android affronte une forte concurrence dans un marché où les solutions logicielles pour appareils mobiles sont nombreuses :

- iPhone OS : le concurrent numéro un pour Android. Il s'agit bien sûr du système présent sur les différents générations d'iPhone, produits par Apple mais également sur ses tablettes iPad ;
- Windows Mobile : tout autant propriétaire, le système pour mobiles proposé par Microsoft est distribué sur de nombreux téléphones ;
- Symbian : récemment passé en open source, ce système d'exploitation est la propriété de Nokia et est présent – entre autres – sur un grand nombre de téléphones de la firme finlandaise ;
- BlackBerry OS : il est présent sur tous les téléphones de la marque RIM (*Research In Motion*) ;
- Palm webOS : le successeur de Palm OS, qui équipe tous les terminaux de la marque éponyme ;
- LiMo : contraction de « Linux Mobile », LiMo est un système ouvert basé, comme son nom l'indique, sur Linux ;
- MeeGo : Intel et Nokia ont récemment annoncé la fin du support de Moblin (basée sur un système Linux Fedora) et de Maemo (basée sur un système Linux Debian) en faveur de MeeGo qui vise un marché plus vaste que les téléphones et les netbooks (téléviseurs connectés notamment). Le cœur serait une reprise de Moblin et Qt serait utilisé pour le développement ;
- et bien d'autres systèmes plus marginaux, souvent développés pour des appareils spécifiques (PDA, etc.).

Android donne la réelle opportunité de voir se raréfier les systèmes spécifiques – souvent propriétaires – par fabricant d'appareil. Mais le chemin est long car, premièrement, il reste une variété conséquente de systèmes, qui fait du développement

d'applications portables sur divers appareils un réel casse-tête, et deuxièmement, des acteurs majeurs comme Samsung et son nouveau système Bada choisissent des stratégies à long terme dont les conséquences sont difficiles à prévoir.

Java, C++, Objective C – et d'autres – sont autant de langages pour autant d'applications. En touchant un grand nombre d'appareils avec une même plate-forme, Android va permettre à une même application d'être distribuée dans une même version.

Le kit de développement Android en détails

Exploiter une nouvelle plate-forme n'est jamais chose aisée. C'est pourquoi Google fournit, en plus du système d'exploitation, un kit de développement (*Software Development Toolkit* ou SDK). Ce SDK est un ensemble d'outils qui permet aux développeurs et aux entreprises de créer des applications.

Le SDK Android est composé de plusieurs éléments pour aider les développeurs à créer et à maintenir des applications :

- des API (interfaces de programmation) ;
- des exemples de code ;
- de la documentation ;
- des outils – parmi lesquels un émulateur – permettant de couvrir quasiment toutes les étapes du cycle de développement d'une application.

> **VOCABULAIRE Les API**
>
> Une API (*Application Programming Interface*) est un ensemble de classes regroupant des fonctions mises à disposition des développeurs. Ces fonctions ou méthodes peuvent être regroupées dans des bibliothèques logicielles ou des services. Le plus souvent, elles effectuent des traitements de bas niveau et proposent au développeur une interface de plus haut niveau pour qu'il puisse accéder à des fonctionnalités plus facilement et surtout plus immédiatement. Par exemple, la plupart des systèmes proposent une API graphique permettant d'afficher des éléments graphiques à l'écran (fenêtres, boutons, etc.) sans avoir à gérer le périphérique dans son intégralité et ce, pixel par pixel.

Le SDK Android est disponible gratuitement sur le site de Google.

> Prenez le temps de lire avec attention la licence d'utilisation.
> ▸ http://developer.android.com

Dans la suite de ce chapitre, nous allons détailler les étapes d'installation et de configuration afin que vous disposiez d'un environnement de développement optimal pour le reste des exemples de ce livre ainsi que vos futurs développements.

Documentation du SDK

La documentation du SDK Android est scindée en deux parties bien distinctes :

- le guide du développeur, disponible en HTML dans le répertoire du SDK que vous venez d'installer (voir la page `docs/guide/index.html`) ;
- la documentation des API au format javadoc, qui est également située dans le répertoire `docs` et accessible grâce au chemin toujours depuis le répertoire d'installation.

Pour les développeurs sous système Windows, ces chemins seront `docs\guide\index.html` et `docs\reference\packages.html`.

De façon générale, la page HTML `docs/index.html` contient les liens vers tous les documents du kit de développement et vers le site Internet de développement Android proposé par Google. Ainsi, si vous avez décompressé votre archive dans le répertoire `C:\android-sdk-windows-2.0`, la page HTML en question sera `C:\android-sdk-windows-2.0\docs\index.html`.

> **À PROPOS La javadoc**
>
> La documentation des classes et des différentes méthodes d'une bibliothèque (ou de l'API Java elle-même) est proposée au format HTML. Pour un développeur Java, cette documentation est incontournable pour trouver une classe ou une méthode mise à disposition par une classe et apprendre à s'en servir.
> Cette bible est générée automatiquement à partir des commentaires du code source formatés avec une syntaxe précise. Par exemple on utilise le tag `@param` pour décrire un des paramètres d'une méthode.
> L'outil permettant de produire cette documentation est `javadoc.exe`, disponible dans le répertoire `bin` des kits de développement Sun.
> Vous trouverez la liste complète des outils accompagnés de leur description sur le site officiel d'Android.
> ▸ http://developer.android.com/intl/fr/guide/developing/tools/index.html.

Les exemples

Le kit de développement est accompagné d'un certain nombre d'exemples illustrant les possibilités du SDK Android. Parmi ces exemples, que vous trouverez dans le répertoire `platforms\android-2.0\samples` de votre installation, il y a notamment :

- un jeu du serpent (répertoire `Snake`) ;
- un projet qui couvre l'utilisation de plusieurs pans de l'API Android comme les alarmes, les notifications, les menus, etc. (répertoire `APIDemos`) ;
- une application pour sauvegarder des notes (répertoire `NotePad`) ;
- … et bien d'autres exemples que nous vous laissons découvrir.

Nous verrons un peu plus tard comment utiliser ces exemples dans le logiciel de développement Eclipse (la création d'un projet sera nécessaire pour que les exemples soient exploitables).

Les outils de développement du SDK

Le SDK est livré avec un certain nombre d'outils couvrant différents aspects du cycle de développement d'une application Android. Ces différents outils seront présentés au fil des chapitres de ce livre.

Le kit de développement propose une boîte à outils complète pour les tâches de compilation, débogage, génération de code AIDL, signature de votre application, etc.

Le répertoire `tools` du kit de développement contient ainsi un ensemble de programmes dont voici quelques exemples :

- `DDMS` : est un outil de débogage puissant ;
- `mksdcard` : sert à créer des cartes mémoire logicielles utilisables avec l'émulateur ;
- `sqlite3` : permet d'accéder aux fichiers de données au format SQLite.

En plus des outils livrés avec le kit de développement, nous utiliserons dans ce livre une autre brique logicielle très utile : *Android Development Tools Plugin* ou ADT. Cet outil s'intègre directement à Eclipse et propose des interfaces et des assistants pour la création et le débogage des applications. Un outil qui nous fera assurément gagner beaucoup de temps !

Configurer votre environnement de développement

Les applications Android utilisent principalement la plate-forme Java pour s'exécuter. Il existe bien un framework natif nommé NDK (*Native Development Kit*) permettant de développer en C/C++, mais sa sortie récente et la difficulté de développer nativement des applications font que son utilisation restera certainement confidentielle, réservée à l'optimisation des performances des applications.

Pour développer et tester des applications en Java, n'importe quel environnement de développement convient. Il est tout à fait possible de ne pas utiliser d'environnement de développement intégré (ou IDE) si vous préférez employer votre éditeur de texte favori.

Pour le développement des applications Android, l'Open Handset Alliance et Google ont cependant souhaité mettre en avant Eclipse, l'environnement de développement de prédilection au sein de la communauté des développeurs Java. Si vous vous demandez les raisons de ce choix, voici quelques réponses :

- l'Open Handset Alliance tenait à mettre à disposition des développeurs des outils gratuits et ouverts. Eclipse est une plate-forme de développement Java réputée, largement utilisée, libre et gratuite et ne dépendant pas d'un éditeur en particulier ;
- Eclipse est disponible sous Windows, GNU/Linux et Mac OS. Cette portabilité entre les différents systèmes d'exploitation du marché signifie que les déve-

loppeurs peuvent utiliser cet IDE pour leurs développements sur n'importe quelle machine. Dans ce livre, nous utiliserons la version Windows d'Eclipse pour tous les exemples et les captures d'écran. Si vous utilisez Eclipse sur un autre système d'exploitation, certaines captures d'écran peuvent être légèrement différentes de celles que nous vous présentons, mais le fonctionnement sera le même.

Ce chapitre détaille dans un premier temps les différentes étapes pour installer et configurer un environnement Java, puis comment installer et configurer la plate-forme Android. Ensuite nous vous présentons l'installation et l'utilisation de l'environnement de développement Eclipse pour créer vos applications.

> **REMARQUE Utilisation de la langue anglaise**
>
> La plupart des copies d'écran sont en anglais : la plate-forme Android étant encore assez jeune, beaucoup de logiciels ne sont pour l'instant disponibles qu'en anglais.

Installer et configurer l'environnement d'exécution Java

Avant d'installer un environnement de développement, vous devez posséder un environnement d'exécution Java ou JRE (pour *Java Runtime Environment*) installé et correctement configuré sur votre machine, ainsi que le JDK (kit de développement de Java).

> **REMARQUE Différence entre le JDK et la JRE**
>
> La plupart des utilisateurs possèdent déjà une machine virtuelle JRE installée sur leur machine pour pouvoir naviguer sur Internet et exécuter certaines applications spécifiques à la plate-forme Java dans leur navigateur telles que les applets.
> Néanmoins, si la JRE permet d'exécuter des applications Java, elle ne vous permet pas d'en créer. Pour pouvoir créer des applications Java, vous devez télécharger et installer le JDK (*Java Development Kit*) qui contient tous les outils (compilateurs, débogueur, etc.) et les bibliothèques pour créer des applications Java. Sachez que le JDK contient également une JRE. Par conséquent, si vous installez la dernière version du JDK vous installerez également une JRE par la même occasion.

Pour télécharger le JDK, rendez-vous sur le site de Sun à l'adresse suivante : http://developers.sun.com/downloads. Dans la suite de cette partie, nous allons installer le JDK qui inclut une JRE.

❶ Une fois sur la page de téléchargement, cliquez ensuite sur *Java SE* puis *Java SE (JDK 6)*.

Pour les besoins de ce livre, nous avons installé la dernière version en date, à savoir « Java 6 Update 14 ». ❷ Une fois que vous avez cliqué sur le bouton de téléchargement *Download* de la page ci-contre, choisissez votre système d'exploitation et acceptez la licence Sun puis cliquez sur *Continue* pour arriver sur la page de téléchargement.

Figure 1–2
1 Page de sélection de téléchargements de Sun Micosystem

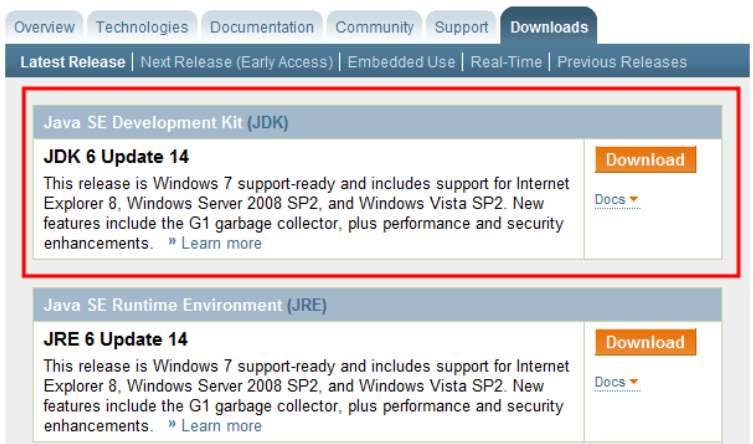


Figure 1–3
2 Page de téléchargement du JDK de Sun Microsystem

3 Cliquez sur le lien vers l'exécutable du programme d'installation du JDK – ici `jdk-6u14-windows-x64.exe` pour un système s'exécutant sous Windows 64 bits ; le navigateur demande si vous souhaitez exécuter ou enregistrer le fichier d'installation. Choisissez *Exécuter* pour lancer l'assistant d'installation immédiatement après le téléchargement de l'exécutable.

Figure 1–4
3 Sélection du système d'exploitation et de la langue du JDK Java

Figure 1–5
3 Page de téléchargement du JDK

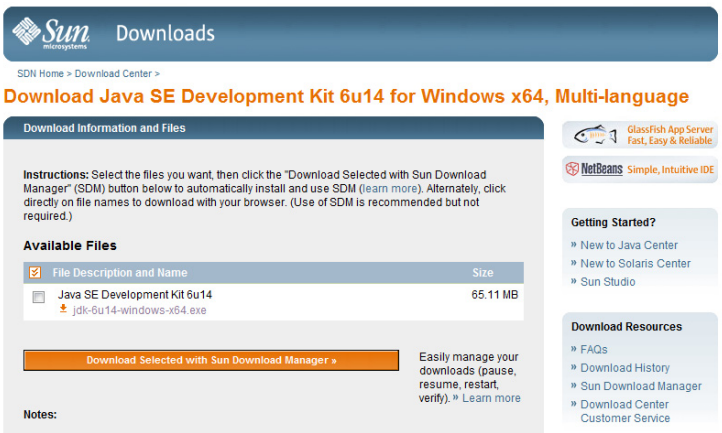
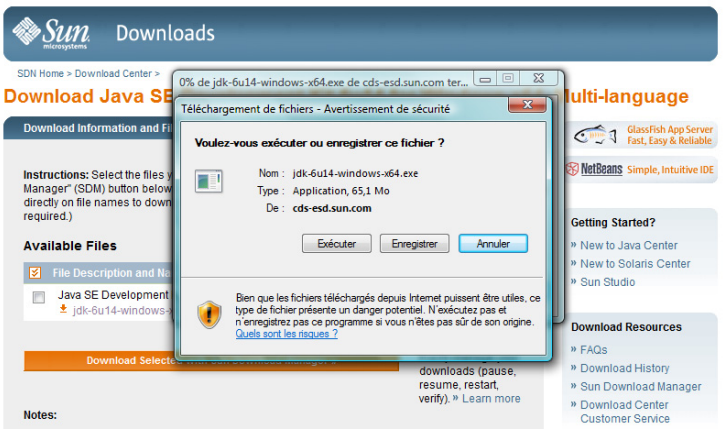
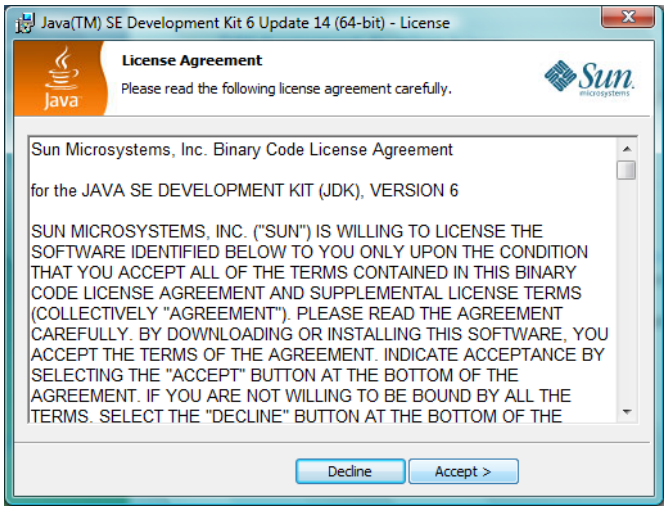


Figure 1–6
Téléchargement du JDK



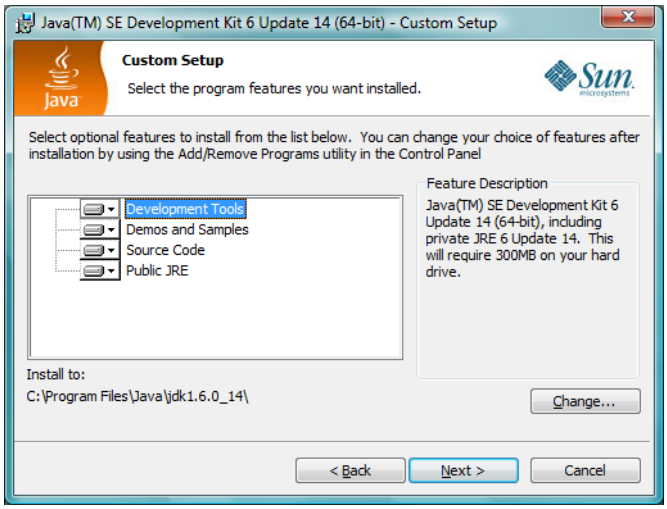
Pendant le processus d'installation, différents écrans vous demandent des informations. ④ Le premier écran est lié à l'acceptation de la licence d'utilisation que vous devrez accepter pour continuer l'installation. Lisez celle-ci et acceptez-la en connaissance de cause.

Figure 1–7
④ Licence d'utilisation du JDK à lire avant installation



⑤ L'écran suivant vous demandera les composants à installer. À moins d'être un développeur Java aguerri, laissez l'ensemble des cases cochées ; au minimum sélectionnez la JRE et les outils de développement.

Figure 1–8
⑤ Sélection des composants du JDK à installer



6 Cliquez ensuite sur le bouton *Suivant* pour procéder à l'installation et suivez les instructions. Une fois l'installation terminée, un écran devrait apparaître pour vous informer que le JDK et sa JRE ont été correctement installés.

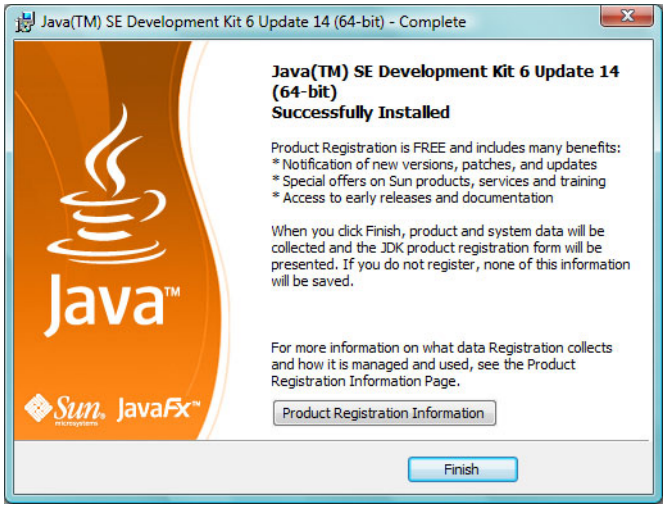


Figure 1–9
6 Installation avec succès du JDK

Pour paramétrer correctement votre système (quel qu'il soit), il reste une dernière étape, classique de l'installation d'un environnement Java : la configuration de la variable d'environnement `JAVA_HOME`, contenant le chemin d'installation de l'environnement Java. Pensez à configurer ce paramètre, nous en aurons besoin plus tard. En effet, beaucoup de bibliothèques ou de programmes Java ont besoin de savoir où les exécutables Java sont placés physiquement pour pouvoir les utiliser, les variables d'environnement (et notamment `JAVA_HOME`) sont spécifiées pour cela.

> **EN SAVOIR PLUS** **Configurer la variable d'environnement JAVA_HOME**
>
> Sous Windows, il faut naviguer dans les interfaces *Panneau de Configuration* > *Système* > *Avancé* > *Variables d'environnement*> *Nouveau*.
> Sous GNU/Linux, en fonction du shell, taper :
> - Avec sh/bash/zsh : `JAVA_HOME="...chemin..."; export JAVA_HOME`
> - Avec csh/tcsh : `setenv JAVA_HOME "...chemin..."`

Installer le kit de développement Android

Le kit de développement Android – plus communément appelé SDK – est en version 2.1 au moment de l'écriture de ce livre. Le processus d'installation a été largement bouleversé à plusieurs reprises entre les différentes versions proposées. Nous

déroulons une installation avec une version 2.0 dans ce chapitre. La méthodologie employée est également applicable à la version 2.1.

Pour télécharger ce kit de développement (SDK), disponible gratuitement, rendez-vous sur le site Android de Google à l'adresse : http://developer.android.com/sdk/index.html.

BONNES HABITUDES Signature de l'archive téléchargée

Peut-être allez-vous télécharger ces logiciels à partir d'un autre site où vous aurez vos habitudes, ou alors sur des tutoriels Internet qui vous donneront un lien de téléchargement ?
Un grand nombre de sites donne maintenant une signature des archives à télécharger calculée grâce à un algorithme appelé MD5 (*Message Digest 5*). Il s'agit d'une fonction de hachage qui calcule l'empreinte numérique d'un fichier avec une probabilité très forte que deux fichiers différents donnent deux empreintes différentes.
Quand vous téléchargerez vos fichiers, vérifiez que votre archive n'a pas été modifiée par rapport à celle distribuée par Google en comparant la clé affichée sur le site Android avec celle que vous aurez calculée grâce à un des multiples logiciels gratuits le permettant.
Vous serez également assuré d'avoir un contenu officiel mais également qu'aucun dysfonctionnement dans le téléchargement ne sera intervenu.

Téléchargez et décompressez l'archive qui contient les logiciels Android.

Toutes les captures d'écran et les manipulations effectuées dans ce livre sont basées sur un système Windows. Néanmoins, les étapes et les écrans rencontrés sont similaires sur tous les systèmes.

L'archive est présentée sous forme d'un répertoire dont le contenu est présenté dans la capture d'écran suivante.

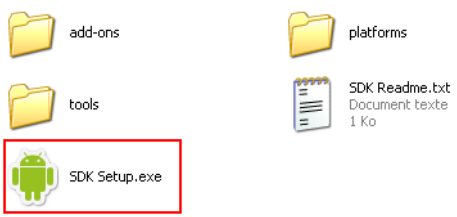


Figure 1–10
Contenu de l'archive téléchargée et décompressée

La version 2.0 introduit un exécutable, `SDK Setup.exe` qui, une fois lancé, propose une suite d'interfaces permettant de télécharger le kit de développement de façon modulaire. Exécutez ce programme.

OUPS Connexion HTTPS bloquée ?

Si un message d'erreur ressemblant à la capture d'écran suivante apparaît, un réglage est nécessaire. Cette erreur indique un problème de connexion HTTPS. Allez dans les préférences du programme en cliquant sur *Settings* et cochez la case *Force https://...*.

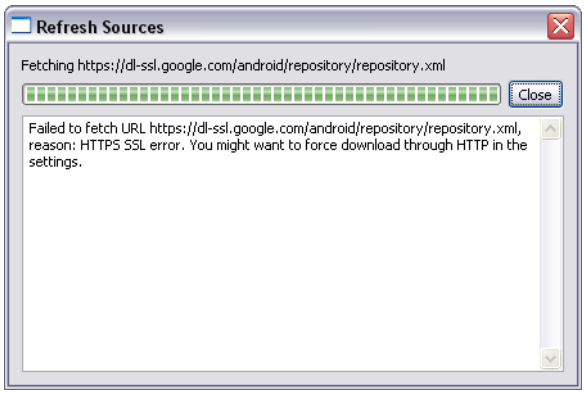


Figure 1–11 Fenêtre possible d'erreur au lancement

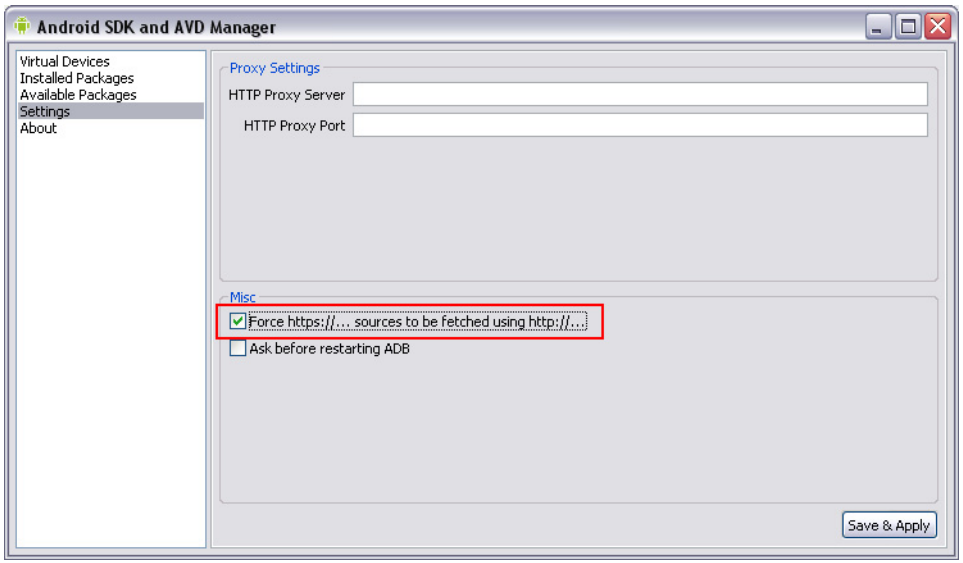


Figure 1–12 Réglage à effectuer pour contourner l'erreur.

Cliquez sur *Save & Apply* pour enregistrer les modifications. Vous pouvez en profiter pour régler les paramètres du proxy si votre connexion Internet est accessible au travers d'un serveur mandataire, ce qui est souvent le cas en entreprise. Puis fermez et relancez le programme SDK `Setup.exe`.

Une nouvelle fenêtre apparaît avec une liste de paquetages, c'est-à-dire ici de modules à installer. On aperçoit notamment le driver USB pour un téléphone de développement (téléphone conçu pour le développement d'Android) mais surtout les différentes versions de la plate-forme Android. Choisissez *Accept All* puis *Install Accepted*. Une fenêtre de progression de l'installation apparaît.

Figure 1–13
Liste des modules disponibles

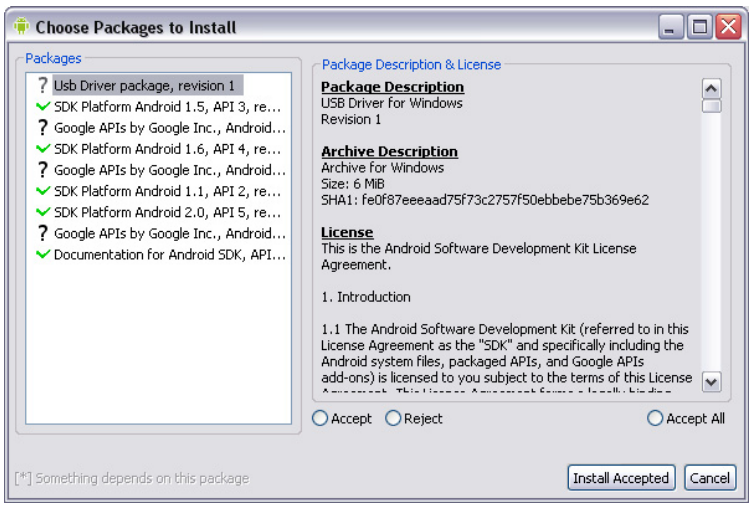
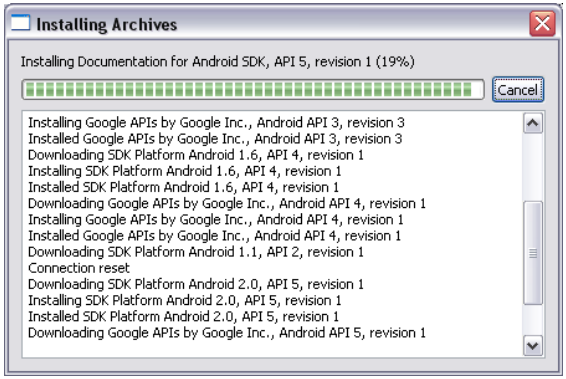
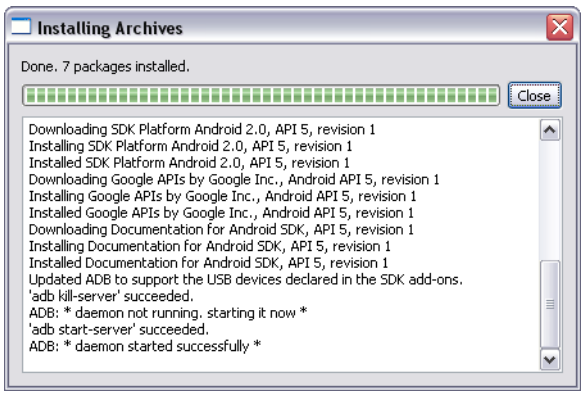


Figure 1–14
Fenêtre de progression de l'installation



Une fois l'installation terminée, cliquez sur le bouton *Close* qui remplacera le bouton *Cancel* disponible pendant la progression de l'installation. Un récapitulatif s'affiche.

Figure 1–15
Fenêtre récapitulative



Fermez la fenêtre, nous avons fini l'installation du kit de développement !

> **POUR ALLER PLUS LOIN** **Personnaliser l'installation**
>
> En utilisant l'installation proposée par défaut, nous avons installé une plate-forme 2.0 contenant les modules pour développer des applications compatibles avec les plates-formes 2.0 et 1.6.
> Si vous ciblez des plates-formes antérieures, il est encore temps de les installer en relançant l'application et en consultant les modules disponibles dans *Available Packages*. Vous trouverez des modules pour les plates-formes 1.5 et 1.1.

Installer et configurer Eclipse

Eclipse est une application Java et nécessite l'installation d'une JRE. Comme nous venons de réaliser cette installation, il ne reste plus qu'à se procurer une version d'Eclipse – 3.4 ou supérieure – depuis le site officiel du projet.

> ▸ http://www.eclipse.org/downloads

Pour obtenir un environnement de développement optimal, choisissez la version la plus adaptée à vos besoins. Eclipse possède un certain nombre de versions différentes, incluant ou non certains plug-ins. Pour les besoins (assez basiques) de ce livre, prenez la version de développement Java (voir figure 1-16).

Eclipse est fourni sous la forme d'une archive au format Zip, laquelle peut être décompressée directement à l'emplacement où vous souhaitez installer Eclipse. Une fois l'archive décompressée, Eclipse est prêt à être démarré en exécutant `eclipse.exe` à la racine de son répertoire d'installation.

Figure 1–16
Téléchargement de Eclipse pour les développeurs Java

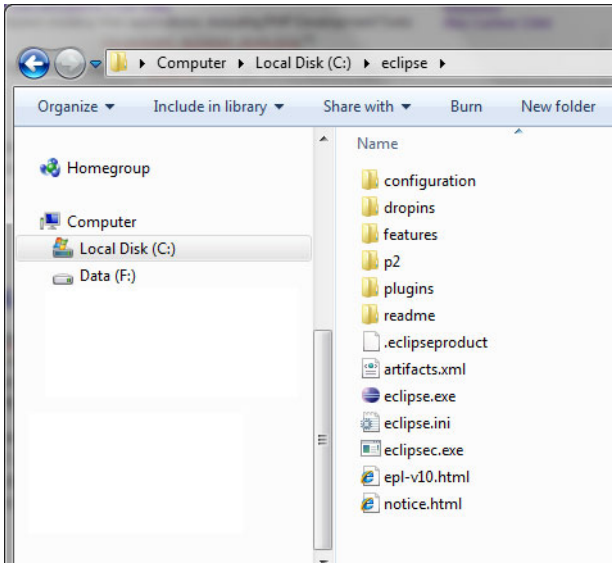


Figure 1–17
Après avoir décompressé l'archive d'Eclipse, vous pouvez lancer l'IDE en exécutant eclipse.exe.

Installer le module ADT pour Eclipse

Eclipse est un IDE (*Integrated Development Environment*) gratuit très prisé des développeurs Java et des entreprises. Pour faciliter le développement d'applications Android, Google propose un module, compatible avec cet IDE, nommé ADT (*Android Development Tools*).

Les exemples de ce livre utilisant Eclipse et le complément ADT, nous ne pouvons que vous recommander d'utiliser celui-ci. Utiliser Eclipse vous permettra également de gagner en efficacité de développement et de maintenir vos applications beaucoup plus simplement qu'en utilisant un autre IDE non conçu pour les développements Android.

ATTENTION Compatibilité entre versions du module ADT et du SDK Android

ADT supporte Android 1.5 à partir de sa version 0.9 uniquement. Une version inférieure d'ADT ne fonctionnera pas correctement.
La version 0.9.4 ou supérieure est recommandée pour la version 2.0 du kit de développement (SDK).
À noter, de façon générale, que la dernière version du SDK permet de créer des applications pour des versions antérieures de la plate-forme. Par exemple la version 2.0 vous permet de créer des applications pour les versions 1.1, 1.5 et 1.6. Nous vous recommandons donc d'installer la dernière version du kit de développement Android, avec la dernière version du module ADT.

Une fois Eclipse démarré, vous devez installer le module ADT. Le téléchargement et l'installation d'ADT sont des processus simples pour l'utilisateur et s'effectuent directement depuis Eclipse.

1 Pour récupérer la dernière version et l'installer en une seule étape, utilisez le système d'installation et de mise à jour *Help > Software Updates > Available Software.*

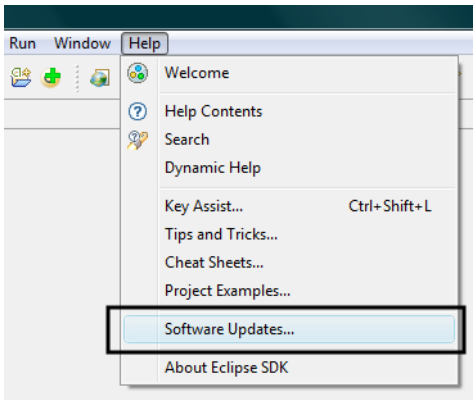


Figure 1–18
1 Comment afficher la fenêtre des mises à jour et des compléments ?

2 Depuis la fenêtre d'installation et de mise à jour, sélectionnez l'onglet *Available Software.*

3 Ensuite cliquez sur le bouton *Add Site* pour ajouter un site de téléchargement.

4 Une fois validé avec *OK*, le site s'ajoute dans la liste des sites.

Cochez le nouveau site et cliquez sur *Install*. Suivez les instructions en acceptant les différentes licences et redémarrez Eclipse (en cliquant sur *Oui*) lorsqu'il vous le demande.

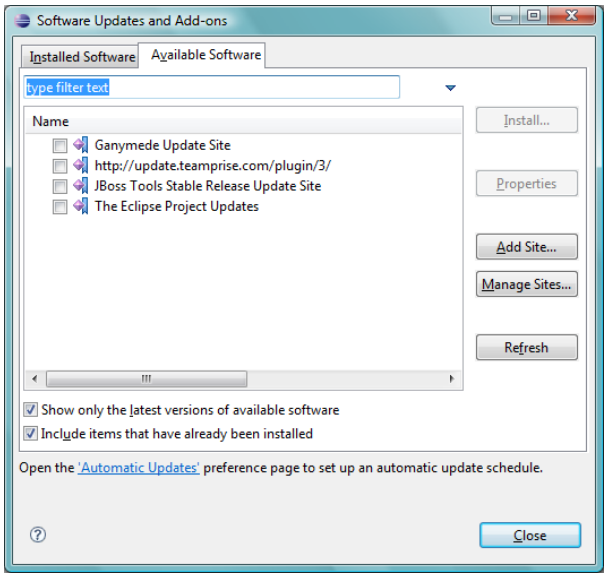


Figure 1–19
2 Liste des plug-ins installés

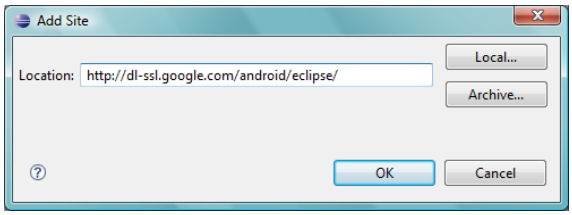


Figure 1–20
3 Saisissez l'adresse du site distant de ADT.

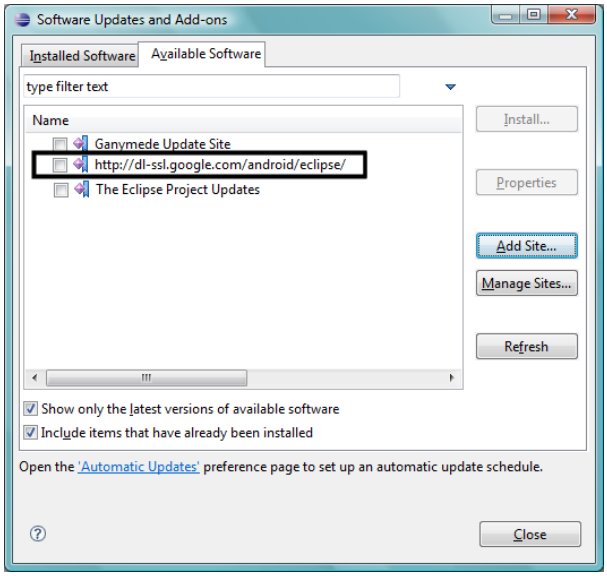


Figure 1–21
4 Le site du complément ADT est ajouté à la liste.

5 Une fois l'IDE redémarré, rendez-vous dans les préférences *Window > Preferences* pour configurer ADT et spécifier l'emplacement du SDK Android avant de valider avec OK.

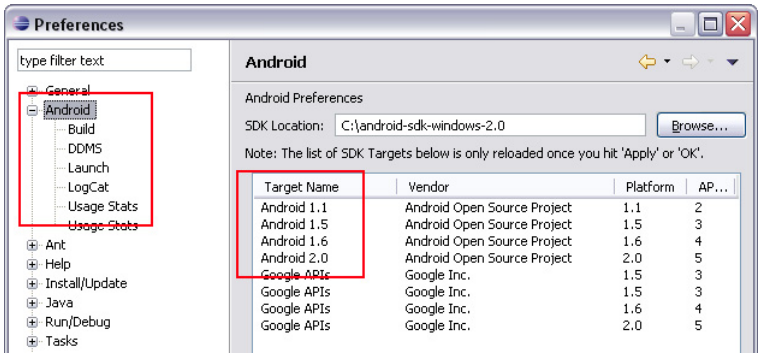


Figure 1–22
5 Configuration du SDK Android dans ADT

L'installation d'ADT devrait avoir enrichi l'interface d'Eclipse avec les éléments suivants :

- le bouton représentant un téléphone permet d'ouvrir le gestionnaire d'AVD (*Android Virtual Devices*) ;
- celui du milieu avec le signe d'addition permet de créer des projets Android ;
- le bouton orné du signe JU permet de créer un projet de test Android utilisant la bibliothèque de tests Junit ;
- le dernier bouton permet de créer des fichiers de ressources Android (disposition d'interface, valeurs, préférences, animation, menu, etc.).

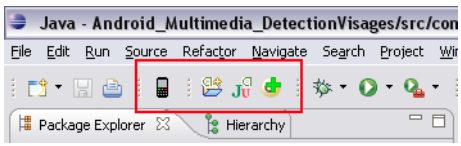


Figure 1–23
Outils rajoutés en accès rapide grâce au plug-in

Si vous n'observez pas les icônes Android, vérifiez que le module ADT est correctement installé et personnalisez la barre en effectuant dessus un clic droit et en sélectionnant *Customize Perspective...*. Ajoutez ensuite tous les éléments propres à Android (voir figure 1-24).

Configurer un appareil virtuel Android

Avant de vous lancer dans le développement, il vous faut au moins créer un profil d'appareil afin de pouvoir lancer votre application. Un profil d'appareil permet de

spécifier à l'émulateur la version du SDK que ce dernier doit lancer ainsi que le type d'écran, la taille de la carte SD, etc.

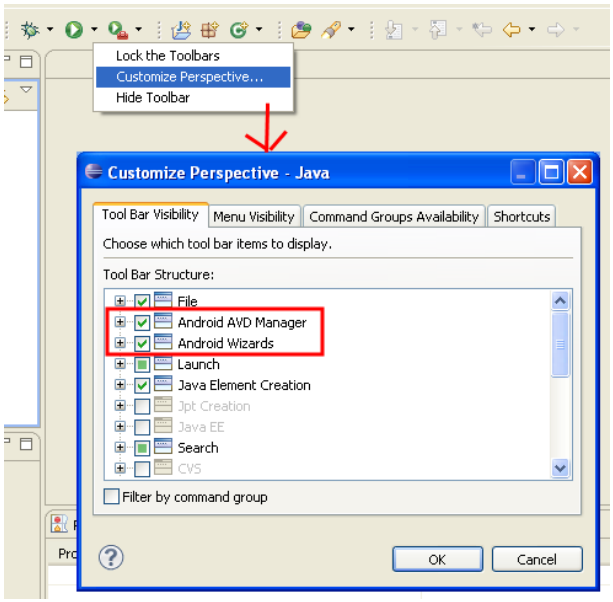


Figure 1–24
Ajout des icônes Android dans la perspective courante

Pour créer un tel profil, cliquez sous Eclipse sur le bouton du gestionnaire d'AVD (bouton représenté par une icône en forme de téléphone comme indiqué auparavant) ou naviguez dans *Window > Android SDK and AVD Manager*. Cliquez ensuite sur le bouton *New...* :

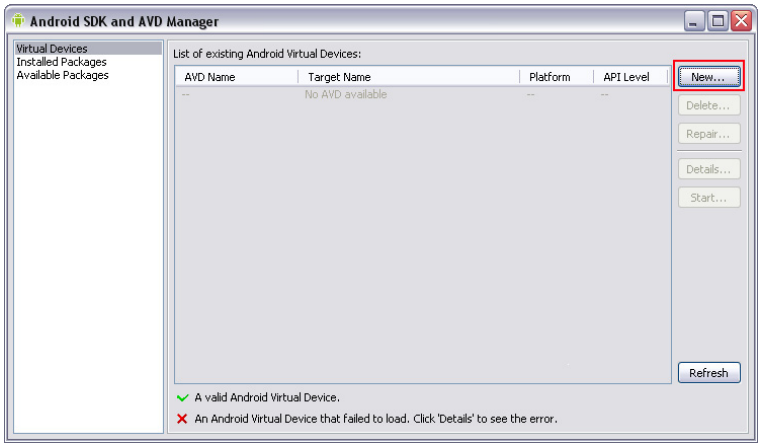


Figure 1–25
Le gestionnaire d'appareils virtuels Android

Une fois dans la fenêtre du gestionnaire d'AVD, remplissez les champs indiqués ci-après dans le tableau 1-2.

Tableau 1–2 Champs à remplir pour configurer un appareil virtuel

Name	Nom de l'appareil virtuel.
SDCard	Taille qui sera utilisée pour la simulation d'une carte mémoire de type SD Card. Choisissez une valeur en fonction de la taille des fichiers que vous voudrez stocker sur la carte, sachant que les téléphones actuels ont tendance à être livrés avec plusieurs centaines de méga-octets pour stocker musiques et vidéos. Pour les exemples de ce livre quelques méga-octets suffiront. De façon générale, une taille de « 64MB » (64 Mo) sera amplement suffisante pour la majorité de vos applications.
Target	La plate-forme cible à émuler. Les choix de cette fenêtre dépendront des paquetages que vous aurez sélectionnés. Si vous les avez tous installés, vous aurez la possibilité de choisir des versions de 1.1 à 2.1 et ainsi de construire des images pour émuler différentes versions de la plate-forme et mieux tester vos applications.
Skin	Choisissez le type d'écran simulé par le programme.

Pour créer l'AVD, cliquez sur le bouton *Create AVD*.

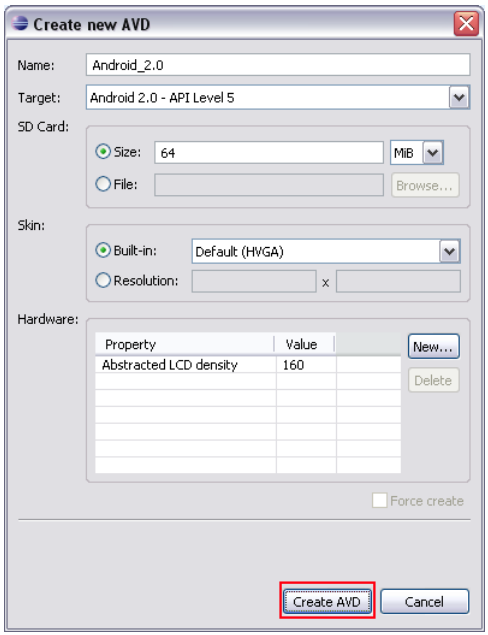


Figure 1–26
Création d'un appareil virtuel Android 1.5

Dans cette partie vous avez installé Java, le SDK Android, l'IDE Eclipse, le complément ADT et configuré un AVD pour exécuter votre application. Toutes ces étapes sont indispensables avant de pouvoir commencer à développer votre première application.

Vous disposez maintenant d'un environnement de développement Eclipse personnalisé et configuré pour développer sur la plate-forme Android.

Votre première application

L'objectif de cette partie est de vous familiariser avec les différents composants et la structure d'une application Android. Beaucoup d'éléments vont sembleront abstraits, voire incompréhensibles dans l'instant, néanmoins ne vous en formalisez pas, tous les éléments seront expliqués et détaillés dès le prochain chapitre.

Créer votre premier projet Android

Ouvrez Eclipse avec le complément ADT précédemment installé. Vous devriez apercevoir de nouvelles icônes dans le menu supérieur. Cliquez sur le second, celui en forme de dossier avec un *a+*, pour lancer l'assistant de création de projet Android.

Figure 1–27
Création d'un projet avec ADT dans Eclipse

> **Oups**
>
> Si vous n'observez pas ces icônes, veuillez vous reporter à la section « Installer le module ADT pour Eclipse ».

Après avoir cliqué sur l'icône de création de projet Android (ou *File > New > Android Project*), la fenêtre d'assistant s'ouvre pour vous proposer de renseigner les différents paramètres du projet.

Renseignez le nom du projet, la version d'Android cible, le nom de l'application ainsi que l'espace de nommage par défaut (le nom du paquetage). Vous pourrez aussi créer une activité directement depuis cette interface. Le champ de version minimale du SDK se remplit directement en fonction du choix de la version de la plate-forme.

En cliquant sur *Finish* (voir figure 1-28), ADT crée pour vous la structure du projet dans Eclipse.

L'arborescence du projet est découpée en plusieurs éléments (voir tableau 1-3).

Tous les fichiers sources iront dans le répertoire `src`. Une fois le projet compilé, vous devriez observer l'ajout d'un fichier `R.java`, qui vous fournira toutes les références des ressources incluses dans le répertoire `res` (ces références seront utilisées lorsque nous écrirons des applications dans les chapitres suivants). À l'instar de tous les fichiers générés, le fichier `R.java` ne sera mis à jour que si l'intégralité de votre application compile correctement.

Figure 1–28
Assistant de création projet Android

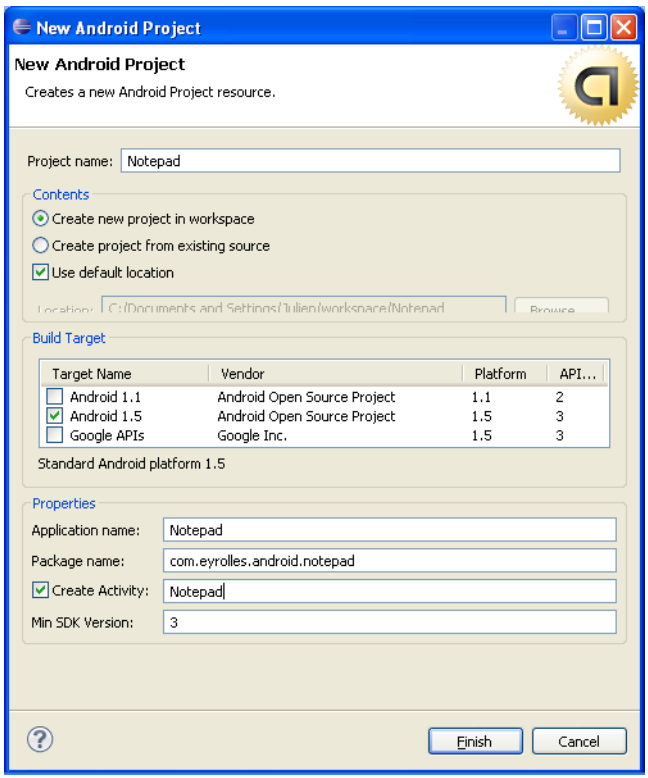


Figure 1–29
Arborescence du projet Notepad

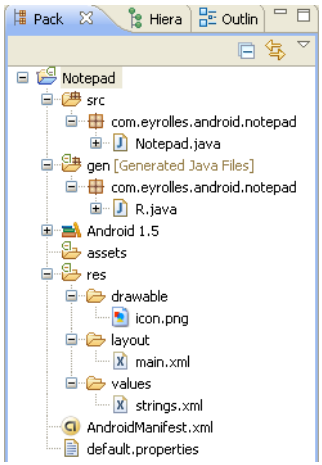


Tableau 1–3 Arborescence d'un projet Android

`src`	Le répertoire de l'ensemble des sources du projet. C'est dans ce répertoire que vous allez ajouter et modifier le code source de l'application.
`libs`	Contient les bibliothèques tierces qui serviront à votre application.
`res`	Contient toutes les ressources telles que les images, les dispositions de l'interface graphique, etc. nécessaires à l'application. Ces ressources seront accessibles grâce à la classe `R` décrite plus bas.
`gen`	Contient l'ensemble des fichiers générés par ADT afin d'assister le développeur. Si vous supprimez un fichier dans ce répertoire, ADT s'empressera aussitôt de le régénérer. Attention, si le projet contient des erreurs ne permettant pas sa compilation, ADT ne générera aucun fichier. Par conséquent, si vous ne trouvez pas la bonne version de fichier ou si le fichier n'est pas généré automatiquement, vérifiez bien que l'ensemble du projet compile.
`assets`	Contient toutes les ressources brutes (*raw bytes*) ne nécessitant aucun traitement par ADT ou Android. À la différence des ressources placées dans le répertoire `res`, les ressources brutes seront accessibles grâce à un flux de données et non grâce à la classe `R` décrite plus loin.
`AndroidManifest.xml`	Le fichier XML décrivant l'application et ses composants – activités, services, etc.
`default.properties`	Fichier de propriétés utilisé pour la compilation.

Le répertoire `res` est composé de différents sous-répertoires dont les suivants :

- `res/drawable` : contient les ressources de type image (PNG, JPEG et GIF) ;
- `res/layout` : contient les descriptions des interfaces utilisateur ;
- `res/values` : contient les chaînes de caractères, les dimensions, etc. ;
- `res/xml` : contient les fichiers XML supplémentaires (préférences, etc.) ;
- `res/menu` : contient la description des menus ;
- `res/raw` : contient les ressources autres que celles décrites ci-dessus qui seront empaquetées sans aucun traitement.

> La notion de ressource sera développée dans le chapitre « Création d'applications et découverte des activités ».

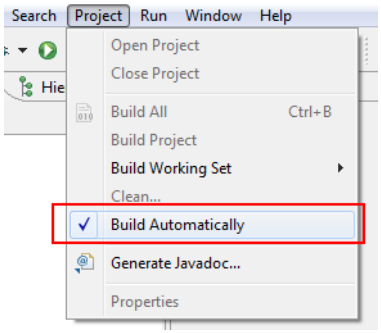
Pour le moment, ne modifiez pas le code ou la configuration du projet. Compilez le projet en l'état afin de l'exécuter.

> **REMARQUE** **Comment utiliser les exemples du SDK ?**
>
> À noter, si vous souhaitez utiliser les exemples fournis dans le kit de développement (répertoire `platforms\android-2.0\samples`), les sources de ces applications ne comportent pas de fichier de projet. Pour pouvoir utiliser ces sources dans Eclipse, vous devrez créer un nouveau projet Android puis importer tous les fichiers de l'exemple dans la structure de votre projet nouvellement créé.

Compiler une application Android

La méthode de compilation d'une application Android dépend de la configuration de votre environnement Android. Par défaut, les projets sont compilés automatiquement dès que vous enregistrez un fichier source. Pour vérifier ce dernier paramètre, allez dans *Project > Build Automatically*.

Figure 1–30
Compilation automatique sans intervention du développeur



La compilation automatique de l'application Android (effectuée par défaut à chaque sauvegarde) – jusqu'à la génération de l'exécutable `.apk` – ne s'effectue que si votre code ne contient pas d'erreurs. S'il reste des erreurs, vous devrez les résoudre avant de lancer à nouveau l'application dans votre émulateur (ou sur votre appareil physique). L'application qui s'exécutera sera la dernière version compilée : s'il reste des erreurs non résolues, il s'agira alors de l'application compilée avant vos modifications et elle ne prendra pas en compte vos derniers changements.

Vous pouvez également configurer Eclipse de façon à compiler uniquement sur action de l'utilisateur, ce qui évitera une compilation fréquente en tâche de fond, peu souhaitable sur les ordinateurs avec peu de ressources.

Quand vous compilez une application, le résultat sera placé selon l'arborescence suivante :

- `bin/classes` : les classes compilées en `'.class'` ;
- `bin/classes.dex` : représente l'exécutable compilé pour la machine virtuelle Dalvik ;
- `bin/monapplicaiton.zip` : contient les ressources de votre application regroupées et compressées dans une archive `.zip` ;
- `bin/monapplication.apk` : il s'agit de votre application compilée et empaquetée pour être déployée dans le système Android. Ce fichier contient le fichier `.dex`, les ressources compilées et non compilées (celles contenues dans le répertoire `/raw`) et enfin le fihier de configuration de l'application.

Exécuter une application Android

Pour exécuter une application Android depuis Eclipse, inutile de changer vos habitudes de développeur : cliquez sur le bouton de lancement d'exécution d'application.

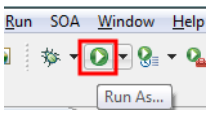


Figure 1–31
Lancement d'une application Android

Si vous n'avez pas encore créé la configuration d'exécution de votre projet, une fenêtre apparaît pour vous demander le type de configuration que vous souhaitez créer.

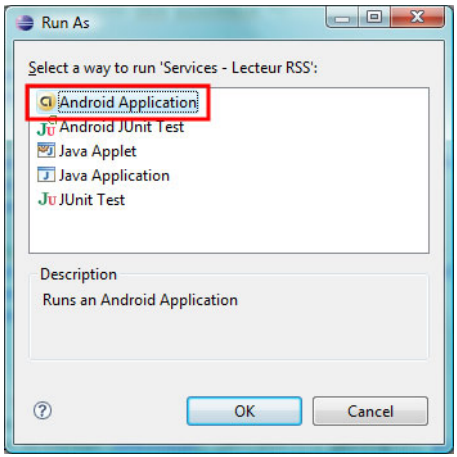


Figure 1–32
Spécifiez le type de *configuration d'exécution* pour exécuter une application Android.

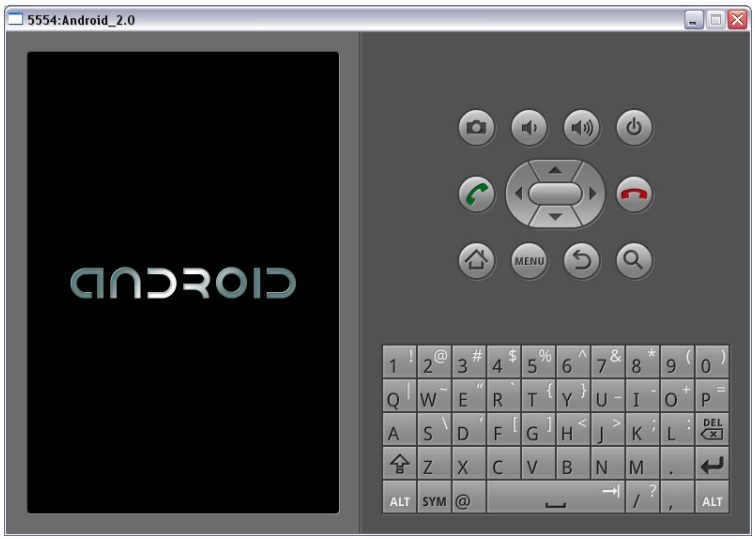


Figure 1–33
Lancement de l'*émulateur* pour exécuter votre application

Sélectionnez le type de configuration voulue, soit *Android Application* pour spécifier l'exécution de l'application dans l'environnement d'exécution Android, c'est-à-dire l'émulateur (voir figure 1-32). Cliquez sur *OK* pour lancer votre application, vous devriez observer le démarrage de l'émulateur (voir figure 1-33). Cela peut durer entre 30 secondes et une minute : soyez donc un peu patient et surtout ne le refermez pas après avoir testé votre application, vous perdriez beaucoup de temps (un peu comme si vous éteigniez votre téléphone après chaque utilisation).

Maîtriser l'émulateur Android

Pour les personnes qui ne possèdent pas de téléphone ou dans le cas d'une équipe ne pouvant mettre à disposition un appareil par développeur, l'émulateur représente la meilleure façon de tester son application. Il émule les différents périphériques disponibles sur les téléphones (par exemple un *trackball*) et permet d'exécuter le système Android et de pouvoir déployer et exécuter les applications que vous développez.

Afin d'être plus efficace, voici les raccourcis ou combinaisons de touches utiles.

Tableau 1–4 Raccourcis clavier à utiliser dans l'émulateur

Touche	Description
Touche *9* pavé numérique (*Fn* + *9* sur les ordinateurs portables)	Effectue une rotation à 90 degrés de l'émulateur (mode paysage). Une seconde pression sur la touche permet de remettre l'émulateur dans le mode portrait.
Touche *+* (*Fn* + '*+*' sur les ordinateurs portables)	Augmente le volume.
Touche *–* (*Fn* + '*-*' sur les ordinateurs portables)	Diminue le volume.
F8	Active/Désactive le réseau cellulaire.
Alt + *Entrée*	Active/Désactive le mode plein écran.
F6	Active/Désactive le trackball (maintenez la pression sur la touche *Suppr*. pour l'activer temporairement)

Déboguer une application Android

Une chose est sûre, développer une application nécessitera forcément du temps, beaucoup de temps, pour résoudre les problèmes de code ou de conception auxquels même les meilleurs développeurs sont confrontés chaque jour. Le débogage est donc une étape essentielle dans le cycle de vie du développement d'une application.

À l'instar de l'exécution, le module ADT permet aussi le débogage de l'application de la même façon qu'avec n'importe quelle autre application Java.

Au lieu de cliquer sur le bouton d'exécution, cliquez sur le bouton de débogage. Si vous aviez précédemment créé la configuration d'exécution pour votre application Android, vous ne devriez pas rencontrer de fenêtre intermédiaire. Pour tester votre application, cliquez sur la flèche située à droite du bouton de débogage.

Figure 1–34
Lancer le débogage avec la *configuration d'exécution* par défaut.

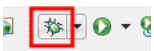
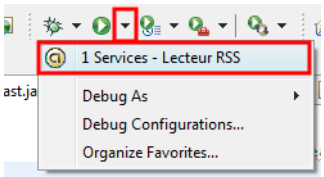


Figure 1–35
Lancer le débogage d'une *configuration d'exécution* précise.



Pour déboguer votre code, il est utile d'ajouter des points d'arrêt dans votre code. Ces points d'arrêt servent à stopper momentanément l'exécution de votre application en vous permettant d'étudier les différentes valeurs des variables ou de les changer au besoin. Pour ajouter un point d'arrêt à votre code, vous pouvez utiliser le menu *Run > Toggle Breakpoint* ou directement utiliser le raccourci *Ctrl + Maj + B*.

Figure 1–36
Ajouter ou supprimer un point d'arrêt dans votre code.

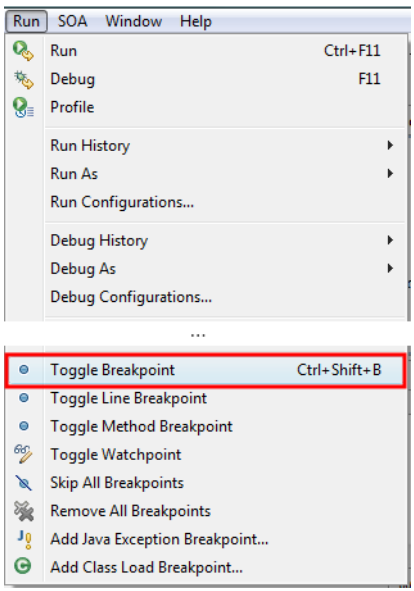
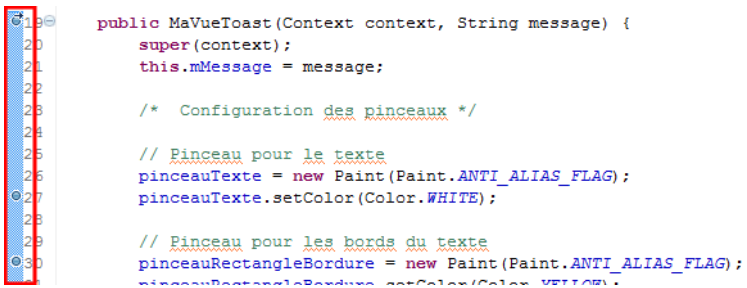


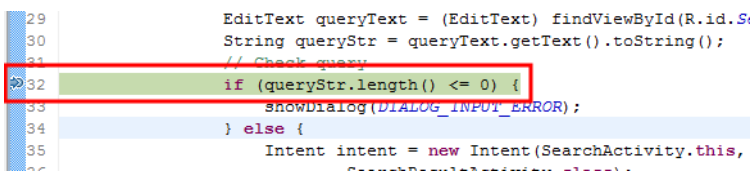
Figure 1–37
Les points d'arrêt peuvent se placer sur des lignes ou des méthodes.



Ce mode permet également d'exécuter l'application pas à pas, afin de suivre le cheminement complet de l'exécution de l'application. Par défaut, les touches usuelles pour déboguer une application sont les suivantes :

- *F8* : continuer l'exécution en sortant du mode pas à pas ;
- *F5* : exécution pas à pas en entrant dans les méthodes appelées dans le code ;
- *F6* : exécution pas à pas sans les méthodes appelées dans le code ;
- *F7* : exécution pas à pas en sortant de la méthode actuelle.

Figure 1–38
Exécution pas à pas d'une application Android dans Eclipse



B.-A.-BA Débogage et test de performances

Exécuter une application en mode débogage ralentit fortement l'exécution de votre application. Ne testez pas les performances de votre application dans ce mode mais dans le mode d'exécution normal.

Le module ADT d'Eclipse permet de suivre l'exécution de votre application sur l'émulateur pendant toute la durée du débogage. Pour cela, sélectionnez tout d'abord l'option de débogage *DDMS* en haut à droite de votre environnement Eclipse.

Figure 1–39
Sélection de la vue DDMS

La vue DDMS ouvre une perspective sur un ensemble d'interfaces permettant de suivre l'activité de l'émulateur : détail des tâches et de la pile des applications, explorateur de fichier, liste des applications s'exécutant dans l'émulateur et console d'administration de l'émulateur (simulation d'appels, d'envoi de SMS, etc.).

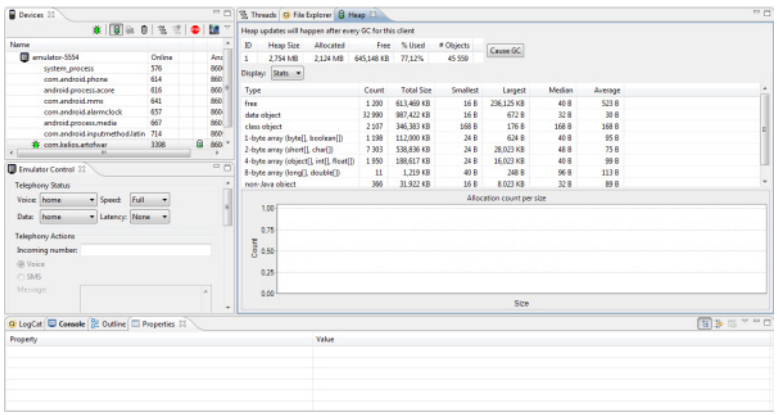


Figure 1–40
La perspective DDMS d'ADT dans Eclipse

Pour visualiser les threads , vous devez au préalable sélectionner l'application dans la vue *Devices* puis cliquer sur les boutons de la vue pour activer la mise à jour des threads et de la pile.

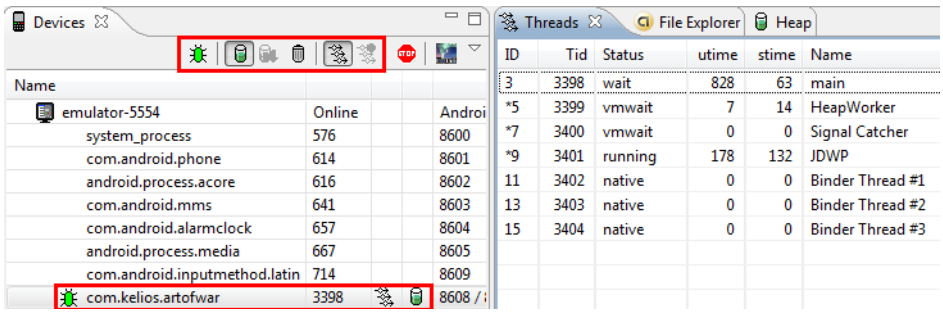


Figure 1–41 Utilisation de la vue des threads de DDMS

La perspective DDMS propose également la vue *LogCat* qui agit comme un véritable filtre d'entrées de journal (erreur, avertissement, information et débogage) pour les applications utilisant l'API de journalisation d'Android (voir figure 1-42).

Cliquez sur les boutons en haut à droite de la vue pour filtrer le niveau des entrées, ou utilisez le champs de recherche du bas pour filtrer précisément les entrées souhaitées.

Pour suivre l'exécution d'un thread lorsque vous déboguez une application, utilisez de préférence la perspective *Debug* et ses vues associées comme pour n'importe quel développement Java (voir figure 1-43).

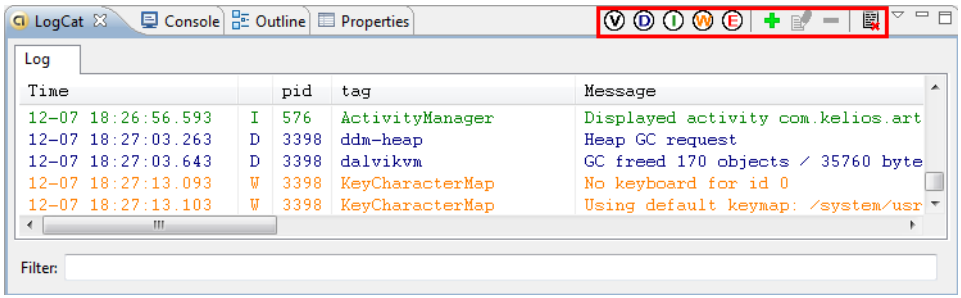


Figure 1–42 La vue LogCat permettant de filtrer les entrées.

Figure 1–43
L'ensemble des threads de l'application s'exécutant

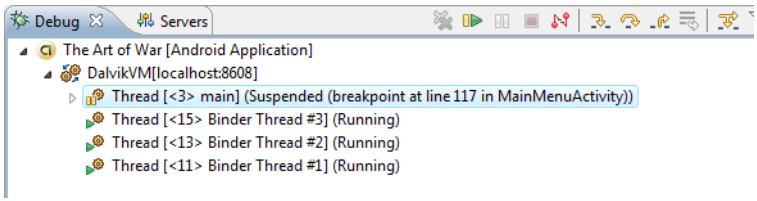
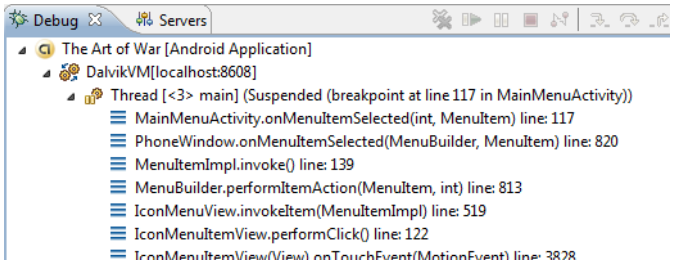


Figure 1–44
La vision d'un thread suspendu par le débogage



Grâce à l'environnement de développement Eclipse et aux outils d'ADT, les étapes de débogage sont rendues plus simples et rapides, vous permettant d'être efficace dans votre création.

> **ALTERNATIVE Utiliser les outils individuellement hors Eclipse**
>
> Si vous ne souhaitez pas utiliser Eclipse comme environnement de développement, le guide du développeur Android, accessible sur le site du SDK Android, vous conseillera pour l'utilisation individuelle des outils disponibles, comme l'outil DDMS que nous avons évoqué.
>
> ▸ http://developer.android.com/guide/developing/debug-tasks.html

En résumé

Dans ce chapitre, nous vous avons montré comment configurer votre environnement de développement, créer et déboguer votre premier projet sur l'émulateur. En annexe, vous trouverez également comment paramétrer votre environnement Android pour utiliser un téléphone de développement.

Les chapitres suivants introduisent les concepts de base Android, aussi prenez le temps de les parcourir avant d'accéder aux différents thèmes traités comme le stockage des données, le réseau, la 3D...

Enfin, détail dont vous n'êtes peut-être pas conscient, vous venez de créer votre premier projet – vous faites donc partie de la communauté des développeurs Android !

2

Création d'applications et découverte des activités

Sans connaissance de sa structure et de son cycle de vie, pas d'application possible. Ce chapitre vous présentera les différents composants d'une application Android et leurs interactions.

Une application Android est un assemblage de composants liés grâce à un fichier de configuration. Nous allons découvrir chaque pièce de notre puzzle applicatif, comment le fichier de configuration de l'application la décrit et comment toutes les pièces interagissent entre elles.

Avant de rentrer dans le détail d'une application Android et de son projet associé, différents concepts fondamentaux sont à préciser :

- les activités ;
- les vues et contrôles (et leur mise en page) ;
- les ressources ;
- le fichier de configuration appelé également manifeste.

Les *vues* sont les éléments de l'interface graphique que l'utilisateur voit et sur lesquels il pourra agir. Les vues contiennent des composants, organisés selon diverses mises en page (les uns à la suite des autres, en grille…).

Les *contrôles* (boutons, champs de saisie, case à cocher, etc.) sont eux-mêmes un sous-ensembles des vues (nous y reviendrons ultérieurement). Ils ont besoin d'accéder aux textes et aux images qu'ils affichent (par exemple un bouton représentant un téléphone aura besoin de l'image du téléphone correspondante). Ces textes et ces images seront puisés dans les fichiers *ressources* de l'application.

Une *activité* peut être assimilée à un écran structuré par un ensemble de vues et de contrôles composant son interface de façon logique : elle est composée d'une hiérarchie de vues contenant elles-mêmes d'autres vues. Une activité est par exemple un formulaire d'ajout de contacts ou encore un plan Google Maps sur lequel vous ajouterez de l'information. Une application comportant plusieurs écrans, possédera donc autant d'activités.

À côté de ces éléments, se trouve un fichier XML : le *fichier de configuration* de l'application. C'est un fichier indispensable à chaque application qui décrit entre autres :

- le point d'entrée de votre application (quel code doit être exécuté au démarrage de l'application) ;
- quels composants constituent ce programme ;
- les permissions nécessaires à l'exécution du programme (accès à Internet, accès à l'appareil photo...).

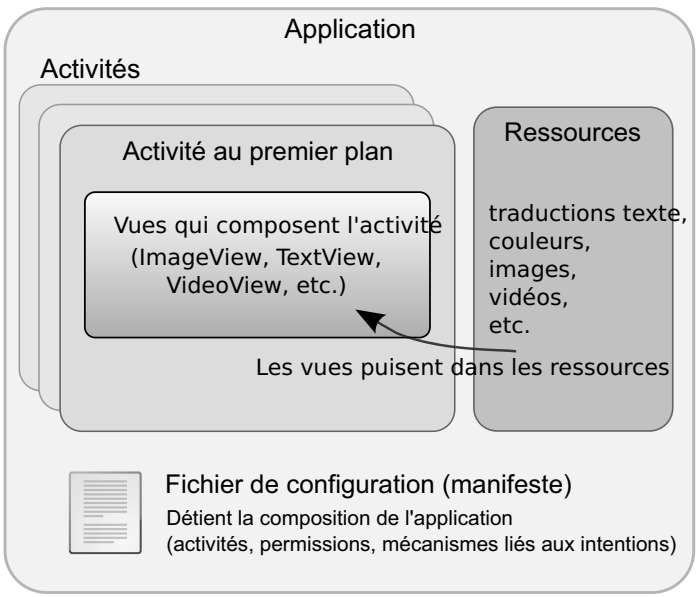


Figure 2–1
Composition d'une application

Rassembler les pièces du puzzle d'une application Android

Une application Android est composée de plusieurs éléments qu'il faut assembler pour obtenir un tout cohérent. Plus une application est complexe, plus le nombre de pièces utilisées sera grand. Voici donc les différents pièces principales du « puzzle » Android :

- activités ;
- services ;
- fournisseurs de contenu ;
- gadgets ;
- objets `Intent` ;
- récepteurs d'Intents ;
- notifications.

Dans ce chapitre nous présentons uniquement l'utilité des composants de base, sachant que chaque élément est détaillé au fil du livre.

Composants applicatifs : activité, service, fournisseur de contenu et gadgets

Parmi les éléments précédents, quatre sont des composants applicatifs.

L'*activité* représente le bloc de base d'une application. Elle correspond à la partie présentation de l'application et fonctionne par le biais de vues qui affichent des interfaces graphiques et répondent aux actions utilisateur. Nous verrons dans le chapitre traitant des interfaces utilisateur avancées qu'une activité correspond à un écran mais qu'elle peut contenir plusieurs vues.

Le *service* est un composant qui fonctionne en tâche de fond, de manière invisible. Ses principales utilisations sont la mise à jour de sources de données ainsi que d'activités visibles et le déclenchement de notifications. Le chapitre traitant des services et de la gestion des threads présente en détail leur utilisation.

Le *fournisseur de contenu* permet de gérer et de partager des informations. Un même fournisseur permet d'accéder à des données au sein d'une application et entre applications. C'est l'objet du chapitre 7 traitant du partage des données.

Le *gadget* est un composant graphique qui s'installe sur le bureau Android. Le calendrier qui affiche de l'information ou le lecteur audio qui permet de contrôler la lecture de fichiers sont deux exemples de gadgets que l'on trouve souvent sur un écran d'accueil.

Tableau 2–1 Les différents composants applicatifs Android et les classes associées

Nom	Classe ou paquetage concerné(e)
Activité	`android.app.Activity`
Service	`android.app.Service`
Fournisseurs de contenu (*content provider*)	`android.content.ContentProvider`
Gadgets (*app widget*)	`android.appwidget.*`

Éléments d'interaction : intents, récepteurs, notifications

Les éléments suivants permettent l'interaction entre les différents composants du système, entre les applications installées sur l'appareil ou avec l'utilisateur.

L'objet `Intent` : il permet de diffuser des messages en demandant la réalisation d'une action. L'accès aux autres applications et au système étant restreinte par le modèle de sécurité Android, ces objets permettent aux applications de fournir ou demander des services ou des données. La transmission se fait à travers tout le système et peut cibler précisément une activité ou un service. Tous ces concepts sont largement discutés dans le chapitre 4 traitant du mécanisme des Intents.

Récepteur d'Intents : il permet à une application d'être à l'écoute des autres afin de répondre aux objets `Intent` qui lui sont destinés et qui sont envoyés par d'autres composants applicatifs.

Notification : une notification signale une information à l'utilisateur sans interrompre ses actions en cours.

Tableau 2–2 Les différents mécanismes d'interaction avec leurs classes associées

Nom	Classe concernée
Intent *(Intent)*	`android.content.Intent`
Récepteur d'Intents *(Broadcast Receiver)*	`android.content.BroadcastReceiver`
Notification (*Notification*)	`android.app.Notification`

Filtres (d'Intents) : un objet `Intent` peut mentionner explicitement un composant cible. Si cette information n'est pas renseignée, Android doit choisir le meilleur composant pour y répondre. Ceci est mené à bien via une comparaison de l'objet `Intent` avec les filtres des différentes applications cibles. Les filtres ne se manipulent généralement pas via l'API mais sont paramétrables grâce à la balise `<intent-filter>` du fichier de configuration.

Permissions

Certaines opérations sont réalisables à condition d'en obtenir la permission. Ces actions sont de plusieurs formes :

- opérations pouvant entraîner un surcoût (connexion, échange de données, envoi de SMS par exemple) ;
- utilisation de données personnelles (accès à vos contacts, à votre compte Google, exploitation de vos informations linguistiques entre autres) ;
- accès au matériel du téléphone (prise de clichés, écriture sur la carte mémoire...).

Si vous utilisez les fonctionnalités liées à de telles permissions, vous devrez déclarer leur utilisation dans le fichier de configuration qui décrit l'application. À l'installation de votre application, l'utilisateur disposera d'un récapitulatif de toutes les permissions demandées pour que l'application fonctionne. Il pourra alors choisir de continuer ou d'interrompre l'installation en connaissance de cause.

Vous venez de J2ME

Pour les développeurs Java ME (Java Micro Edition appelé également J2ME), vous noterez de grandes ressemblances avec le système de permissions de la plate-forme de Sun.

Cycle de vie d'une application : gestion des processus

Les applications Android ont un fonctionnement particulier : elles réagissent à des changements d'états imposés par le système (démarrage, pause, reprise, arrêt...). Néanmoins elles n'ont aucun contrôle direct sur leur propre cycle de vie. En cas de besoin, le système peut mettre en pause ou alors complètement arrêter une activité s'il juge nécessaire de le faire, par exemple si l'application consomme trop de ressource processeur ou mémoire. Ce comportement est dit non déterministe.

Chaque application fonctionne dans son propre processus. Le système Android est responsable de la création et de la destruction des processus et gère ses ressources avec comme objectif la sécurité mais aussi la disponibilité et la réactivité de l'appareil. Ainsi un processus consommant beaucoup de ressources processeur, ne pourra entraver d'autres fonctionnalités, comme la réponse à un appel entrant. Ceci implique qu'un processus peut être tué à n'importe quel moment sans son consentement pour libérer des ressources nécessaires à d'autres applications.

L'ordre d'arrêt des processus pour libérer des ressources est déterminé par la priorité du processus donnée par le système. La priorité de l'application est elle-même déterminée par son composant le plus prioritaire (une application possédant un fournisseur de contenu a par exemple plus de priorité qu'une application qui en est dépourvue).

Deux applications de même priorité seront départagées par le temps qu'elles auront passé à une priorité plus faible.

Tableau 2–3 Priorités établies par le système en fonction de l'état de ses composants

Processus	Priorité	Description
Processus actif	Maximale	Processus au premier plan qui héberge des applications dont des composants interagissent avec l'utilisateur. C'est un processus qu'Android protège en libérant des ressources.
Processus visible	Élevée	Processus visible mais inactif, il contient des activités au moins en partie visibles mais pas au premier plan.
Processus d'un service démarré	Élevée	Processus qui réalise des actions sans interface graphique visible. Il a donc une priorité moindre que les processus vus plus haut, mais il est considéré comme de premier plan à la différence des processus en tâche de fond. Il est conservé le plus longtemps possible.
Processus en tâche de fond	Faible	Processus hébergeant des activités non visibles ou des services non démarrés. Ces processus plus nombreux seront plus facilement éliminés en fonction de leur utilisation.
Processus en fin de vie	Minimale	Android dispose d'un cache d'applications qui ont terminé leur cycle de vie, à des fins d'optimisation lors de leur redémarrage.

Reprenons l'exemple concret décrit dans les vidéos publiées par des développeurs de Google disponibles sur Internet à l'adresse http ://www.youtube.com/user/androiddevelopers. Le scénario est le suivant : un utilisateur est sur l'écran d'accueil et décide de consulter ses courriels dans sa boîte de réception. Une fois sur la liste de ses courriels, il veut en visualiser un. Un lien présent dans ce courriel l'intrigue, il veut le consulter en cliquant dessus, ce qui lance le navigateur. À partir du navigateur, l'utilisateur souhaite consulter des informations via l'application Google Maps. Enfin, une fois sa consultation achevée, il retourne à la consultation de ses messages.

Que se passe-t-il au niveau des quatre applications tournant dans quatre processus différents ?

Avant toutes ces actions, deux processus sont démarrés :

- le processus système qui contient le gestionnaire d'activités et la pile d'appels utilisée par toutes les activités, quel que soit le processus dans lequel elles figurent ;
- le processus `home` qui contient l'activité qui gère l'écran d'accueil.

Avant de lancer l'activité de messagerie, le système sauvegarde l'état de l'activité `home` (qui correspond à l'écran d'accueil) vers le processus système (cette sauvegarde sert en cas de problème). Ensuite le système peut créer le processus de la messagerie et lancer l'activité qui affichera la liste des courriels de l'utilisateur.

L'utilisateur sélectionne un courriel et avant de créer l'activité pour visualiser cet e-mail, l'état de l'activité précédente est également sauvegardé. Pour rentrer un peu plus dans le détail de cette sauvegarde, ce ne sont pas les courriels qui sont sauvegardés, mais l'état de l'activité pour savoir quel courriel est sélectionné par exemple. Une fois cette sauvegarde réalisée, le système peut lancer l'activité suivante qui affiche le détail d'un courriel. Cette activité est lancée au sein du même processus `mail`.

L'action du clic sur le lien contenu dans le message entraîne également la sauvegarde de l'état de l'activité. Un processus pour le navigateur est créé et l'activité du navigateur est lancée et affichée. Au cours de la navigation, l'utilisateur souhaite utiliser un lien qui déclenche l'application Google Maps. L'état du navigateur est sauvegardé, mais voilà, nous n'avons plus de place. Que se passe-t-il ?

Nous partons tout simplement à la chasse aux processus pour en tuer un et libérer des ressources. Nous ne pouvons pas détruire le processus `home` car il est nécessaire qu'il soit toujours utilisable. Nous ne choisissons pas non plus le navigateur : nous en venons tout juste et il y a de fortes chances que nous y retournions. Par contre, l'application de lecture des courriels semble un très bon candidat : elle est détruite avec son processus. Le processus Google Maps est créé ; l'activité est lancée et immédiatement disponible pour l'utilisateur.

> Nous éclaircirons la question des sauvegardes de l'état de l'application au chapitre 6 sur la persistance des données.

Imaginons que l'utilisateur veuille revenir en arrière grâce à la touche *Précédent*. La première chose qui va se passer est la disparition de l'activité Google Maps du haut de la pile des activités. Nous revenons au navigateur. L'activité correspondant au navigateur est toujours en mémoire, dans son processus. Il n'y a pas lieu de restaurer son état mais elle passe néanmoins en premier plan.

Et si l'utilisateur souhaite encore revenir en arrière ? Il s'attend à ce que son message soit affiché, mais les activités et les processus de l'application de courriels sont détruits ! Il faut donc faire de la place pour exécuter l'application de messagerie en tuant le processus de l'application Google Maps. Un nouveau processus pour l'application de messagerie est lancé et une nouvelle instance d'activité de lecture de courriers est lancée mais ne se trouve pas dans l'état laissé par l'utilisateur. Pour revenir à cet état, il faut relire les informations sauvegardées. L'activité de messagerie prend la place du navigateur en haut de la pile d'activités.

Pour revenir à la liste de courriels, pas besoin de créer un processus : le processus de messagerie existe déjà. Il suffit de créer l'activité et de restaurer son état.

Toute cette machinerie de processus et d'applications est heureusement invisible pour l'utilisateur. Pour développer des applications, il faut en revanche avoir conscience de toutes ces étapes dans la navigation utilisateur entre applications et de tout le travail réalisé par le système en termes de gestion de processus et de sauvegarde et restauration d'états.

Commande à retenir

Un outil très utile pour suivre les priorités des processus est `adb` (*Android Debug Bridge*) qui permet de voir l'état de tous les processus tournant sur l'émulateur. La commande est la suivante :

```
adb shell dumpsys activity
```

Cette commande vous fournira un ensemble d'informations sur les processus tournant comme leur `pid` (identifiant processus), leur priorité…
À noter que son pendant en interface graphique est disponible grâce au complément ADT sous Eclipse.

Qu'est-ce qu'une activité Android ?

Une activité peut être assimilée à un écran qu'une application propose à son utilisateur. Pour chaque écran de votre application, vous devrez donc créer une activité. La transition entre deux écrans correspond (comme vu un peu plus haut dans le cycle de vie d'une application) au lancement d'une activité ou au retour sur une activité placée en arrière-plan.

Une activité est composée de deux volets :

- la logique de l'activité et la gestion du cycle de vie de l'activité qui sont implémentés en Java dans une classe héritant de `Activity` (nous reviendrons plus tard sur tous ces concepts) ;
- l'interface utilisateur, qui pourra être définie soit dans le code de l'activité soit de façon plus générale dans un fichier XML placé dans les ressources de l'application.

Voici venu le moment de voir à quoi ressemble l'activité la plus simple possible.

Code 2-1 : Squelette minimal pour créer une première activité

```
import android.app.Activity;
import android.os.Bundle;

public class ActiviteSimple extends Activity {
   /**
    * Méthode appelée à la création de l'activité
    * @param savedInstanceState permet de restaurer l'état
    * de l'interface utilisateur
    */
```

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

Cycle de vie d'une activité

Tout ce que nous avons vu en parlant du cycle de vie d'une application, notamment sur la gestion des processus en fonction des ressources, a un impact direct sur les activités et notamment sur leur cycle de vie.

Les états principaux d'une activité sont les suivants :

- active (*active*) : activité visible qui détient le focus utilisateur et attend les entrées utilisateur. C'est l'appel à la méthode `onResume`, à la création ou à la reprise après pause qui permet à l'activité d'être dans cet état. Elle est ensuite mise en pause quand une autre activité devient active grâce à la méthode `onPause` ;
- suspendue (*paused*) : activité au moins en partie visible à l'écran mais qui ne détient pas le focus. La méthode `onPause` est invoquée pour entrer dans cet état et les méthodes `onResume` ou `onStop` permettent d'en sortir ;
- arrêtée (*stopped*) : activité non visible. C'est la méthode `onStop` qui conduit à cet état.

Voici un diagramme (voir figure 2-2) qui représente ces principaux états et les transitions y menant.

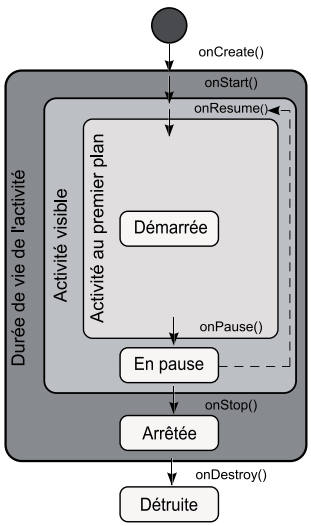


Figure 2–2
Cycle de vie d'une activité

Le cycle de vie d'une activité est parsemé d'appels aux méthodes relatives à chaque étape de sa vie. Il informe ainsi le développeur sur la suite des événements et le travail qu'il doit accomplir. Voyons de quoi il retourne en observant chacune de ces méthodes.

Code 2-2 : Squelette d'une activité

```
package com.eyrolles.android.activity;

import android.app.Activity;
import android.os.Bundle;

public final class TemplateActivity extends Activity {

   /**
    * Appelée lorsque l'activité est créée.
    * Permet de restaurer l'état de l'interface
    * utilisateur grâce au paramètre savedInstanceState.
    */
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // Placez votre code ici
   }

   /**
    * Appelée lorsque que l'activité a fini son cycle de vie.
    * C'est ici que nous placerons notre code de libération de
    * mémoire, fermeture de fichiers et autres opérations
    * de "nettoyage".
    */
   @Override
   public void onDestroy(){
      // Placez votre code ici
      super.onDestroy();
   }
   /**
    * Appelée lorsque l'activité démarre.
    * Permet d'initialiser les contrôles.
    */
   @Override
   public void onStart(){
      super.onStart();
      // Placezvotre code ici
   }

   /**
    * Appelée lorsque l'activité passe en arrière plan.
    * Libérez les écouteurs, arrêtez les threads, votre activité
    * peut disparaître de la mémoire.
    */
```

```
@Override
public void onStop(){
   // Placez votre code ici
   super.onStop();
}

/**
 * Appelée lorsque l'activité sort de son état de veille.
 */
@Override
public void onRestart(){
   super.onRestart();
   //Placez votre code ici
}

/**
 * Appelée lorsque que l'activité est suspendue.
 * Stoppez les actions qui consomment des ressources.
 * L'activité va passer en arrière-plan.
 */
@Override
public void onPause(){
   //Placez votre code ici
   super.onPause();
}

/**
 * Appelée après le démarrage ou une pause.
 * Relancez les opérations arrêtées (threads).
 * Mettez à jour votre application et vérifiez vos écouteurs.
 */
@Override
public void onResume(){
   super.onResume();
   // Placez votre code ici
}

/**
 * Appelée lorsque l' activité termine son cycle visible.
 * Sauvez les données importantes.
 */
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
   // Placez votre code ici
   // sans quoi l'activité aura perdu son état
   // lors de son réveil
   super.onSaveInstanceState(savedInstanceState);
}
```

```
    /**
     * Appelée après onCreate.
     * Les données sont rechargées et l'interface utilisateur.
     * est restaurée dans le bon état.
     */
    @Override
    public void onRestoreInstanceState(Bundle savedInstanceState) {
        super.onRestoreInstanceState(savedInstanceState);
        //Placez votre code ici
    }
}
```

Les vues

Les vues sont les briques de construction de l'interface graphique d'une activité Android. Les objets `View` représentent des éléments à l'écran qui permettent d'interagir avec l'utilisateur via un mécanisme d'événements.

Plus concrètement, chaque écran Android contient un arbre d'éléments de type `View` dont chaque élément est différent de par ses propriétés de forme, de taille...

Bien que la plupart des éléments dont nous ayons besoin – textes, boutons... – soient fournis par la plate-forme, il est tout à fait possible de créer des éléments personnalisés (ce sujet est abordé au chapitre 5 sur les interfaces utilisateur avancées).

Les vues peuvent être disposées dans une activité (objet `Activity`) et donc à l'écran soit par une description XML, soit par un morceau de code Java.

Tous ces aspects de création d'interfaces graphiques sont abordés en détails dans le chapitre suivant.

Les ressources

Le but est ici de présenter les différents types de ressources prises en charge et les concepts généraux concernant leur utilisation (syntaxe, format, appel…) dans les projets. La mise en pratique plus concrète de ces notions se fera tout au long des exemples développés sur chaque thème et les concepts plus avancés comme l'internationalisation sont développés plus en détails dans le chapitre traitant des interfaces utilisateur avancées.

> **À RETENIR Les ressources**
>
> Les ressources sont des fichiers externes – ne contenant pas d'instruction – qui sont utilisés par le code et liés à votre application au moment de sa construction. Android offre un support d'un grand nombre de fichiers ressources comme les fichiers images JPEG et PNG, les fichiers XML…

L'externalisation des ressources en permet une meilleure gestion ainsi qu'une maintenance plus aisée. Android étend ce concept à l'externalisation de la mise en page des interfaces graphiques (*layouts*) en passant par celle des chaînes de caractères, des images et bien d'autres...

Physiquement, les ressources de l'application sont créées ou déposées dans le répertoire `res` de votre projet. Ce répertoire sert de racine et contient lui-même une arborescence de dossiers correspondant à différents types de ressources.

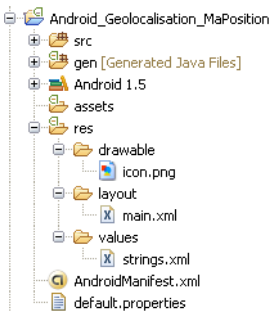
Tableau 2–4 Les types majeurs de ressources avec leur répertoire associé

Type de ressource	Répertoire associé	Description
Valeurs simples	`res/values`	Fichiers XML convertis en différents types de ressources. Ce répertoire contient des fichiers dont le nom reflète le type de ressources contenues : 1. `arrays.xml` définit des tableaux ; 2. `string.xml` définit des chaînes de caractères ; 3. ...
Drawables	`res/drawable`	Fichiers `.png`, `.jpeg` qui sont convertis en bitmap ou `.9.png` qui sont convertis en "9-patches" c'est-à-dire en images ajustables. Note : à la construction, ces images peuvent être optimisées automatiquement. Si vous projetez de lire une image bit à bit pour réaliser des opérations dessus, placez-la plutôt dans les ressources brutes.
Layouts	`res/layout`	Fichiers XML convertis en mises en page d'écrans (ou de parties d'écrans), que l'on appelle aussi gabarits.
Animations	`res/anim`	Fichiers XML convertis en objets animation.
Ressources XML	`res/xml`	Fichiers XML qui peuvent être lus et convertis à l'exécution par la méthode `resources.getXML`.
Ressources brutes	`res/raw`	Fichiers à ajouter directement à l'application compressée créée. Ils ne seront pas convertis.

Toutes ces ressources sont placées, converties ou non, dans un fichier de type `APK` qui constituera le programme distribuable de votre application. De plus, Android crée une classe nommée `R` qui sera utilisée pour se référer aux ressources dans le code.

La structure des projets a évolué avec les versions des kits de développement. Voici une capture d'écran d'un projet typique Android en version 1.5 avec quelques-unes des ressources évoquées (voir figure 2-3).

Figure 2–3
Structure d'un projet Android



Les versions 2.x apportent des modifications, notamment pour prendre en compte les différentes résolutions d'écrans. Nous y reviendrons, notamment lors de la présentation des ressources de type image.

Utilisation des ressources

Les ressources que vous créez ou qu'Android propose peuvent être utilisées directement dans votre application ou être référencées dans d'autres ressources.

Ressources appelées dans votre code

Les ressources sont accessibles et utilisées depuis le code grâce à la classe statique `R`. Cette classe est automatiquement générée en fonction des ressources présentes dans votre projet au moment de la compilation et de la construction de l'application.

Code 2-3 : Exemple de classe R générée automatiquement

```
package com.eyrolles.android.exemples;
public final class R {
    public static final class string { (1)
        public static final int bienvenue=0x7f040000; (2)
        public static final int texte_bouton_quitter=0x7f040001;
        public static final int texte_titre_ecran=0x7f040002;
    };
    public static final class layout {
        public static final int ecran_de_demarrage=0x7f030001;
        public static final int ecran_principal=0x7f030000;
    };
    public static final class drawable {
        public static final int image_android=0x7f020000;
    };
};
```

La classe `R` contient donc des classes en membres statiques pour chaque type de ressources défini dans le projet (1). Vous pouvez accéder à des ressources via les sous-classes `R.string` pour une chaîne de caractères, mais aussi `R.color` pour les couleurs… Chaque sous-classe expose donc au moins une variable dont le nom correspond à l'identifiant de la ressource (2).

Pour utiliser une ressource, il suffit donc de connaître son type et son identifiant. La syntaxe est alors la suivante :

```
android.R.type_de_ressource.nom_ressource
```

Voici deux exemples pour préciser l'utilisation des ressources. La première utilisation est correcte :

```
// Fixe la mise en page d'une activité
setContentView(R.layout.ecran_de_demarrage);
```

La seconde ne l'est pas : on utilise l'identifiant d'une ressource alors qu'on attend une chaîne de caractères en paramètre pour cette méthode :

```
// Création par copie d'une chaîne de caractères
String titre = new String(R.string.texte_titre_ecran);
```

Si comme dans le second exemple vous voulez manipuler des instances, il faut les extraire de la table grâce aux méthodes de la classe `Ressources` dont l'instance s'obtient ainsi :

```
Resources resources = getResources();
```

Avec cette instance, on peut utiliser les méthodes telles que `getString`, `getColor` et bien d'autres qui, avec un identifiant donné, renvoient une instance du type demandé avec la valeur extraite de la table.

Par exemple, pour récupérer le nom de l'application défini par une ressource `app_name`, on utilisera la ligne de code suivante :

```
String nom = resources.getString(R.string.app_name);
```

EN COULISSES À propos de la création de la classe R

Si vous utilisez le module ADT sous Eclipse, la classe R sera régénérée à chaque changement de votre projet. En tout cas, cette classe ne doit pas être modifiée manuellement. Si besoin est, utilisez l'outil AAPT (*Android Asset Packaging Tool*) qui se trouve dans le répertoire `tools` du SDK pour compiler votre projet et générer votre classe `R`.

Ressources référencées par d'autres ressources

Vous pouvez également utiliser vos ressources comme valeurs d'attributs dans d'autres ressources sous forme XML. Cette possibilité est très utilisée dans les mises en page par exemple.

La notation pour faire référence à une autre ressource est la suivante :

```
attribute="@[package_name:]resource_type/resource_identifier"
```

En voici l'utilisation au sein d'un exemple qui crée une mise en page sous forme de table (d'une ligne et de deux colonnes) dont les cellules sont remplies par des chaînes de caractères (1) et (2) :

Code 2-4 : Exemple d'une disposition d'écran

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:stretchColumns="1">
    <TableRow>
        <TextView
            android:text="@string/table_contenu_cellule_gauche" /> (1)
        <TextView
            android:text="@string/table_contenu_cellule_droite" /> (2)
    </TableRow>
</TableLayout>
```

Utilisation de ressources système

Les applications natives Android externalisent elles aussi leurs ressources. Dès lors, il est possible d'utiliser leurs images, leurs chaînes de caractères...

Le mode opératoire pour accéder à ces ressources est le même que pour vos ressources personnelles.

Pour une utilisation au sein du code, il suffit d'utiliser classe `android.R` :

```
android.R.drawable.ic_dialog_alert
```

Pour accéder à ces ressources dans un fichier XML, il faut spécifier `android` comme espace de nommage.

Code 2-5 : Exemple d'utilisation des ressources d'Android

```
<?xml version="1.0" encoding="utf-8"?>
<EditText id="text"
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:text="@android:string/unknownName" />
```

Créer des ressources

Valeurs simples

Il est possible de déclarer des chaînes de caractères, des dimensions, des couleurs, des tableaux (de chaînes de caractères ou d'entiers) comme valeurs simples.

Elles sont déclarées dans des fichiers XML. Ces fichiers XML peuvent contenir tous les types à la fois, mais par convention on sépare les ressources en regroupant les mêmes types dans un fichier au nom explicite comme `res/values/strings.xml` pour regrouper toutes les chaînes de caractères.

Voici un exemple de fichier (dans vos projets, veillez à bien séparer chaque type de ressources).

Code 2-6 : Définition d'un fichier de ressources contenant plusieurs types

```
<?xml version="1.0" encoding="utf-8"?> (1)
<resources> (2)
   <string name="nom_application">Suivi des SMS</string>
   <color name="couleur_des_boutons">#FFAABBCC</color>
   <array name="tableau_entiers">
      <item>1765</item>
      <item>3</item>
   </array>
</resources>
```

Ne pas oublier la déclaration XML (1) dans vos fichiers et veillez impérativement à mettre les balises qui décrivent les ressources à l'intérieur de l'élément racine `<resources>` (2).

Couleurs

Une couleur définit une valeur RVB (rouge, vert et bleu) et une transparence. Cette couleur peut être utilisée à de multiples endroits comme pour définir la couleur d'un

texte. Il existe différents formats dont la syntaxe globale est la suivante : `<color name=nom_couleur>valeur_de_la_couleur</color>`

Les différents formats d'une couleur sont les suivants :

- #RGB (Rouge-Vert-Bleu/valeur hexadécimale sur 1 caractère [0,16])
- #ARGB (Alpha (transparence)-Rouge-Vert-Bleu)
- #RRGGBB (Rouge-Vert-Bleu/valeur hexadécimale sur 2 caractères [0,255])
- #AARRGGBB

Exemple de déclaration de chaîne de caractères au format AARRGGBB (valeur pour la transparence puis pour les composantes de rouge, vert et bleu) :

Code 2-7 : Définition d'un fichier de ressources de couleurs

```
<resources>
  <color name="bleu_transparent">#50ff00FF</color>
</resources>
```

Utilisation des couleurs (exemple avec la déclaration de la couleur `bleu_transparent`) :

- Java : `R.color.bleu_transparent`
- XML : `@[package:]color/bleu_transparent`

Chaînes de caractères et texte formaté

Les chaînes de caractères avec un format optionnel peuvent être utilisées comme ressources.

Leur syntaxe est la suivante : `<string name=nom_chaine>valeur_de_la_chaine</string>`.

Ce format assez simple vous permet d'utiliser trois balises HTML standard `<b>` (**gras**), `<i>` (*italique*) et `<u>` (souligné) :

```
<string name="ex1">Un texte <b>mis en forme</b></string>
```

À noter que si vous voulez utiliser des guillemets ou des apostrophes, vous devez les échapper en les faisant précéder du caractère slash ('\') :

```
<string name="ex2">Voici l\'application de ce conseil</string>
```

Quelques exemples de déclaration de chaînes de caractères :

Code 2-8 : Définition d'un fichier de ressources de chaînes de caractères

```
<resources>
   <string name="app_name">Exemple Android Eyrolles</string>
   <string name="menu_principal">Menu Principal</string>
</resources>
```

Utilisation des chaînes de caractères :

- Java : `R.string.le_nom`
- XML : `@[package:]string/un_nom`

Unités de mesure (« dimensions »)

Les dimensions sont la plupart du temps référencées dans les styles et les mises en page. Elles peuvent servir de constantes entre différents éléments d'écrans. Voici les unités prises en charge par Android : px (pixels), in (pouces), mm (millimètres), pt (points), dp (« density-independant » pixel), sp (« scale-independant pixel »). Vous trouverez plus de détail sur ces unités dans le chapitre 4 sur les interfaces utilisateur.

Voici quelques exemples :

Code 2-9 : Définition d'un fichier de ressources de dimensions

```
<resources>
    <dimen name="taille_texte">5sp</dimen>
</resources>
```

Utilisation des dimensions :

- Java : `R.dimen.un_nom`

```
Resources.getDimen(R.dimen.taille_texte);
```

- XML : `@[package:]dimen/un_nom`

```
<TextView android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textSize="@dimen/taille_texte"/>
```

Images

Fichiers bitmap

Android prend en charge les fichiers bitmap sous différents formats avec une préférence bien affirmée dans la documentation pour le format PNG. Il est cependant possible d'utiliser le format JPEG ou le format GIF (qui est déconseillé par la documentation).

Utilisation des images de type bitmap :

- Java : `R.drawable.fichier_bitmap`
- XML : `@[package:]drawable/fichier_bitmap`

PRÉCISIONS Les fichiers dits bitmaps

Les fichiers dits bitmaps sont des formats d'images qui ont en commun une organisation des données qui code l'image sous forme de champs de bits. Les formats dits vectoriels sont un exemple d'autres possibilités de représentation et de stockage des données.
Les formats JPEG, GIF ou PNG sont plus rarement classés dans cette catégorie puisqu'ils utilisent des méthodes de compression. C'est le choix fait dans la documentation Android, nous le suivons !
Le format éponyme BMP (pour **B**it**M**a**P**), que l'on doit à Microsoft, n'en est qu'un cas particulier.

À partir d'Android 1.6, le kit de développement propose des possibilités étendues pour la gestion des résolutions d'écran hétérogènes dues à des appareils Android équipés de matériels aux tailles et caractéristiques différentes. Pour ce faire, la plate-forme classe les tailles d'écran et leurs résolutions en trois catégories.

Concernant les tailles d'écran, le fichier de configuration dispose d'une balise `<supports-screens>` qui grâce aux attributs `android:smallScreens`, `android:normalScreens` et `android:largeScreens` permet de spécifier quelle(s) taille(s) d'écran votre application supporte.

Concernant les résolutions, chacune de ces catégories (*low*, *medium*, *high*) dispose d'un répertoire où enregistrer vos ressources.

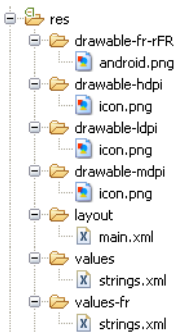


Figure 2–4
Exemple de projet avec des ressources spécialisées

Le guide de développement Android propose un dossier complet sur le développement d'applications compatibles avec des écrans différents :
- http://developer.android.com/guide/practices/screens_support.html

Vous aurez également certainement remarqué que le premier répertoire de la capture d'écran est composé d'un format de langue. On peut en effet également personnaliser les ressources graphiques en proposant des images différentes en fonction de la langue choisie sur l'appareil. Pour plus d'informations, l'internationalisation des applications est traitée au chapitre 5 sur la création d'interfaces utilisateur avancées.

Images étirables

Les images étirables, dites *Nine-Patch*, sont des images PNG dans lesquelles on peut définir des portions étirables que le système redimensionnera pour obtenir la taille souhaitée. Leur utilisation :

- Java : `R.drawable.fichier`
- XML : `@[package:]drawable.fichier`

Elles permettent de réaliser des interfaces graphiques très travaillées.

Animations

La déclaration d'animations est plus compliquée, mais le résultat n'est pas décevant avec des possibilités de rotation, de fondu, de translation et de changements de taille.

Les animations sont placées dans un ou plusieurs fichiers dans le répertoire `res/anim`. Voici les quatre types d'animations proposées :

- `<alpha>` : permet de faire un fondu ;
- `<scale>` : définit un facteur d'échelle de départ et d'arrivée en X et Y ;
- `<translate>` : permet de déplacer l'élément ;
- `<rotate>` : propose une rotation avec un point pivot et un angle en degrés.

Android propose également des attributs permettant de nombreux réglages comme la durée, la répétition, etc.

Voici un exemple d'animation de type fondu avec la transparence qui passe de 0 à 1 :

Code 2-10 : Exemple d'une ressource d'animation

```
<?xml version="1.0" encoding="utf-8"?>
...
<alpha xmlns:android="http://schemas.android.com/apk/res/android"
   android:interpolator="@android:anim/accelerate_interpolator"
   android:fromAlpha="0.0"
   android:toAlpha="1.0"
   android:duration="100" />
...
```

Les animations sont traitées en détail dans le chapitre sur les interfaces graphiques avancées.

Autres ressources

Toutes les ressources n'ont pas été détaillées dans ce chapitre. Voici celles qui seront développées à l'occasion des chapitres de ce livre :

- les menus : les menus de l'application peuvent être définis comme ressources XML. Ce point sera abordé dans le chapitre portant sur la création d'interfaces graphiques ;

- la mise en page (*layouts* ou encore gabarits) : les mises en pages définies dans les fichiers de ressources XML permettent de découper la couche présentation en évitant de construire la mise en page d'une interface graphique dans le code. Ce point sera également abordé dans le chapitre qui vous guidera lors de la création d'interfaces graphique.

Le fichier de configuration Android : la recette de votre application

Chaque application Android nécessite un fichier de configuration : `AndroidManifest.xml`. Ce fichier est placé dans le répertoire de base du projet, à sa racine. Il décrit le contexte de l'application, les activités, les services, les récepteurs d'Intents (*Broadcast receivers)*, les fournisseurs de contenu et les permissions.

Structure du fichier de configuration

Un fichier de configuration est composé d'une racine (le tag `manifest` 1) et d'une suite de nœuds enfants qui définissent l'application.

Code 2-11 : Structure vide d'un fichier de configuration d'une application

```
<manifest 1
     xmlns:android=http://schemas.android.com/apk/res/android 2
     package="fr.domaine.application"> 3

</manifest>
```

La racine XML de la configuration est déclarée avec un espace de nom Android (`xmlns:android` 2) qui sera utile plus loin dans le fichier ainsi qu'un paquetage 3 dont la valeur est celle du paquetage du projet.

B-A-BA Les espaces de nommage (namespaces)

Le langage XML étant un métalangage extensible, chacun peut utiliser les balises qu'il souhaite pour structurer ses données et éventuellement les faire valider par une DTD ou un schéma pour en vérifier la structure et en partie la pertinence.
Comment, dès lors, distinguer deux balises utilisées dans des contextes différents ? Et bien en utilisant un espace de nommage qui associe une URI à un ensemble de balises – pour résumer. Ainsi une activité associée à l'espace de nommage Android peut être identifiée sans ambiguïté par rapport à une activité qui représente un emploi du temps dans une application connexe.

Voici le rendu possible d'un manifeste qui donne une bonne idée de sa structure. Ce fichier est au format XML. Il doit donc toujours être :

- bien formé : c'est-à-dire respecter les règles d'édition d'un fichier XML en termes de nom des balises, de balises ouvrante et fermante, de non-imbrication des balises, etc. ;
- valide : il doit utiliser les éléments prévus par le système avec les valeurs prédéfinies.

Profitons-en pour étudier les éléments les plus importants de ce fichier de configuration.

Code 2-12 : Structure du fichier AndroidManifest.xml extraite de la documentation

```
<?xml version="1.0" encoding="utf-8"?>

<manifest>

    <uses-permission /> 1
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />

    <application> 2

        <activity> 3
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>

        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>

        <service> 4
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>

        <receiver> 5
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>
```

```
        <provider> 6
            <grant-uri-permission />
            <path-permission />
            <meta-data />
        </provider>

        <uses-library />

    </application>

</manifest>
```

`<uses-permission>` 1

Les permissions qui seront déclarées ici seront un prérequis pour l'application. À l'installation, l'utilisateur se verra demander l'autorisation d'utiliser l'ensemble des fonctions liées à ces permissions comme la connexion réseau, la localisation de l'appareil, les droits d'écriture sur la carte mémoire...

`<application>` 2

Un manifeste contient un seul et unique nœud application qui en revanche contient des nœuds concernant la définition d'activités, de services…

`<activity>` 3

Déclare une activité présentée à l'utilisateur. Comme pour la plupart des déclarations que nous allons étudier, si vous oubliez ces lignes de configuration, vos éléments ne pourront pas être utilisés.

`<service>` 4

Déclare un composant de l'application en tant que service. Ici pas question d'interface graphique, tout se déroulera en tâche de fond de votre application.

`<receiver>` 5

Déclare un récepteur d'objets `Intent`. Cet élément permet à l'application de recevoir ces objets alors qu'ils sont diffusés par d'autres applications ou par le système.

`<provider>` 6

Déclare un fournisseur de contenu qui permettra d'accéder aux données gérées par l'application.

Manipulation avec l'éditeur du module ADT pour Eclipse

Le complément ADT pour Eclipse vous propose un éditeur pour gérer votre fichier de configuration de façon visuelle plutôt que de manipuler du XML.

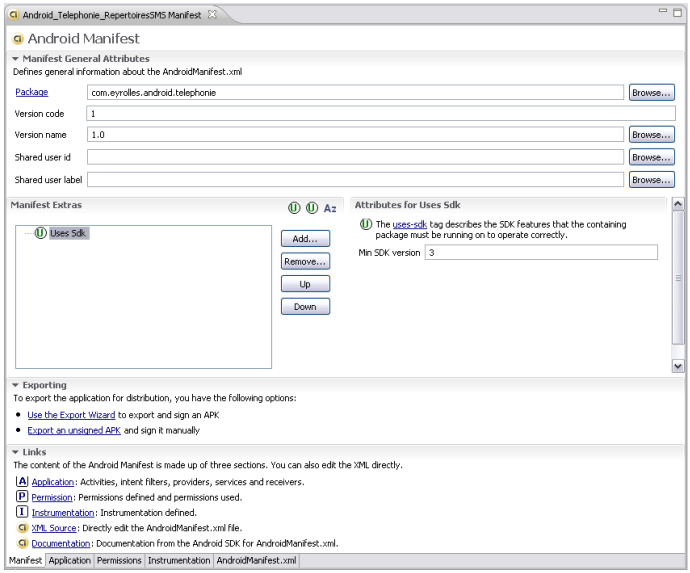


Figure 2–5
Éditeur de fichier de configurtion du plug-in Eclipse

Pour faire apparaître cet assistant, deux solutions s'offrent à vous. La première consiste à double-cliquer sur le fichier de configuration (`AndroidManifest.xml`) dans la vue projet d'Eclipse. La seconde demande de réaliser un clic droit sur le fichier de configuration et de sélectionner *Open With>Android Manifest Editor*.

Cet éditeur vous donne directement accès aux caractéristiques de votre application, aux permissions et plus généralement à tout le fichier de configuration de l'application grâce aux onglets situés en bas de la vue.

Personnaliser notre première application Android

Après ces quelques lignes de théorie, construisons une application et prenons un soin tout particulier à réaliser une première activité. Plus l'application sera complexe, plus elle nécessitera d'écrans et donc d'activités.

Une activité peut occuper tout l'écran ou apparaître sous forme de fenêtre semi-transparente. Une application typique pourra regrouper plusieurs activités telles qu'une activité principale et une secondaire (classe `Activity`), une zone de liste de données (`ListActivity`), une boîte de dialogue (`AlertDialog`) et pourquoi pas une page de paramètres (`PreferenceActivity`).

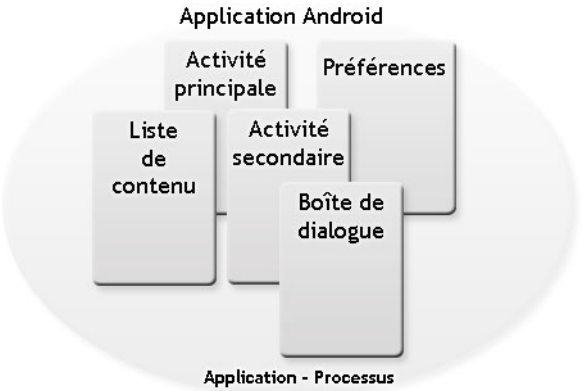


Figure 2–6
Représentation schématique des activités

Vous pouvez également créer vos propres activités en héritant d'une classe de base. Mais nous n'en sommes pas encore là ! Tout d'abord, commençons par la base : notre programme affichera un écran noir avec un texte à l'intérieur.

Créez un projet comme nous l'avons vu au chapitre précédent, soit grâce aux icônes de raccourci, soit par le menu.

Une fois construit, le projet contient déjà l'activité principale qui sera exécutée au lancement de l'application. Au nom de l'application et à celui du paquetage près, l'activité aura la structure suivante.

Code 2-13 : Activité principale créée par défaut

```
import android.app.Activity;
import android.os.Bundle;

public class ActivitePrincipale extends Activity (1) {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

À retenir une nouvelle fois : une activité hérite de l'objet `Activity` (1) pour implémenter ses propres fonctionnalités.

Une fois l'activité lancée, le résultat est le même que celui obtenu au chapitre précédent. Entrons dans le détail pour faire évoluer ce code.

Pour changer ce texte nous allons jeter un œil du côté des gabarits de disposition des contrôles. Comme nous l'avons vu plus haut, ces gabarits permettent de définir une interface utilisateur à l'aide d'un fichier XML contenu dans le dossier *res/layout* de notre projet.

L'objet `LayoutInflater` se charge de transformer la représentation XML des contrôles en objet. Voyons de plus près la composition de notre fichier XML qui a été générée automatiquement.

Code 2-14 : Aperçu du fichier main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    > (1)
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
    /> (2)
</LinearLayout>
```

Ce fichier se compose d'un `LinearLayout` (1) contenant un `TextView` (2). En d'autres termes nous avons un conteneur et un label. Le chapitre 5 vous montrera en détail comment ajouter des contrôles et tirer profit des nombreuses propriétés des gabarits.

Pour reprendre notre code de création de l'activité, nous aurions tout aussi bien pu nous passer du fichier XML. Voici un équivalent en code.

Code 2-15 : Créer une activité sans main.xml

```
package com.eyrolles.android.activity;

import android.app.Activity;
import android.os.Bundle;
import android.widget.LinearLayout;
import android.widget.TextView;

public class MainActivity extends Activity {

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        //setContentView(R.layout.main);

        LinearLayout layout = new LinearLayout(this); (1)
        TextView text = new TextView(this); (2)
        text.setText(R.string.hello);
        layout.addView(text); (3)
        setContentView(layout);
    }
}
```

On retrouve bien les deux contrôles `LinearLayout` (1) et `TextView` (2) de notre fichier XML ainsi que le texte correspondant à l'identifiant `hello` (3) puisé dans les ressources.

L'avantage de l'utilisation du fichier XML réside dans le fait qu'il est premièrement plus facile à lire et à créer notamment grâce à l'assistant intégré, et deuxièmement parce qu'il permet de créer des interfaces multilingues qui seront abordées dans le chapitre 5 traitant des interfaces graphiques avancées.

> **À RETENIR L'assistant d'édition des gabarits**
>
> L'usage de l'assistant n'est pas toujours chose aisée. En effet celui-ci rencontre souvent des difficultés à gérer des mises en pages complexes. Il est donc primordial de vous familiariser avec la représentation XML des contrôles. Vous pouvez vous reporter à l'annexe de ce livre pour en apprendre plus sur l'assistant de conception visuel.

Reste à personnaliser ce fameux texte !

Ouvrez le fichier `strings.xml` qui se trouve dans le répertoire `res/values`. Il contient les chaînes de caractères utilisées comme ressources dans l'application.

Code 2-16 : Ressources textuelles utilisées dans l'application

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, Test!</string> (1)
    <string name="app_name">Mon Application</string>
</resources>
```

La chaîne de caractères à modifier porte l'identifiant `hello` (1). Choisissez votre message et relancez l'application dans l'émulateur.

Ceci fait, il reste un aspect que nous n'avons pas abordé : le paramétrage de l'application qui a été fait automatiquement. Nous avions vu que chaque activité de notre application devait être déclarée dans un fichier de configuration à la racine du projet nommé `AndroidManifest.xml`. Voyons à quoi ressemble le nôtre.

Code 2-17 : Déclarer une activité dans AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.eyrolles.android.activity"
    android:versionCode="1"
    android:versionName="1.0">
    <application
    android:icon="@drawable/icon"
    android:label="@string/app_name">
       <activity
            android:name=".ActivitePrincipale"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="4" />
</manifest>
```

On retrouve les ressources associées à l'activité : l'icône et le nom de l'application. Ces deux propriétés sont configurables en manipulant les ressources du projet.

La balise `activity` contient bien le nom de l'activité qui a été créée ainsi qu'une balise enfant concernant les objets `Intent` à filtrer. Elle permet de déterminer l'activité principale à lancer au démarrage : nous y reviendrons dans le chapitre 4 consacré au mécanisme des objets `Intent`.

En résumé

Le socle de développement Android est posé. Les différentes pièces du puzzle vous sont maintenant familières. C'est le moment d'aborder les différents thèmes que nous vous proposons.

Le passage par les chapitres concernant les interfaces graphiques et les objets `Intent` est quasi indispensable, aussi ne les ignorez pas. Ensuite vous serez à même de pouvoir utiliser les autres thèmes que nous vous proposons en fonction des besoins de votre application : réseau, multimédia... la réponse à vos problématiques arrive !

3

Création d'interfaces utilisateur

Il n'y a pas de bonne application sans bonne interface utilisateur : c'est la condition de son succès auprès des utilisateurs.

Les interfaces graphiques prennent une place de plus en plus importante dans le choix des applications par les utilisateurs, tant dans l'implémentation de concepts innovants qu'au niveau de l'ergonomie.

Comme sur bien des plates-formes, les interfaces d'applications Android sont organisées en vues et gabarits, avec néanmoins quelques spécificités.

Le concept d'interface

Une interface n'est pas une image statique mais un ensemble de composants graphiques, qui peuvent être des boutons, du texte, mais aussi des groupements d'autres composants graphiques, pour lesquels nous pouvons définir des attributs communs (taille, couleur, positionnement, etc.). Ainsi, l'écran ci-après (figure 3-1) peut-il être vu par le développeur comme un assemblage (figure 3-2).

La représentation effective par le développeur de l'ensemble des composants graphiques se fait sous forme d'arbre, en une structure hiérarchique (figure 3-3). Il peut n'y avoir qu'un composant, comme des milliers, selon l'interface que vous souhaitez représenter. Dans l'arbre ci-après (figure 3-3), par exemple, les composants ont été organisés en 3 parties (*haut*, *milieu* et *bas*).

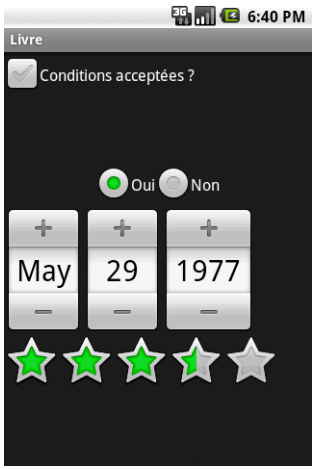


Figure 3–1
Exemple d'un écran

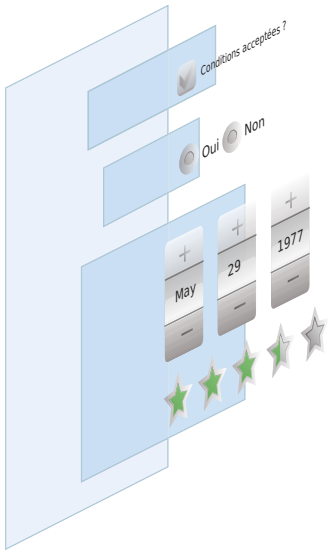


Figure 3–2
Le même écran vu par le développeur

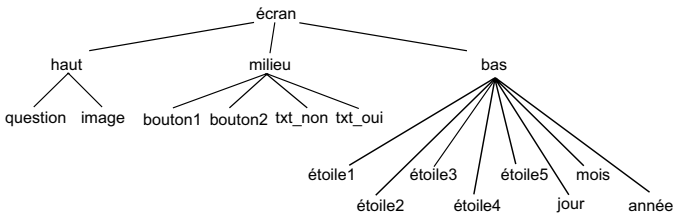


Figure 3–3
Arborescence de l'interface précédente

Cet exemple donne l'idée générale de l'organisation des interfaces – car les données graphiques ne sont pas exactement représentées ainsi. Nous verrons plus loin comment cela est géré pour Android.

Sous Android, vous pouvez décrire vos interfaces utilisateur de deux façons différentes (nous reviendrons plus tard en détail sur ce point) : avec une description déclarative XML ou directement dans le code d'une activité en utilisant les classes adéquates. La façon la plus simple de réaliser une interface est d'utiliser la méthode déclarative XML via la création d'un fichier XML que vous placerez dans le dossier `/res/layout` de votre projet.

Les vues

Le composant graphique élémentaire de la plate-forme Android est la *vue* : tous les composants graphiques (boutons, images, cases à cocher, etc.) d'Android héritent de la classe `View`. Ainsi, dans la suite ce livre, gardez bien à l'esprit que lorsque nous créons une vue, nous créons en fait un objet graphique.

Tout comme nous l'avions fait dans l'exemple précédent, Android vous offre la possibilité de regrouper plusieurs vues dans une structure arborescente à l'aide de la classe `ViewGroup`. Cette structure peut à son tour regrouper d'autres éléments de la classe `ViewGroup` et être ainsi constituée de plusieurs niveaux d'arborescence.

L'utilisation et le positionnement des vues dans une activité se fera la plupart du temps en utilisant une mise en page qui sera composée par un ou plusieurs gabarits de vues.

Positionner les vues avec les gabarits

Un gabarit, ou *layout* dans la documentation officielle, ou encore mise en page, est une extension de la classe `ViewGroup`. Il s'agit en fait d'un conteneur qui aide à positionner les objets, qu'il s'agisse de vues ou d'autres gabarits au sein de votre interface.

Vous pouvez imbriquer des gabarits les uns dans les autres, ce qui vous permettra de créer des mises en forme évoluées. Dans la suite de ce livre, nous parlerons ainsi de gabarit parent et de gabarit enfant (ou plus généralement d'éléments enfants voire simplement d'enfants), le gabarit enfant étant inclus dans le gabarit parent.

Comme nous l'avons dit plus haut, vous pouvez décrire vos interfaces utilisateur soit par une déclaration XML, soit directement dans le code d'une activité en utilisant les classes adéquates. Dans les deux cas, vous pouvez utiliser différents types de gabarits. En fonction du type choisi, les vues et les gabarits seront disposés différemment :

- `LinearLayout` : permet d'aligner de gauche à droite ou de haut en bas les éléments qui y seront incorporés. En modifiant la propriété `orientation` vous pourrez signaler au gabarit dans quel sens afficher ses enfants : avec la valeur `horizontal`, l'affichage sera de gauche à droite alors que la valeur `vertical` affichera de haut en bas ;
- `RelativeLayout` : ses enfants sont positionnés les uns par rapport aux autres, le premier enfant servant de référence aux autres ;
- `FrameLayout` : c'est le plus basique des gabarits. Chaque enfant est positionné dans le coin en haut à gauche de l'écran et affiché par-dessus les enfants précédents, les cachant en partie ou complètement. Ce gabarit est principalement utilisé pour l'affichage d'un élément (par exemple, un cadre dans lequel on veut charger des images) ;
- `TableLayout` : permet de positionner vos vues en lignes et colonnes à l'instar d'un tableau.

Voici un exemple de définition déclarative en XML d'une interface contenant un gabarit linéaire (le gabarit le plus commun dans les interfaces Android).

Code 3-1 : Interface avec un gabarit LinearLayout

```
<!-- Mon premier gabarit -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
</LinearLayout>
```

Chaque gabarit possède des attributs spécifiques, et d'autres communs à tous les types de gabarits. Parmi les propriétés communes, vous trouverez les propriétés `layout_weight` et `layout_height`. Celles-ci permettent de spécifier le comportement du remplissage en largeur et en hauteur des gabarits et peuvent contenir une taille (en pixels ou dpi) ou les valeurs constantes suivantes : `fill_parent` et `wrap_content`.

La valeur `fill_parent` spécifie que le gabarit doit prendre toute la place disponible sur la largeur/hauteur. Par exemple, si le gabarit est inclus dans un autre gabarit – parent (l'écran étant lui-même un gabarit) – et si le gabarit parent possède une largeur de 100 pixels, le gabarit enfant aura donc une largeur de 100 pixels. Si ce gabarit est le gabarit de base – le plus haut parent – de notre écran, alors ce dernier prend toute la largeur de l'écran.

Si vous souhaitez afficher le gabarit tel quel, vous pouvez spécifier la valeur `wrap_content`. Dans ce cas, le gabarit ne prendra que la place qui lui est nécessaire en largeur/hauteur.

À RETENIR Les unités de mesure

Pour spécifier les dimensions des propriétés de taille de vos composants graphiques (largeur, hauteur, marges, etc), vous pouvez spécifier plusieurs types d'unité. Le choix de l'utilisation d'une unité par rapport à une autre se fera en fonction de vos habitudes et si vous souhaitez spécifier une taille spécifique ou relative, par exemple : 10 px, 5 mm, 20 dp, 10 sp.

Voici les unités prises en charge par Android :

- pixel (px) : correspond à un pixel de l'écran ;
- pouce (in) : unité de longueur, correspondant à 2,54 cm. Basé sur la taille physique de l'écran ;
- millimètre (mm) : basé sur la taille physique de l'écran ;
- point (pt) : 1/72 d'un pouce ;
- pixel à densité indépendante (dp ou dip) : une unité relative se basant sur une taille physique de l'écran de 160 dpi. Avec cette unité, 1 dp est égal à 1 pixel sur un écran de 160 pixels. Si la taille de l'écran est différente de 160 pixels, les vues s'adapteront selon le ratio entre la taille en pixels de l'écran de l'utilisateur et la référence des 160 pixels ;
- pixel à taille indépendante (sp) : fonctionne de la même manière que les pixels à densité indépendante à l'exception qu'ils sont aussi fonction de la taille de polices spécifiée par l'utilisateur. Il est recommandé d'utiliser cette unité lorsque vous spécifiez les tailles des polices.

Ces deux dernières unités sont à privilégier car elles permettent de s'adapter plus aisément à différentes tailles d'écran et rendent ainsi vos applications plus portables. Notez que ces unités sont basées sur la taille physique de l'écran : l'écran ne pouvant afficher une longueur plus petite que le pixel, ces unités seront toujours rapportées aux pixels lors de l'affichage (1 cm peut ne pas faire 1 cm sur l'écran selon la définition de ce dernier).

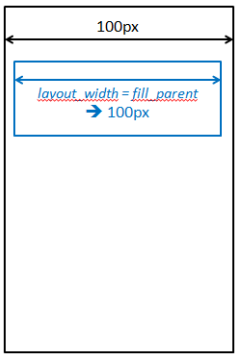


Figure 3–4
La largeur d'un composant graphique dont la propriété *layout_width* est égale à *fill_parent* correspond à la largeur de son gabarit parent.

Vous pouvez spécifier une taille précise en px (pixel) ou dip (dpi). Notez cependant que si vous avez besoin de spécifier une taille, notamment si vous avez besoin d'intégrer une image avec une taille précise, préférez les valeurs en *dip* à celles en *px*. En effet, depuis la version 1.6, Android est devenu capable de s'exécuter sur plusieurs tailles d'écrans. Les valeurs en dip permettent un ajustement automatique de vos éléments alors que les valeurs en px prennent le même nombre de pixels quelle que soit

la taille de l'écran, ce qui peut très vite compliquer la gestion de l'affichage sur la multitude d'écrans disponibles.

Créer une interface utilisateur

La création d'une interface se traduit par la création de deux éléments :

- une définition de l'interface utilisateur (gabarits, etc.) de façon déclarative dans un fichier XML ;
- une définition de la logique utilisateur (comportement de l'interface) dans une classe d'activité.

Cela permet d'avoir une séparation stricte entre la présentation et la logique fonctionnelle de votre application. De plus, un intégrateur graphique pourra modifier l'interface sans interférer avec le code du développeur.

> **À RETENIR Interface et activité**
>
> À une interface utilisateur est associée une activité ; à une activité est associée une interface utilisateur.

Définir votre interface en XML

Une bonne pratique est de définir intégralement votre interface dans la déclaration XML. Retenez néanmoins qu'il est aussi possible (et bon nombre de scénarios ne pourront s'en passer) d'instancier dynamiquement des vues depuis votre code.

Les fichiers de définition d'interface XML sont enregistrés dans le dossier `/layout` de votre projet. Prenez garde à ce que le nom du fichier ne comporte que des lettres minuscules et des chiffres (pas de lettre majuscule ou de caractère spécial, comme l'impose la convention de nommage Java).

Chaque fichier de définition d'interface, pour peu qu'il se trouve bien dans le répertoire `res/layout` de votre projet, possède un identifiant unique généré automatiquement par l'environnement de développement. De cette façon, vous pouvez y faire référence directement dans votre code. Par exemple, si le fichier se nomme `monLayout`, vous pouvez y faire référence dans votre code grâce à la constante `R.layout.monLayout`. Comme vous le verrez, l'utilisation des identifiants est très courante : ils vous serviront à récupérer des éléments, à spécifier des propriétés et à utiliser les nombreuses méthodes des activités (`findViewById`, `setContentView`, etc.).

Dans votre projet Android, créez un fichier `main.xml` dans le dossier `/layout` et placez-y le contenu suivant.

Code 3-2 : Définition d'interface

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:id="@+id/monText"
    />
</LinearLayout>
```

Cette interface définit un gabarit de type `LinearLayout` et un composant graphique de saisie de texte de type `TextView`. Vous remarquerez que nous avons défini un attribut `android:id` avec la valeur `@+id/monText`. Cette valeur nous permettra de faire référence à cet élément dans le code avec la constante `R.id.montext`. Pour le moment nous n'allons pas détailler les éléments et les attributs : nous y reviendrons plus loin dans ce chapitre.

Le complément ADT génère automatiquement un identifiant pour cette définition d'interface. Nommée `main.xml` et placée dans le répertoire `res/layout`, cet identifiant sera `R.layout.main`.

Associer votre interface à une activité et définir la logique utilisateur

Dans une application, une interface est affichée par l'intermédiaire d'une activité. Vous devez donc avant toute chose créer une activité en ajoutant une nouvelle classe à votre projet dérivant de la classe `Activity`.

Le chargement du contenu de l'interface s'effectue à l'instanciation de l'activité. Redéfinissez la méthode `onCreate` de l'activité pour y spécifier la définition de l'interface à afficher via la méthode `setContentView`. Cette méthode prend en paramètre un identifiant qui spécifie quelle ressource de type interface doit être chargée et affichée comme contenu graphique. Nous utilisons l'identifiant généré automatiquement pour spécifier l'interface que nous avons précédemment créée.

Code 3-3 : Spécifier une vue à l'activité

```
import android.app.Activity;
import android.os.Bundle;

public class Main extends Activity {
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

L'interface (code 3-2) a défini un composant graphique `TextView` vide. Pour y accéder depuis le code et pouvoir le manipuler, vous devez d'abord récupérer une instance de l'objet avec la méthode `findViewById` de l'activité. Une fois l'objet récupéré, vous pourrez le manipuler de la même façon que n'importe quel objet, par exemple pour modifier son texte. L'affichage répercutera le changement.

Le code suivant récupère le composant graphique `TextView` que nous avons nommé `monText` dans l'interface, puis utilise la méthode `setText` pour changer le contenu de son texte.

Code 3-4 : Récupérer un élément de l'interface et interagir avec

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
public class Main extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

     TextView monTexte = (TextView)findViewById(R.id.monText);
      monTexte.setText("Bonjour tout le monde !");
   }
}
```

Le résultat dans l'émulateur Android est celui de la figure 3-5.

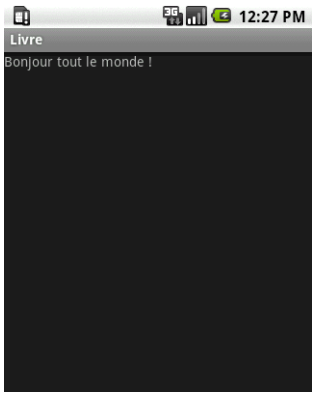


Figure 3–5
Le résultat de l'exemple 3-4

Créer une interface sans définition XML

Vous auriez pu arriver au même résultat depuis le code source de l'application, sans utiliser de définition d'interface XML. Cette technique possède quelques inconvénients, comme le fait de ne pas séparer la définition de sa présentation de la logique de cette dernière, ou encore de rendre le code plus difficile à maintenir par des profils métiers différents comme un développeur et un graphiste.

Code 3-5 : Créer une vue depuis le code source

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class Main extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView monTextView = new TextView(this);
        setContentView(monTextView);
        monTextView.setText("Bonjour tout le monde !");
    }
}
```

Dans cet exemple, nous n'avons pas appliqué de gabarit `LinearLayout` via `setContentView`. Nous avons uniquement instancié un composant graphique `TextView` puis utilisé la méthode `setContentView` en lui fournissant ce même `TextView`.

Ceci nous donne un résultat identique à l'exemple précédent avec fichier XML.

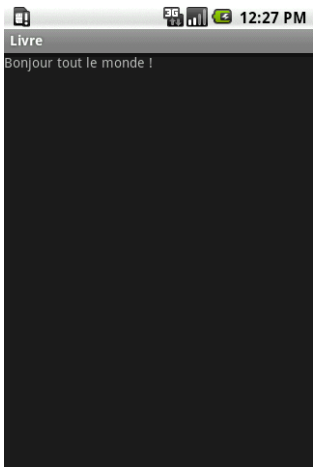


Figure 3–6
Résultat de l'exemple *3*-5

La méthode `setContentView` n'accepte qu'une seule vue. Si vous construisez votre interface depuis le code, vous aurez probablement plus d'une vue à intégrer dans votre activité. Il vous faudra donc réunir vos vues dans un gabarit de vues (par exemple le gabarit `LinearLayout`) et le transmettre à la méthode `setContentView`. Dans notre exemple, nous allons utiliser un gabarit `LinearLayout` pour intégrer plusieurs éléments `TextView` dans notre activité.

Code 3-6 : Créer un groupe de vues de façon programmatique

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.LinearLayout;
import android.widget.TextView;

public class Main extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Nous instancions un LinearLayout dans lequel nous
        // intégrerons nos différents TextView
        LinearLayout monLinearLayout = new LinearLayout(this);

        // Nous paramétrons monLinearLayout afin qu'il affiche
        // les vues les unes au-dessus des autres
        monLinearLayout.setOrientation(LinearLayout.VERTICAL);

        // Nous instancions nos deux TextViews à afficher
        TextView monTextView1 = new TextView(this);
        TextView monTextView2 = new TextView(this);

        // Nous ajoutons les deux TextViews dans notre monLinearLayout
        monLinearLayout.addView(monTextView1);
        monLinearLayout.addView(monTextView2);

        // Nous appliquons monLinearLayout sur notre activité
        setContentView(monLinearLayout);

        // Nous paramétrons un texte à afficher sur nos 2 TextViews
        monTextView1.setText("Bonjour tout le monde !");
        monTextView2.setText("Ceci est mon 2eme texte");
    }
}
```

Dans certains cas, vous serez amené à positionner des vues plutôt que de les aligner. Pour se faire, il faudra utiliser un gabarit `RelativeLayout` et le paramètre `gravity`, spécifiant l'attrait d'un composant vers un coin de l'écran.

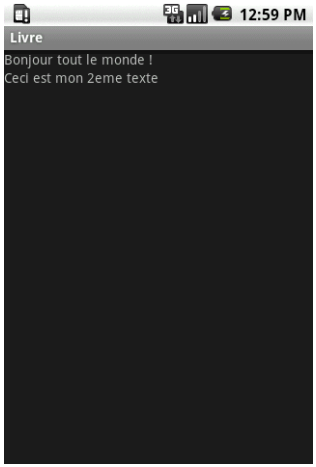


Figure 3–7
Le résultat dans l'émulateur de l'exemple *3*-6

Nous allons maintenant modifier la gravité de nos éléments `TextView` dans la définition de l'interface afin de les aligner en haut et en bas de notre gabarit.

Code 3-7 : Positionnement des vues en déclaratif

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical"
  >
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/monText"
    android:text="Texte en haut à droite"
    android:gravity="top|right"
    />
  <TextView
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="bottom|center_horizontal"
    android:id="@+id/monText2"
    android:text="Texte en bas au centre"
    />
</LinearLayout>
```

Vous noterez l'utilisation du caractère | pour spécifier une combinaison de valeurs et placer vos éléments au plus proche de vos besoins.

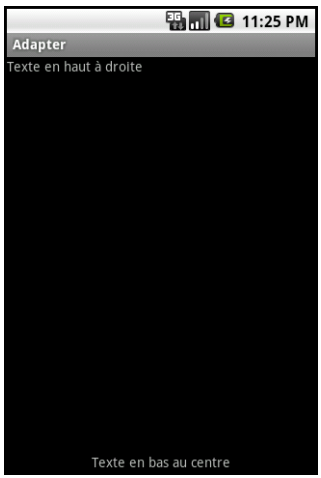


Figure 3–8
Résultat de l'exemple *3-7*

Dans cet exemple, la gravité `top | right` (haut | droite) est appliquée sur l'élément `TextView` (*texte en haut à droite*) qui occupe toute la largeur grâce à la valeur `fill_parent` du paramètre `layout_width`. Ce paramétrage indique au texte de se positionner le plus en haut et à droite de l'espace qu'il occupe. Si à la place de la valeur `fill_parent` nous avions spécifié `wrap_content`, le texte aurait été aligné visuellement à gauche car la place qu'occuperait l'élément `TextView` serait limitée à son contenu.

Pour aligner en bas le deuxième `TextView`, nous avons dû paramétrer la propriété `layout_width` et `layout_height` avec la valeur `fill_parent` et spécifier la valeur `bottom | center_horizontal` (bas | centre horizontal) à la propriété `gravity`.

Gérer les événements

Sous Android, toutes les actions de l'utilisateur sont perçues comme un événement, que ce soit le clic sur un bouton d'une interface, le maintien du clic, l'effleurement d'un élément de l'interface, etc. Ces événements peuvent être interceptés par les éléments de votre interface pour exécuter des actions en conséquence.

Le mécanisme d'interception repose sur la notion d'écouteurs, aussi appelés *listeners* dans la documentation Java. Il permet d'associer un événement à une méthode à appeler en cas d'apparition de cet événement. Si un écouteur est défini pour un élément graphique et un événement précis, la plate-forme Android appellera la méthode associée dès que l'événement sera produit sur cet élément. Par exemple, on

pourra définir un écouteur sur l'événement clic d'un bouton pour afficher un message « Bouton cliqué !». C'est justement ce que nous allons faire ci-après.

Notez que les événements qui peuvent être interceptés ainsi que les méthodes associées sont imposés. Pour un événement `OnClick` (élément cliqué), la méthode associée sera `OnClick()` : il nous suffira alors de définir cette méthode pour notre écouteur pour qu'elle soit appelée lorsque l'utilisateur cliquera sur l'élément graphique associé. Ainsi, pour que l'interface réagisse à un événement sur un élément, il faudra choisir l'élément et l'événement à intercepter, définir un écouteur sur cet événement et définir la méthode associée.

Avant de gérer un événement du type « cliquer sur un bouton », commençons par créer un gabarit avec un bouton centré au milieu de l'écran. Pour intégrer un bouton dans notre gabarit, il suffit d'ajouter une vue `Button`.

Modifiez les déclarations XML du fichier `main.xml` de l'exemple précédent pour y insérer un bouton.

Code 3-8 : Insertion d'un bouton dans l'interface

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_height="fill_parent"
  android:layout_width="fill_parent"
  android:gravity="center_vertical|center_horizontal"
  >
   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/monBouton"
      android:text="Cliquez ici !"
      >
   </Button>
</LinearLayout>
```

Une fois l'instance du bouton `monBouton` récupérée dans le code avec la référence `R.id.monBouton`, vous pouvez associer l'écouteur correspondant à l'événement désiré (ici, à l'aide de `setOnClickListener`.). Modifiez le code de l'activité `main.java`.

Code 3-9 : Création d'un écouteur sur un bouton

```
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
```

```
import android.widget.Button;
import android.widget.Toast;

public class Main extends Activity {

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      setContentView(R.layout.main);

      // Nous cherchons le bouton dans notre interface
      ((Button)findViewById(R.id.monBouton))
      // Nous paramétrons un écouteur sur l'événement 'click' de ce bouton
      .setOnClickListener(new OnClickListener() {
         @Override
         public void onClick(View v) {
            // Nous affichons un message à l'utilisateur
            Toast.makeText(Main.this, "Bouton cliqué !", Toast.LENGTH_LONG).show();
         }
      });
   }
}
```

Dans la méthode `onClick` de l'écouteur que nous avons redéfinie, nous déclenchons une alerte visuelle par l'intermédiaire d'un toast (les alertes visuelles telles que les toasts seront détaillées dans le chapitre 11 traitant des services).

Figure 3–9
Résultat de l'exemple 3-9 lorsque l'utilisateur clique sur le bouton

Intégrer des éléments graphiques dans votre interface

Nous venons de voir comment intégrer un bouton sur une interface mais il existe bien d'autres éléments graphiques que nous pouvons ajouter. Nous allons en voir quelques-uns dans cette partie de façon à construire vos premières interfaces, sachant que nous aborderons la création d'interfaces plus complexes dans le chapitre 5.

Intégrer une image dans votre interface

Pour ajouter une image dans votre interface, utilisez la vue de type `ImageView`. Vous pouvez spécifier la source de l'image directement dans votre définition XML, via l'attribut `src`, ou alors utiliser la méthode `setImageResource` en passant l'identifiant en paramètre.

Pour l'exemple suivant, nous utilisons une image nommée `icon.png` qui se trouve déjà dans les ressources de votre projet (cette ressource est incluse dans le modèle de projet Android d'Eclipse et se trouve dans le dossier `/res/drawable`).

Pour spécifier la taille d'une image dans la valeur de la propriété `src`, plusieurs options s'offrent à vous : soit vous utilisez la dimension réelle de l'image (`wrap_content`) ou la taille de l'écran (`fill_parent`), soit vous spécifiez la taille exacte (exemple : 100 px, 25 dp, etc).

Mettez à jour la définition de l'interface de l'application en insérant un élément `ImageView`.

Code 3-10 : Intégrer une vue image dans une interface

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_height="fill_parent"
   android:layout_width="fill_parent"
   android:gravity="center_vertical|center_horizontal"
  >
   <ImageView
      android:id="@+id/monImage"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:src="@drawable/icon"
   >
   </ImageView>
</LinearLayout>
```

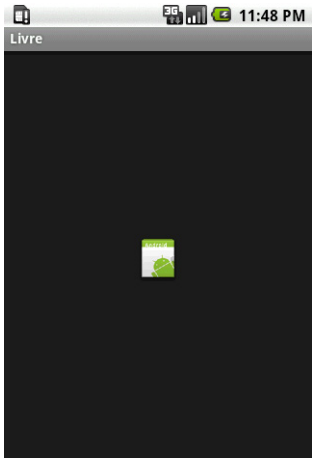


Figure 3–10
Le résultat de l'exemple *3*-10

Vous remarquerez que le code source de l'application n'a pas été changé : seul le code XML de l'interface a eu besoin d'être modifié pour afficher l'image. Si vous aviez souhaité spécifier la source de l'image directement dans le code :

```
ImageView monImage = ...
monImage.setImageResource(R.drawable.icon);
```

Intégrer une boîte de saisie de texte

Pour proposer à l'utilisateur de saisir du texte, vous pouvez utiliser la vue `EditText`.

L'exemple suivant montre l'utilisation de cette vue. Modifiez le contenu du fichier `main.xml` de votre projet pour y insérer la déclaration de la vue `EditText`.

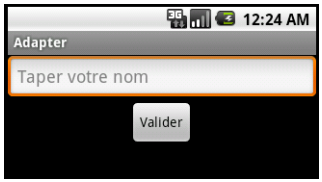
Code 3-11 : Ajout d'une vue de saisie de texte à l'interface utilisateur

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">
   <!--
      L'élément EditText permettra à
      l'utilisateur de saisir son nom.
   -->
   <EditText
      android:id="@+id/monEditText"
      android:layout_height="wrap_content"
      android:hint="Taper votre nom"
      android:layout_width="fill_parent">
```

```
    </EditText>
    <!--
       Le Bouton qui permettra à l'utilisateur
       de valider sa saisie
    -->
    <Button
       android:id="@+id/monBouton"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="Valider"
       android:layout_gravity="center_horizontal">
    </Button>
</LinearLayout>
```

Exécutez le projet à nouveau, après avoir enregistré la modification, pour voir le résultat.

Figure 3–11
Résultat de l'exemple 3-11



La propriété `hint` de l'élément `EditText` permet l'affichage en fond d'une aide. Tant qu'aucun texte n'est saisi dans la zone dédiée, cette aide apparaît grisée sur le fond. Cela peut vous éviter de mettre un texte de description avant ou au-dessus de chacune des zones de saisie et d'indiquer à l'utilisateur l'objet de sa saisie.

Voyons maintenant comment récupérer ce que l'utilisateur a inscrit lorsqu'il clique sur le bouton *Valider*.

Modifiez la méthode `onCreate` de l'activité `Main` de votre projet.

Code 3-12 : Récupérer la saisie de l'utilisateur

```
...

   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      // nous ajoutons un écouteur de type OnClickListener sur le bouton
       // de validation.
      ((Button)findViewById(R.id.monBouton)).setOnClickListener(
                                          ➥ new OnClickListener() {

         @Override
         public void onClick(View v) {
```

```
            // On récupère notre EditText
            EditText texte = ((EditText)findViewById(R.id.monEditText));
            // On garde la chaîne de caractères
            String nom = texte.getText().toString();
            // On affiche ce qui a été tapé
            Toast.makeText(Main.this, nom, Toast.LENGTH_SHORT).show();
        }
    });
  }
}

...
```

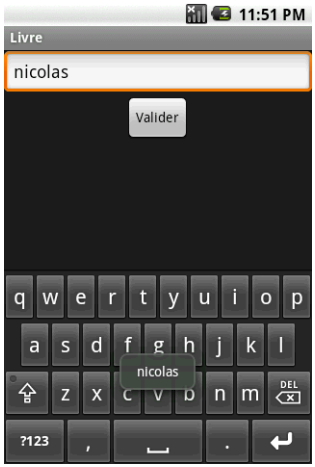


Figure 3–12
Résultat de l'exemple 3-12, après renseignement du nom et clic sur le bouton

Comme vous pouvez le voir, avec l'élément `EditText`, le clavier virtuel apparaît automatiquement dès que la zone de saisie est sélectionnée, sans que vous ayez besoin de le spécifier dans le code.

Intégrer d'autres composants graphiques

Nous allons maintenant intégrer dans notre interface plusieurs types de vues que vous serez amené à utiliser dans vos applications : case à cocher, bouton avec image, bouton radio, sélecteur de date, etc. Autant d'éléments que vous utiliserez couramment dans vos interfaces.

Dans l'exemple que nous vous proposons, nous allons ajouter les éléments suivants : une case à cocher (`CheckBox`), un bouton image (`ImageButton`), deux boutons radio (`RadioGroup` et `RadioButton`), un contrôle de saisie de date (`DatePicker`), une barre de vote (`RatingBar`) et deux horloges, l'une digitale (`DigitalClock`) et l'autre analogique (`AnalogClock`). Avant de vous expliquer comment fonctionne chacun ces éléments, voyons de façon concrète comment les intégrer dans votre application.

JARGON Widget, contrôle et composant graphique

Dans la littérature, les composants graphiques sont parfois nommés *widget* (contraction des mots anglais *window* et *gadget*) ou *contrôles* (de l'anglais *control*).
Dans ce chapitre, par soucis de clarté, nous privilégions la dénomination *composant graphique* (ou de façon plus générale *élément graphique*) mais vous retrouverez aussi les mots *widget* ou *contrôle* au détour d'une phrase dans le reste du livre. Gardez bien à l'esprit que lorsque contrôle est abordé dans le cadre d'une interface graphique, il s'agit de composant graphique.

Pour ajouter ces différents éléments, dans votre projet, modifiez le fichier de définition d'interface `main.xml`.

Code 3-13 : Diverses vues dans un même gabarit

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical" android:layout_width="fill_parent"
           android:layout_height="fill_parent">
   <!-- Case a cocher -->
   <CheckBox
      android:id="@+id/CheckBox01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Conditions acceptées ?">
   </CheckBox>
   <!-- Bouton avec une Image -->
   <ImageButton
      android:id="@+id/ImageButton01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:src="@drawable/icon">
      <!-- @drawable/icon est une Image qui se trouve dans le dossier /res/
drawable de notre projet -->
   </ImageButton>
   <!-- Groupe de boutons radio -->
   <RadioGroup
      android:id="@+id/RadioGroup01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:orientation="horizontal"
      android:layout_gravity="center_horizontal">
      <!-- Radio Bouton 1 -->
      <RadioButton
         android:id="@+id/RadioButton01"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Oui"
         android:checked="true">
```

```
         <!-- Nous avons mis checked=true par défaut sur notre 1er bouton
radio afin qu'il soit coché -->
      </RadioButton>
      <!-- Bouton radio 2 -->
      <RadioButton
         android:id="@+id/RadioButton02"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Non">
      </RadioButton>
   </RadioGroup>
   <!-- Sélectionneur de date -->
   <DatePicker
      android:id="@+id/DatePicker01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content">
   </DatePicker>
   <!-- Barre de vote -->
   <RatingBar
      android:id="@+id/RatingBar01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content">
   </RatingBar>
   <!-- Nous ajoutons un LinearLayout
      avec une orientation horizontale
      afin d'afficher de gauche à droite les views
      que nous allons grouper dedans -->
   <LinearLayout
      android:orientation="horizontal"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="center_horizontal">
      <!-- Horloge Digital -->
   <DigitalClock
      android:text="Horloge"
      android:id="@+id/DigitalClock01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="center_vertical">
   </DigitalClock>
   <!-- Horloge Analogique -->
   <AnalogClock
      android:id="@+id/AnalogClock01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content">
   </AnalogClock>
</LinearLayout>
```

De la même façon, modifiez votre fichier `main.java` pour y placer le code suivant.

Code 3-14 : Composition de diverses vues dans un même conteneur

```
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.DatePicker;
import android.widget.ImageButton;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.RatingBar;
import android.widget.Toast;

public class Main extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      // On récupère notre case à cocher pour intercepter l'événement
      // d'état (cochée ou pas)
      ((CheckBox)findViewById(R.id.CheckBox01)).setOnCheckedChangeListener(
                  ➥ new CheckBox.OnCheckedChangeListener() {
        public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
           afficheToast("Case cochée ? : " + ((isChecked)?"Oui":"Non"));
         }
      });

      // On récupère notre sélectionneur de date (DatePicker) pour attraper
      // l'événement du changement de date
      // Attention, le numéro de mois commence a 0 dans Android, mais pas les jours.
      // Donc si vous voulez mettre le mois de Mai, vous devrez fournir 4 et non 5
      ((DatePicker)findViewById(R.id.DatePicker01)).init(1977, 4, 29,
                  ➥ new DatePicker.OnDateChangedListener() {
         @Override
         public void onDateChanged(DatePicker view, int year, int monthOfYear,
                  ➥ int dayOfMonth) {
            // On affiche la nouvelle date qui vient d'être changée dans notre
            // DatePicker
            afficheToast("La date a changé\nAnnée : " + year + " | Mois : "
                  ➥ + monthOfYear + " | Jour : " + dayOfMonth);
         }
      });

      // On récupère notre barre de vote pour attraper la nouvelle note que
      // sélectionnera l'utilisateur
```

```
        ((RatingBar)findViewById(R.id.RatingBar01)).setOnRatingBarChangeListener(
                    ➥ new RatingBar.OnRatingBarChangeListener() {
           @Override
           public void onRatingChanged(RatingBar ratingBar, float rating,
                    ➥ boolean fromUser) {
              // On affiche la nouvelle note sélectionnée par l'utilisateur
              afficheToast("Nouvelle note : " + rating);
           }
        });

        // On récupère notre groupe de bouton radio pour attraper le choix de
        // l'utilisateur
        ((RadioGroup)findViewById(R.id.RadioGroup01)).setOnCheckedChangeListener(
                    ➥ new RadioGroup.OnCheckedChangeListener() {
           @Override
           public void onCheckedChanged(RadioGroup group, int checkedId) {
              // On affiche le choix de l'utilisateur
              afficheToast("Vous avez répondu : "
                    ➥ + ((RadioButton)findViewById(checkedId)).getText());
           }
        });

        // On récupère notre Bouton Image pour attraper le clic effectué par
        // l'utilisateur
        ((ImageButton)findViewById(R.id.ImageButton01)).setOnClickListener(
                    ➥ new OnClickListener() {
           @Override
           public void onClick(View v) {
              // On affiche un message pour signalé que le bouton image a été pressé
              afficheToast("Bouton Image pressé");
           }
        });
     }

     // Méthode d'aide qui simplifiera la tâche d'affichage d'un message
     public void afficheToast(String text)
     {
        Toast.makeText(this, text, Toast.LENGTH_SHORT).show();
     }
}
```

Dans cet exemple, nous employons diverses vues que vous serez souvent amené à utiliser :

- la case à cocher (`CheckBox`) : propose un choix basique à vos utilisateurs : « oui » ou « non ». Pour récupérer à tout moment l'état de la case à cocher, il vous suffit de le récupérer avec la méthode `isChecked`. Dans notre exemple nous récupérons l'état dès qu'il change, grâce à l'écouteur `OnCheckedChangeListener` que nous

avons associé à la case à cocher. Ceci nous permet de récupérer immédiatement le changement d'état et si nécessaire d'exécuter du code en conséquence ;

- le bouton image (`ImageButton`) : fonctionne de la même façon qu'un bouton classique mais présenté précédemment sous la forme de l'image choisie. La principale différence avec un bouton classique est que l'on ne peut ajouter un texte à afficher : seule l'image est affichée ;
- le groupe de boutons radio et les boutons radio (`RadioGroup` et `RadioButton`) : propose une liste de choix à réponse unique (seul un des boutons du groupe de boutons pourra être coché à un instant *t*). Afin que les boutons radio (de type `RadioButton`) puissent basculer de l'un à l'autre, vous devez les regrouper dans un élément `ViewGroup` de type `RadioGroup`. Si vos boutons radio ne sont pas regroupés dans un `RadioGroup`, ils seront tous indépendants. Dans notre exemple, nous attrapons le changement d'état des boutons radio dès qu'ils sont pressés grâce à l'écouteur `OnCheckedChangeListener`que nous avons associé au `RadioGroup`. Si vous souhaitez récupérer le bouton radio pressé à n'importe quel moment il vous suffira d'appeler la méthode `getCheckedRadioButtonId` ;
- le sélectionneur de date (`DatePicker`) : affiche les boutons nécessaires pour saisir une date. Attention celui-ci s'adapte visuellement à la langue et région du téléphone. Ainsi, un téléphone en langue « Anglais US » affiche la date sous la forme Mois-Jour-Année tandis que le même téléphone en langue française l'affiche sous la forme Jour-Mois-Année. Dans notre exemple, nous récupérons le changement de date de notre sélectionneur de date dès que l'utilisateur modifie une valeur grâce à l'écouteur `OnDateChangedListner` que nous lui avons associé. Pour récupérer les valeurs sélectionnées, vous pouvez à tout moment utiliser les méthodes`getDayOfMonth`, `getMonth`, `getYear`. Attention, comme signalé en commentaire dans l'exemple, le numéro des mois commence à 0 (janvier = 0, février = 1 etc.) ce qui n'est pas le cas pour les jours et les années ;
- la barre de vote (`RatingBar`) : affiche une barre comportant des étoiles pour représenter et/ou récupérer une notation. Dans notre exemple nous récupérons le changement de note dès que l'utilisateur modifie celle-ci grâce à l'écouteur `OnRatingBarChangeListener` associé. Vous pouvez également à tout moment dans votre code récupérer la note avec la méthode `getRating`.
- l'horloge digitale (`DigitalClock`) : affiche une horloge digitale dans votre application. Par exemple : 10:05:25 ;
- l'horloge analogique (`AnalogClock`) : affiche une horloge à aiguilles dans votre application.

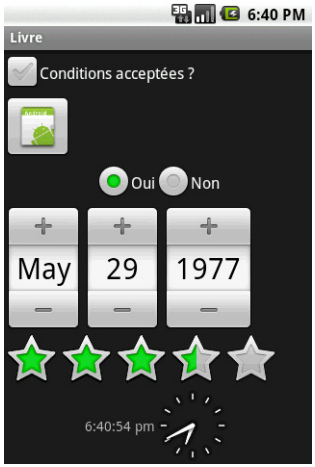


Figure 3–13
Résultat du code 3-14 dans l'émulateur Android

En ce qui concerne la barre de vote, vous pouvez définir la graduation de la note en modifiant le code XML comme ceci.

Code 3-15 : changement du pas pour la notation dans la barre de vote

```
<RatingBar
    android:id="@+id/RatingBar01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:stepSize="1"
</RatingBar>
```

Par défaut la vue `RatingBar` possède un pas de 0,5. Ce qui vous permet de récupérer des notes à 0,5 près, comme 3,5 par exemple. Dans le code ci-dessus, en précisant la propriété `stepSize=1` nous paramétrons le pas à 1. Ce qui veut dire que l'utilisateur ne pourra donner qu'une note entière.

Découper ses interfaces avec include

Comme dans les interfaces web, votre application comportera souvent des portions d'interfaces communes à l'ensemble de l'application (menu de navigation, en-tête et pied de page, etc). Ces portions d'interfaces communes – par exemple pour une application mobile, un bandeau supérieur contenant des informations – devront être dupliquées dans toutes les définitions de l'interface. Ce travail sera fastidieux et n'en sera que plus compliqué à maintenir puisque chaque modification devra être répercutée pour chaque définition d'interface.

Afin de combler cette lacune, la plate-forme Android dispose d'une instruction `include` permettant d'inclure dans la définition d'une interface, des éléments contenus dans une autre interface. Avec ce mécanisme, il suffit juste de modifier la partie de l'interface qui est incluse pour que la modification soit répercutée dans l'ensemble des interfaces.

Afin d'illustrer ce mécanisme, nous allons créer deux fichiers XML (un en-tête de page et un pied de page) qui seront ensuite inclus dans une autre interface.

Créez un premier fichier XML (l'en-tête de page) et nommez-le `include_haut.xml`.

Code 3-16 : Gabarit du haut de page

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content">
   <ImageView
      android:id="@+id/ImageView01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:src="@drawable/icon">
   </ImageView>
   <TextView
      android:id="@+id/TexteHaut"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Ceci est mon haut de page"
      android:layout_gravity="center_vertical">
   </TextView>
</LinearLayout>
```

Puis créez une second fichier XML (le pied de page) nommé `include_bas.xml`.

Code 3-17 : Gabarit du pied de page

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content">

   <TextView
      android:id="@+id/TextView01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Ceci est mon pied de page"
      android:layout_gravity="center_vertical">
   </TextView>
```

```
    <ImageView
        android:id="@+id/ImageView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/icon">
    </ImageView>

</LinearLayout>
```

Pour inclure une interface dans une autre, utilisez l'élément `<include>` à l'endroit où vous souhaitez insérer les éléments communs de l'interface et renseignez l'attribut `layout` pour spécifier l'identifiant de l'interface à inclure à cet emplacement.

Assemblons nos deux définitions d'interface d'en-tête et de pied de page (code 3-16 et 3-17) dans un fichier `main.xml`.

Code 3-18 : Inclusion d'interfaces dans le gabarit d'une interface

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">
    <!-- Inclusion de l'interface comportant l'en-tête de page -->
    <include
        android:id="@+id/include01"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        layout="@layout/include_haut">
    </include>

    <TextView
        android:text="Ici se trouve le texte et tout ce que\nje souhaite
        afficher\ndans mon application"
        android:id="@+id/TextView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    </TextView>

    <!-- Insertion du pied de page -->
    <!--
        Afin de positionner le pied de page en bas de l'écran, nous
        l'intégrons dans un LinearLayout prenant toute la place
        restante et paramétrons l'élément include avec le paramètre
        android:layout_gravity="bottom" pour le coller en bas.
    -->
```

```
    <LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      >
      <!-- Insertion de l'interface comportant le pied de page -->
       <include
          android:id="@+id/include02"
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          layout="@layout/include_bas"
          android:layout_gravity="bottom">
       </include>
    </LinearLayout>
</LinearLayout>
```

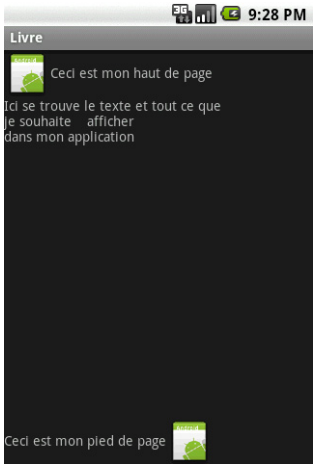


Figure 3–14
Résultat du *code* 3-18, après insertion de l'interface

Comme dans n'importe quelle interface, vous pouvez accéder aux éléments insérés.

Code 3-19 : Récupérer une vue d'un gabarit importé

```
// Nous récupérons notre gabarit d'inclusion.
LinearLayout monInclude = (LinearLayout)findViewById(R.id.include01);

// Nous récupérons le composant TextView qui se trouve
// dans la partie importée de l'interface.
TextView monText = (TextView)monInclude.findViewById(R.id.TexteHaut);

// Nous changeons le texte de ce composant.
monText.setText("Nouveau texte dans le haut de page");
```

Par sa simplicité de réalisation, ce mécanisme d'importation d'interface permet aux développeurs d'être souvent plus efficaces dans la réalisation et la maintenance des applications Android, dans la mesure où l'interface (que l'on inclut) n'a besoin d'être écrite qu'à un seul endroit pour être présente partout.

Ajouter des onglets

La taille de l'écran d'un téléphone étant limitée, vous serez souvent amené à utiliser des onglets pour afficher tout ce que votre application peut mettre à disposition de l'utilisateur.

Pour créer des onglets dans votre interface, vous devez d'abord ajouter un élément `TabHost` dans le gabarit de l'interface. À ce `TabHost`, vous devez ajouter un `TabWidget` comportant impérativement un identifiant `android:id="@android:id/tabs"`. Enfin, vous devrez ajouter un `ViewGroup` comportant les vues à afficher pour chaque onglet. Pour mettre en pratique ce que nous venons d'écrire, créez une interface dans un fichier `main.xml`, comme dans le code suivant.

Code 3-20 : Intégrer des onglets à l'interface utilisateur

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
<!-- Le TabHost qui contient tous les éléments de nos onglets
-->
<TabHost
   android:id="@+id/TabHost01"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <!-- TabWidget qui sert à afficher les onglets -->
      <TabWidget android:id="@android:id/tabs"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content">
      </TabWidget>
      <!-- contenu de nos onglets. -->
      <FrameLayout
         android:id="@android:id/tabcontent"
         android:layout_width="fill_parent"
         android:layout_height="fill_parent">
```

```
        <!-- Contenu de l'onglet N°1 -->
        <LinearLayout
           android:orientation="vertical"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:scrollbars="vertical"
            android:id="@+id/Onglet1">
           <TextView
              android:text="Ceci est un texte dans l'onglet N°1"
              android:id="@+id/TextViewOnglet1"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content">
           </TextView>
        </LinearLayout>
        <!-- Contenu de l'onglet N°2 -->
        <LinearLayout
            android:orientation="vertical"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:id="@+id/Onglet2">
           <TextView
              android:text="Ceci est un texte dans l'onglet N°2"
              android:id="@+id/TextViewOnglet2"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content">
           </TextView>
           <ImageView
              android:id="@+id/ImageView01"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:src="@drawable/icon">
           </ImageView>
        </LinearLayout>
        <!-- Contenu de l'onglet N°3 -->
        <LinearLayout
            android:orientation="vertical"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:id="@+id/Onglet3">
           <TextView
              android:text="Ceci est un texte dans l'onglet N°3"
              android:id="@+id/TextViewOnglet3"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content">
           </TextView>
        </LinearLayout>
      </FrameLayout>
   </LinearLayout>
</TabHost>
</LinearLayout>
```

Dans le code de l'activité associée à l'interface, insérez les lignes suivantes.

Code 3-21 : Intégrer des onglets à l'interface utilisateur

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TabHost;
import android.widget.Toast;
import android.widget.TabHost.TabSpec;

public class Main extends Activity {

   private TabHost monTabHost;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main7);

      // Récupération du TabHost
      monTabHost =(TabHost) findViewById(R.id.TabHost01);
      // Avant d'ajouter des onglets, il faut impérativement appeler la méthode
      // setup() du TabHost
      monTabHost.setup();

      // Nous ajoutons les 3 onglets dans notre TabHost

      // Nous paramétrons le 1er Onglet
      TabSpec spec = monTabHost.newTabSpec("onglet_1");
      // Nous paramétrons le texte qui s'affichera dans l'onglet
      // ainsi que l'image qui se positionnera
      // au dessus du texte.
     spec.setIndicator("Onglet 1",getResources().getDrawable(R.drawable.icon));
      // On spécifie le Layout qui s'affichera lorsque l'onglet sera sélectionné
      spec.setContent(R.id.Onglet1);
      // On ajoute l'onglet dans notre TabHost
      monTabHost.addTab(spec);

      // Vous pouvez ajouter des onglets comme ceci :
      monTabHost.addTab(monTabHost.newTabSpec("onglet_2").setIndicator(
            ➥ "Onglet 2").setContent(R.id.Onglet2));
      monTabHost.addTab(monTabHost.newTabSpec("onglet_3").setIndicator(
            ➥ "Onglet 3").setContent(R.id.Onglet3));

      // Nous paramétrons un écouteur onTabChangedListner pour récupérer
      // le changement d'onglet.
      monTabHost.setOnTabChangedListener(
            new TabHost.OnTabChangeListener (){
               public void onTabChanged(String tabId){
```

```
                    // Vous pourrez exécuter du code lorsqu'un
                    // onglet est cliqué. Pour déterminer
                    // quel onglet a été cliqué, il
                    // vous suffira de vérifier le tabId envoyé lors
                    // du clic et d'exécuter votre code en
                    // conséquence.
                    Toast.makeText(Main.this, "L'onglet avec l'identifiant : "
                        ➥ + tabId + " a été cliqué", Toast.LENGTH_SHORT).show();
                }
            }
        );
    }
}
```

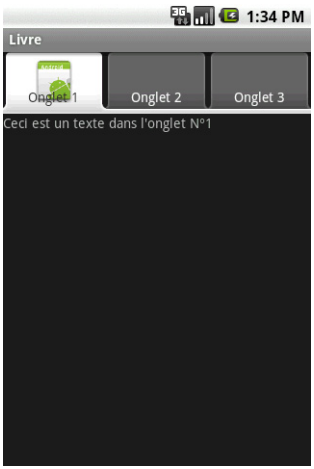


Figure 3–15 Résultat des codes 3-20 et 3-21

Figure 3–16 Résultat des codes 3-20 et 3-21 : sélection du deuxième onglet

Comme vous pouvez le constater, le contenu du deuxième onglet est bien différent du premier et comporte bien tous les éléments que nous avons paramétrés dans la déclaration d'interface en XML. Lorsque que vous cliquez sur un autre onglet, le message associé au clic apparaît et vous pouvez exécuter du code en conséquence si nécessaire.

Dans certains cas, votre contenu pourra comporter plus d'éléments qu'il ne peut en afficher à l'écran. Dans ce cas, vous pourrez faire en sorte de faire défiler le contenu avec l'aide d'un ascenseur défini par l'élément `ScrollView` (ou vue de défilement). La particularité du `Scrollview` est qu'il ne peut avoir qu'un seul enfant. Pour pallier cette limitation, insérez un autre gabarit, par exemple un `LinearLayout`, qui affichera alors tous les éléments du gabarit avec une barre de défilement sur l'écran.

Code 3-22 : Utilisation d'une vue de défilement ScrollView

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
   <ScrollView
      android:id="@+id/ScrollView01"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">
      <LinearLayout
         android:id="@+id/LinearLayout01"
         android:layout_width="fill_parent"
         android:layout_height="fill_parent"
         android:orientation="vertical">
         <TextView
            android:text="Ceci est un texte\nComportant plusieurs\nlignes"
            android:id="@+id/TextView01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
         </TextView>
         <Button
            android:text="Mon bouton"
            android:id="@+id/Button01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
         </Button>
         <ImageView
            android:id="@+id/ImageView01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:src="@drawable/icon">
         </ImageView>
         <TextView
            android:text="Ceci est un texte\nComportant plusieurs\nlignes"
            android:id="@+id/TextView01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
         </TextView>
         <DatePicker android:id="@+id/DatePicker01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
         </DatePicker>
         <TextView
            android:text="Ceci est un texte\nComportant plusieurs\nlignes"
            android:id="@+id/TextView01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
         </TextView>
```

```
        <DatePicker
           android:id="@+id/DatePicker02"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content">
        </DatePicker>
     </LinearLayout>
  </ScrollView>
</LinearLayout>
```

Figure 3–17
Résultat du code 3-22 avec l'utilisation de la vue ScrollView

Comme vous pouvez le voir, une barre de défilement est visible sur la droite et est prise en compte pour l'ensemble du `LinearLayout` qui se trouve dans le `ScrollView`.

Différents outils permettant de réaliser une interface sont inclus dans le module ADT d'Eclipe. Ces outils sont présentés en annexe du présent ouvrage.

En résumé

La création d'interface est rendue simple et efficace sur la plate-forme Android. Ce chapitre n'est que la première partie sur ce sujet. Le chapitre 5, dédié également aux interfaces graphiques, vous montrera comment aller plus loin avec les vues et les gabarits, mais aussi comment créer vos propres composants.

L'interface d'une application est un élément essentiel : son succès reposera bien souvent sur une interface claire et intuitive. Sachez garder vos interfaces épurées et au plus proche du fonctionnement du système ou des applications standards. De cette façon l'utilisateur ne sera pas perturbé et aura l'impression d'utiliser votre application comme si elle faisait partie intégrante du système.

4

Communication entre applications : la classe Intent

Au cœur du système Android, les Intents forment un mécanisme sophistiqué et complet permettant aux activités et aux applications d'interagir entre elles.

La communication interne du système Android est basée sur l'envoi et la réception de messages exprimant l'intention d'une action. Représentant une description abstraite d'une opération à effectuer, chacun de ces messages peut être émis à destination d'un autre composant de la même application (une activité, un service, etc.) ou celui d'une toute autre application.

Issu de la classe `Intent`, ce message permet de véhiculer toutes les informations nécessaires à la réalisation de l'action :

- informations à destination du composant qui le réceptionnera (action à effectuer et les données avec lesquelles agir) ;
- informations nécessaires au système pour son traitement (catégorie du composant cible du message et instructions d'exécution de l'action).

Nous verrons plus loin que le terme d'action peut simplement correspondre à la transmission d'informations entre applications.

Par exemple, le démarrage des composants d'une application (activités, services, etc.) est réalisé au moyen d'un objet `Intent`. L'utilisation d'un composant externe peut ainsi être décidée au moment de l'exécution de l'application et non lors de la compilation de

l'application. Ce mécanisme permet d'éviter les dépendances et de rendre beaucoup plus dynamique et pertinente l'utilisation des applications dans un contexte donné. Par exemple, si vous souhaitez utiliser un navigateur web pour afficher une page particulière depuis votre application, vous n'allez pas cibler un navigateur en particulier, mais demander au système d'ouvrir un navigateur qui peut être celui fourni par défaut par le système ou un navigateur alternatif que l'utilisateur préfère utiliser.

> **JARGON La métaphore de l'intention**
>
> Vous trouverez parfois l'objet Intent traduit en français par « intention » (y compris, rarement, dans le présent livre). En effet, la métaphore se justifie dans la mesure où l'objet correspond au « souhait » d'une action, mais sans en expliciter les modalités, en déléguant parfois au système le choix de l'application cible.

Vous pouvez envoyer des Intents au système de deux façons : soit en ciblant un composant précis d'une application (on parle alors de mode *explicite*), soit en laissant le système déléguer le traitement (ou non) de cette demande au composant le plus approprié (on parle alors de mode *implicite*).

Un système de filtres permet à chaque application de filtrer et de gérer uniquement les Intents qui sont pertinents pour celle-ci. Une application peut ainsi être dans un état d'inactivité, tout en restant à l'écoute des intentions circulant dans le système.

Principe de fonctionnement

Les objets `Intent` ont essentiellement trois utilisations : ils permettent de démarrer une activité au sein de l'application courante ou de solliciter d'autres applications et d'envoyer des informations.

Le démarrage d'une activité au sein d'une même application est utilisée pour la navigation entre écrans d'une interface graphique et l'appel d'une boîte de dialogue. C'est le seul cas où vous devrez démarrer une activité en mode explicite.

Lorsqu'un besoin ne peut être satisfait par l'application elle-même, elle peut solliciter une autre application pour y répondre (le besoin peut être l'exécution d'une action ou bien la transmission d'informations). Elle peut se contenter de transmettre son intention au système qui, lui, va se charger trouver l'application et le composant le plus approprié puis démarrer ce dernier et lui transmettre l'Intent correspondant. On parle alors de *résolution de l'intention* : à un contexte donné, le système choisit au mieux l'application correspondant à un objet `Intent` donné.

Les Intents ont aussi d'autres utilisations, dont le démarrage d'un service. Le mécanisme relatif aux objets `Intent` et leur utilisation sont en effet indispensables pour les applica-

tions fonctionnant en arrière plan (telles que les services, détaillés au chapitre 11) afin de recevoir des actions à effectuer (par exemple pour un service de lecteur multimédia, les actions pourront être de lire une musique, passer à la chanson suivante ou encore mettre la lecture en pause en cas d'appel entrant) mais également pour pouvoir communiquer avec d'autres applications.

Il est aussi possible de vouloir diffuser un objet `Intent` à plusieurs applications (par exemple pour informer l'ensemble des applications ouvertes que la batterie est défaillante). Le système pouvant très vite devenir verbeux, les applications peuvent mettre en place un filtre permettant de ne conserver que les Intents que l'application juge nécessaires.

Nous allons regarder de plus près comment mettre en pratique ces différentes utilisations, ainsi que la notion de diffusion multiple et de réception dans ce qui suit.

ZOOM L'objet Intent

Un objet `Intent` véhicule toutes les informations nécessaires à la réalisation d'une action (ou à la réception d'information) :

- *le nom du composant ciblé* : cette information facultative permet de spécifier de façon non ambigüe le nom du composant qui sera utilisé pour réaliser l'opération. Si le nom du composant n'est pas renseigné, le système déterminera le composant le plus approprié ;
- *l'action* : une chaîne de caractères définissant l'action à réaliser. Dans le cadre d'un récepteur d'Intents, il s'agira de l'action qui s'est produite et pour laquelle le système ou l'application informe toutes les autres ;
- *les données* : le type de contenu MIME sous la forme d'une chaîne de caractères et le contenu ciblé sous la forme d'un URI. Par exemple, si vous souhaitez afficher une page web, vous utiliserez l'action `ACTION_VIEW` et un URI de la forme http://<adresse du site> ;
- *les données supplémentaires* : sous la forme d'une collection de paires clé/valeur, vous pouvez spécifier des paramètres supplémentaires. Par exemple, si l'Intent est une demande d'envoi d'un courriel, l'utilisation de la constante `EXTRA_EMAIL` permet de préciser les adresses de messagerie des destinataires ;
- *la catégorie* : cette information complémentaire permet de cibler plus précisemment qui devra gérer l'Intent émis. Par exemple, l'utilisation de la constante `CATEGORY_BROWSABLE` demande le traitement par un navigateur alors que la constante `CATEGORY_LAUNCHER` spécifie que l'activité est le point d'entrée de l'application ;
- *les drapeaux* : principalement utilisés pour spécifier comment le système doit démarrer une activité. Par exemple, l'utilisation de la constante `FLAG_ACTIVITY_NO_ANIMATION` spécifie que l'activité doit être démarrée sans jouer l'animation de transition entre écrans.

Naviguer entre écrans au sein d'une application

Une application est souvent composée de plusieurs écrans qui s'enchaînent les uns à la suite des autres en fonction de l'utilisateur et chaque écran est représenté par une activité définissant son interface utilisateur et sa logique. La principale utilisation

d'un Intent est le démarrage de ces activités (une à la fois) permettant cet enchaînement. De façon plus générale, chaque composant de votre application nécessitera l'emploi d'un Intent pour être démarré.

Il existe deux méthodes pour démarrer une activité, en fonction de la logique de l'interface : parfois nous aurons besoin de savoir comment s'est déroulée l'activité (et obtenir un retour lors de son arrêt), parfois non. Nous commencerons par ce dernier cas.

Démarrer une activité

Pour démarrer une activité sans attendre de retour, utilisez la méthode `startActivity` avec comme paramètre une instance de la classe `Intent` spécifiant le type de classe de l'activité à exécuter :

Code 4-1 : Démarrer une activité de façon explicite

```
Intent intent = new Intent(this,ActiviteADemarrer.class);
startActivity(intent);
```

Le constructeur de la classe `Intent` prend les paramètres suivants :

- **`Context PackageContext`** : le contexte à partir duquel l'Intent est créé et sera envoyé. Ce paramètre fait référence la plupart du temps à l'activité en cours pointée par le mot clef `this` ;
- **`Class<?> cls`** *:* un type de classe Java héritant de la classe `Activity` et pointé ici par le mot clef `ActiviteADemarrer.class`.

> **ERREUR DU DÉBUTANT** **Exception lors du démarrage d'une activité**
>
> Avant de pouvoir démarrer l'activité, il faut au préalable la déclarer dans le fichier de configuration `AndroidManifest.xml` de l'application. Sans cela, une erreur du type `ActivityNotFoundException` sera générée lors de l'appel de la méthode `startActivity`.

Voici un exemple de déclaration pour une activité s'appelant `MonActivite` devant figurer a minima à l'intérieur de la section `<application>` du fichier de configuration de l'application :

Code 4-2 : Déclarer une activité dans AndroidManifest.xml

```
<application ...>
   <activity android:name=".MonActivite" />
</application>
```

> **À RETENIR Spécifier le nom de la classe dans l'attribut android:name**
>
> L'attribut `android:name` attend le nom complet de la classe. Cela comprend l'intégralité de l'espace de nom, le nom du paquetage et le nom de la classe. Si une classe `MaClasse` est dans le paquetage `maCompagnie.monApplication`, alors le nom complet sera `maCompagnie.monApplication.MaClasse`.

Démarrer une activité et obtenir un retour

Comme vu précédemment, la méthode `startActivity` de la classe `Activity` permet de démarrer une autre activité (que nous appellerons ci-après « activité enfant »). Cette méthode, bien que très utile dans son usage courant, ne propose cependant aucun mécanisme de retour d'information vers l'activité « parent » qui a démarré l'activité enfant. En d'autres termes, vous ne serez jamais averti de l'état de l'activité enfant que vous aurez lancée. Prenons comme exemple une activité enfant proposant à l'utilisateur un formulaire de réponse *Oui / Non*. Comment récupérer la valeur saisie par l'utilisateur dans l'activité enfant depuis l'activité principale ?

Figure 4–1
Exemple de formulaire Oui / Non

Pour gérer ce type de scénario, à savoir lancer une activité enfant et en connaître sa valeur de retour, il est possible d'utiliser la méthode `startActivityForResult` prévue à cet effet. Avec cette méthode, lorsque l'activité enfant aura terminé sa tâche, elle en avertira l'activité parent.

> **À NOTER Activités et sous-activités**
>
> Afin de différencier les activités enfant avec et sans retour vers l'activité parent, nous emploierons ci-après le terme de *sous-activité* lorsque l'activité enfant sera démarrée avec la méthode `startActivityForResult`. Nous allons maintenant voir comment gérer ce retour entre sous-activité et application principale.

Pour démarrer une activité enfant utilisez la méthode `startActivity` ou `startActivityForResult`. Avec cette dernière méthode, lorsque la sous-activité aura terminé sa tâche, elle en avertira l'activité parent et lui communiquera une valeur en retour.

Après avoir créée l'activité enfant et défini son existence dans le fichier de configuration `AndroidManifest.xml` de l'application, démarrez celle-ci en exécutant le code suivant :

Code 4-3 : Démarrer une activité avec retour

```
...
private static final int CODE_MON_ACTIVITE = 1;
...
Intent intent = new Intent(this,lambda.class);
// La constante CODE_MON_ACTIVITE représente l'identifiant de la requête
// (requestCode) qui sera utilisé plus tard pour identifier l'activité
// renvoyant la valeur de retour.
startActivityForResult(intent, CODE_MON_ACTIVITE);
```

MÉTHODE Identifier votre sous-activité par un requestCode

Vous noterez la constante `CODE_MON_ACTIVITE` définie par le développeur afin d'identifier quelle activité a provoqué le retour de valeur. Un simple `switch(requestCode)` permettra ainsi de traiter plusieurs valeurs de retour dans une même méthode. Si la valeur est inférieure à 0, l'activité parent n'attend pas de retour, ce qui équivaut à utiliser la méthode `startActivity`.

Renvoyer une valeur de retour

Pour renvoyer la valeur de retour à l'activité principale, appelez la méthode `setResult` de la classe `Activity` en indiquant comme paramètre le code de retour. Android prévoit plusieurs valeurs par défaut telles que `RESULT_OK` et `RESULT_CANCELED`, que nous utilisons dans l'exemple ci-dessous :

Code 4-4 : Renvoyer une valeur

```
@Override
public void onClick(View v) {
   switch(v.getId()){
   case R.id.button1:
      setResult(RESULT_OK);
      finish();
      break;

   case R.id.button2:
      setResult(RESULT_CANCELED);
      finish();
      break;
   }
}
```

Récupérer la valeur de retour

Pour récupérer la valeur de retour envoyée par une activité enfant, utilisez la méthode `onActivityResult` de l'activité parent, depuis laquelle vous avez appelé la méthode `startActivityForResult` :

Code 4-5 : Agir en fonction de la valeur de retour

```
@Override
protected void onActivityResult(int requestCode
   , int resultCode
   , Intent data) {
   // Le code de requête est utilisé pour identifier l'activité enfant
   switch(requestCode){
   case CODE_MON_ACTIVITE:
      switch(resultCode){
      case RESULT_OK:
         Toast.makeText(this
            , "Action validée"
            , Toast.LENGTH_LONG).show();
         return;
      case RESULT_CANCELED:
         Toast.makeText(this
            , "Action annulée"
            , Toast.LENGTH_LONG).show();
         return;
      default:
         // Faire quelque chose
         return;
      }
   default:
      // Faire quelque chose
      return;
   }
}
```

À RETENIR Spécificité de la méthode setResult

La méthode `setResult` ne fonctionnera pas comme espéré si l'activité parent est définie avec l'attribut `android:launchMode="singleInstance"`. En effet, la méthode `OnActivityResult` de l'activité parent sera appelée dès le lancement de la sous-activité et non à la fin de l'exécution de l'activité enfant ; de ce fait votre appel à la méthode `setResult` dans l'activité enfant sera vain.

La méthode `onActivityResult` utilise trois paramètres pour identifier l'activité et ses valeurs de retour :

- `int requestCode` : valeur identifiant quelle activité a appelé la méthode (par exemple, la valeur de la constante `CODE_MON_ACTIVITE` définie dans le code 4-3).

Cette même valeur a été spécifiée comme paramètre de la méthode `startActivityForResult` lors du démarrage de la sous-activité ;

- **`int resultCode`** : représente la valeur de retour envoyée par la sous-activité pour signaler son état à la fin de la transaction. C'est une constante définie dans la classe `Activity` (`RESULT_OK`, `RESULT_CANCELED`, etc.) ou par ledéveloppeur ;
- **`Intent data`** : cet objet permet d'échanger des données comme nous l'avons vu précédemment.

Solliciter d'autres applications

Nous avons déjà utilisé plusieurs fois un objet `Intent` pour démarrer des activités (via le nom du type d'activité) au sein d'une même application. Cependant, qu'en est-il si vous souhaitez faire appel à un composant d'une autre application, par exemple pour ouvrir une page web ? La réponse est : toujours en utilisant un Intent !

En effet, l'envoi d'un Intent permet également de demander à un composant d'une autre application que la vôtre de traiter l'action que vous souhaiteriez réaliser. C'est le système qui décide alors de l'application à utiliser pour accomplir votre souhait. Pour décider du composant le plus approprié, le système se base sur les informations que vous spécifiez dans votre objet `Intent` : action, données, catégorie, etc.

Ainsi vous exprimez votre intention au système et le système se chargera de résoudre votre intention pour vous proposer le composant de l'application le plus approprié. Ce mécanisme permet d'éviter les dépendances vers des applications puisque l'association entre votre application et le composant nécessaire se fait au moment de l'exécution et non de la compilation. Cela permet de lancer des activités en fonction du contexte et de ne pas se soucier de l'application qui sera réellement utilisée.

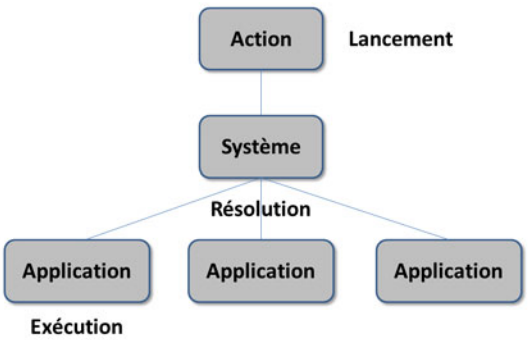


Figure 4–2
Mode de résolution des intentions

Ce système d'utilisation d'Intent implicite, puisque le système doit résoudre celle-ci en fonction de son environnement, recourt à des filtres comme points d'entrée pour distribuer les intents aux composants les plus appropriés. Les informations qui sont utilisées pour la résolution sont : l'action, les données (l'URI et le type de contenu MIME) et la catégorie. Les autres données ne jouent pas de rôle dans la résolution et la distribution des Intent aux applications.

> **EN SAVOIR PLUS SUR Résolution par les filtres d'intents**
>
> Si vous souhaitez approfondir vos connaissances concernant la résolution des Intents reçus par le système Android et les filtres d'Intents, nous vous conseillons de consulter la documentation officielle qui décrit le mécanisme de façon exhaustive :
> - http://developer.android.com/guide/topics/intents/intents-filters.html

Déléguer au système le choix de l'application

Vous pouvez envoyer une intention et demander au système de choisir le composant le plus approprié pour exécuter l'action transmise. Prenons un exemple : nous souhaitons composer un numéro de téléphone. Nous allons utiliser le type d'action `ACTION_DIAL` (voir la liste des actions natives plus loin dans ce chapitre) permettant de demander au système de composer un numéro.

Pour pouvoir demander au système de réaliser cette action, nous créons simplement un nouvel objet `Intent` dont nous spécifions le type d'action et les valeurs complémentaires dans le constructeur ; et cela sans préciser le type de la classe ciblée. Puis démarrez une nouvelle activité en spécifiant cet objet `Intent`. À la charge du système, en fonction des filtres d'Intents déclarés par les applications, de déterminer à quel composant d'une quelconque application envoyer cette demande d'action.

Pour appeler un numéro de téléphone, l'intention que nous enverrons au système sera composée d'une action et d'un URI comportant le numéro à appeler. Le système déterminera quelle application ouvrir, par défaut l'interface de composition de numéro ou votre application si vous avez spécifié le filtre d'Intents adéquat :

Code 4-6 : Composer un numéro de téléphone

```
Uri uri = Uri.parse("tel:0612345678");
Intent intent = new Intent(Intent.ACTION_DIAL, uri);
startActivity(intent)
```

> **A SAVOIR Les URIs (Uniform ressource Identifier)**
>
> Un URI (*Uniform ressource Identifier*) est un identifiant unique permettant d'identifier une ressource de façon non ambigüe sur un réseau. Le W3C a mis au point une recommandation (RFC 3986) afin d'en spécifier la syntaxe.

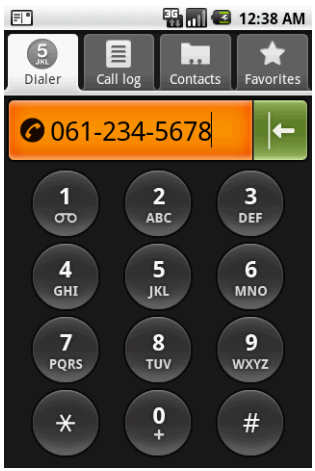


Figure 4–3
Lancement de la composition d'un numéro

> **A RETENIR Format des URI pour les actions**
>
> Le format des URI spécifiés comme données pour les actions dépendent du type d'action. Quoiqu'il en soit, la composition d'un URI se découpe toujours selon le format suivant : *schéma://hote:port/chemin*. Le schéma '*tel*' permet d'émettre un numéro de téléphone, '*market*' d'atteindre l'Android Market, '*www*' d'ouvrir une page web, etc.

En plus des actions natives d'Android pouvant être gérées par des applications présentes par défaut sur la plate-forme (comme « appeler un correspondant » ou « ouvrir une page web »), vous pourrez créez vos propres Intents pour étendre les possibilités du système. Pour créer un objet `Intent` personnalisé, héritez de la classe `Intent` et redéfinissez les constructeurs et méthodes nécessaires.

Les actions natives

Le système Android propose plusieurs actions natives dont voici une liste non exhaustive.

Tableau 4–1 Actions natives sous Android

Action	Définition
`ACTION_ANSWER`	Prendre en charge un appel entrant.
`ACTION_CALL`	Appeler un numéro de téléphone. Cette action lance une activité affichant l'interface pour composer un numéro puis appelle le numéro contenu dans l'URI spécifiée en paramètre.
`ACTION_DELETE`	Démarrer une activité permettant de supprimer une donnée identifiée par l'URI spécifiée en paramètre.

Tableau 4–1 Actions natives sous Android (suite)

Action	Définition
ACTION_DIAL	Afficher l'interface de composition des numéros. Celle-ci peut être pré-remplie par les données contenues dans l'URI spécifiée en paramètre.
ACTION_EDIT	Éditer une donnée.
ACTION_SEARCH	Démarrer une activité de recherche. L'expression de recherche de la pourra être spécifier dans la valeur du paramètre SearchManager.QUERY envoyé en extra de l'action.
ACTION_SEND	Envoyer des données texte ou binaire par courriel ou SMS. Les paramètres dépendront du type d'envoi.
ACTION_SENDTO	Lancer une activité capable d'envoyer un message au contact défini par l'URI spécifiée en paramètre.
ACTION_VIEW	Démarrer une action permettant de visualiser l'élément identifié par l'URI spécifiée en paramètre. C'est l'action la plus commune. Par défaut les adresses commençant par *http:* lanceront un navigateur web, celles commençant par *tel:* lanceront l'interface de composation de numéro et celles débutant par *geo:* lanceront Google Map.
ACTION_WEB_SEARCH	Effectuer une recherche sur Internet avec l'URI spécifiée en paramètre comme requête.

Par exemple, nous pouvons démarrer l'application Android Market pour pointer sur la fiche de notre application (si celle-ci a été déployée sur le marché Android) :

Code 4-7 : Lancer l'Android Market avec un Intent

```
private void doMarket(){
   Uri uri = Uri.parse(
      "market://search?q=pub:\"Nicolas et Emmanuel\"");
   Intent intent = new Intent(Intent.ACTION_VIEW, uri);
   startActivity(intent);
}
```

ou simplement lancer un navigateur avec une page web définie :

Code 4-8 : Démarrer un navigateur internet

```
private void doWebBrowser(){
   Uri uri = Uri.parse("http://www.google.fr/");
   Intent intent = new Intent(Intent.ACTION_VIEW, uri);
   startActivity(intent);
}
```

> ALLER PLUS LOIN **La documentation du marché Android**
>
> Vous trouverez plus de détails sur le marché Android dans la documentation officielle disponible en ligne à l'adresse :
> ▸ http://developer.android.com/guide/publishing/publishing.html

Accorder les permissions liées aux actions

Certaines actions requièrent des privilèges spéciaux qui vous amèneront à ajouter des permissions dans le fichier de configuration de votre application.

Par exemple, pour émettre un appel depuis votre application, il ne suffira pas uniquement de créer un objet `Intent` avec l'action `ACTION_CALL`. Il vous faudra également ajouter la permission `android.permission.CALL_PHONE` comme indiqué plus bas :

Code 4-9 : Appeler un numéro de téléphone

```
Uri uri = Uri.parse("tel:0612345678");
Intent intent = new Intent(Intent.ACTION_CALL, uri);
startActivity(intent);
```

> ALLER PLUS LOIN **La liste complète des permissions**
>
> Vous trouverez la liste de toutes les permissions dans la documentation Android à cette adresse :
> ▸ http://developer.android.com/reference/android/Manifest.permission.html

Pour ajouter la permission liée à cette action, insérez la ligne suivante dans la partie `<manifest>` du fichier de configuration de l'application :

Code 4-10 : Spécifier les permissions de l'application

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

Filtrer les actions

Android permet aux applications de spécifier quelles sont les actions qu'elles gèrent : le système peut ainsi choisir le composant le mieux adapté au traitement d'une action (véhiculée dans un objet `Intent`).

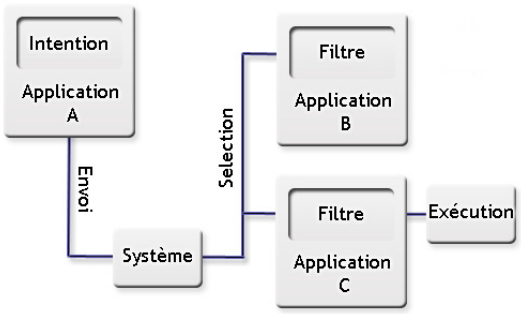


Figure 4–4
Mode de filtrage des actions gérées par les composants de l'application

Pour définir un filtre d'Intents, déclarez l'élément `intent-filter` dans le fichier de configuration de l'application. L'exemple suivant expose l'action, la catégorie et la donnée (ici le schéma d'URI pris en compte) :

Code 4-11 : Définir un filtre d'Intents dans le fichier de configuration de l'application

```
<activity ...
   <intent-filter>
      <action
         android:name="android.intent.action.VIEW" />
      <category
         android:name="android.intent.category.DEFAULT" />
      <category
         android:name="android.intent.category.BROWSABLE" />
      <data
         android:scheme="demo" />
   </intent-filter>
</activity>
```

Chaque élément de la balise est important puisqu'il détermine le niveau de filtrage :

- *action :* identifiant unique sous forme de chaîne de caractères. Il est d'usage d'utiliser la convention de nom Java ;
- *category* : premier niveau de filtrage de l'action. Cette balise indique dans quelle circonstance l'action va s'effectuer ou non. Il est possible d'ajouter plusieurs balises de catégorie ;
- *data* : filtre l'objet `Intent` au niveau des données elles-mêmes. Par exemple en jouant avec l'attribut `android:host` on peut répondre à une action comportant un nom de domaine particulier, comme *www.monsite.com*.

Grâce à la définition de ce filtre, l'activité déclarée par le fichier de configuration de l'application (code 4-11) réagira à l'action du code suivant :

Code 4-12 : Exécuter une action grâce au filtre d'Intents

```
Uri uri = Uri.parse(
   "demo://ceci est une chaine de caractère");
Intent intent = new Intent(Intent.ACTION_VIEW, uri);
startActivity(intent);
```

Vous noterez que dans cet exemple, nous n'avons pas spécifié la classe du composant à utiliser pour cette action, ni explicitement quelle activité démarrer. C'est le système qui détermine seul le composant le mieux adapté en interrogeant les filtres définis dans les différents fichiers de configuration des applications installées.

Exploiter l'objet Intent de l'activité

Une fois le filtre d'Intents défini et opérationnel, quelques lignes de code suffiront pour traiter l'Intent transmis. Si le composant n'est pas démarré, le système s'occupera de créer automatiquement une nouvelle instance du composant pour traiter l'intention.

Code 4-13 : Réagir à la réception de l'intention d'un autre composant

```
@Override
public void onCreate(Bundle savedInstanceState) {
   ...
   String data = getIntent().getDataString();
      if(data != null)
         // Ici le code à exécuter
   ...
}
```

Il est possible de récupérer la donnée sous forme d'un objet `Uri` tel qu'il a été envoyé initialement :

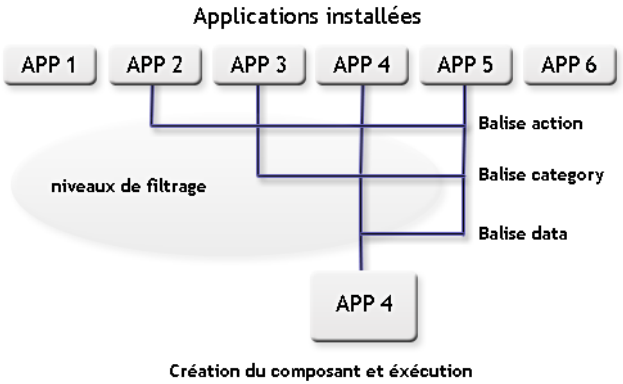
Code 4-14 : Récuperer l'Uri d'un Intent

```
Uri data = getIntent().getData();
   if(data != null)
      // Ici le code à exécuter
```

À RETENIR Remplacer les applications natives d'Android est possible !

Les applications natives utilisent le même mécanisme que les applications tierces. Elles ne sont pas prioritaires ; il est donc possible de les remplacer partiellement ou totalement.

Figure 4–5
Niveaux de filtrage des actions



Aller plus loin avec les Intents

Démarrer un service

Comme pour une activité, pour démarrer un service, utilisez la méthode `startService` d'un objet `Context`, par exemple une activité (la classe `Activity` dérive de ce type). L'unique paramètre de cette méthode est un objet `Intent` représentant l'intention ciblant soit un service précis, comme dans l'exemple suivant, soit une action :

Code 4-15 : Démarrer un service depuis une activité

```
Intent intent = new Intent(this,ServiceADemarrer.class);
startService(intent);
```

Pour plus d'informations sur les services, veuillez consulter le chapitre 11.

Embarquer des données supplémentaires

Lorsque vous demandez à un autre composant de réaliser une action, vous devrez bien souvent spécifier d'autres données supplémentaires. Par exemple, si vous souhaitez ouvrir une activité contenant un navigateur web, il y a de grande chance que cette dernière attende de vous une adresse web.

À cette fin, la classe `Intent` dispose de méthodes `putExtra` et `getExtras` grâce auxquelles un objet de type `Bundle` véhiculera vos données d'une activité à une autre.

L'insertion de données dans ce conteneur se fait au moyen de la méthode `putExtra` et l'extraction au moyen de la méthode `getExtras`.

L'échange des données se fait par l'association d'une clé (représentée par une chaîne de caractères) à une donnée à échanger. Pour ajouter chaque donnée à échanger, il faut appeler la méthode `putExtra` avec la clé et la donnée à échanger. La méthode `putExtra` dispose d'une surcharge pour pratiquement tous les types de données primitifs : `int`, `float`, `String`, etc.

Ainsi, pour définir les données à échanger lors du démarrage d'une activité, la méthode `putExtra` permet d'ajouter une information à l'action qui sera ensuite transmise à l'activité cible.

En reprenant l'exemple précédent (code 4-1), nous pouvons ajouter à l'objet `Intent` une donnée de type `String` (« hello ! », associée à la clé « maclé ») de la manière suivante :

Code 4-16 : Ajouter une donnée à un Intent

```
Intent intent = new Intent(this,MonActivite.class);
intent.putExtra("maclé", "hello !");
startActivity(intent);
```

La méthode `putExtra` prend deux paramètres qui sont les suivants :

- `String name` : la clef unique permettant d'identifier la donnée dans la collection du `Bundle` (ici «maclé ») ;
- `value` : la donnée à échanger.

Une fois l'intention reçue par l'application, via une instance de la classe `Intent`, vous pourrez récupérer les données contenues dans l'objet `Bundle` transmis à l'aide de la méthode `getExtras`. Une fois l'objet `Bundle` récupéré, appelez l'une des méthodes `getXXX` correspondant au type de la donnée recherchée (`getString` pour une donnée de type `String`, `getInt` pour une donnée de type `Int`, etc.) en spécifiant le nom de la clé associée pour récupérer les informations.

Voici comment récupérer la chaîne de caractères (associée à la clé « maclé ») transmise dans l'exemple précédent :

Code 4-17 : Récuperer une donnée d'un *Bundle*

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   Bundle extra = this.getIntent().getExtras();
   if(extra != null)
      String data = extra.getString("maclé");
}
```

Transférer un Intent

Si votre application réceptionne un Intent comportant une action que vous ne traitez pas ou que vous ne souhaitez pas traiter dans un contexte particulier, vous pouvez décider de transmettre celui-ci à une autre activité plus adaptée pour gérer l'action. Pour passer la main à une autre activité, utilisez la méthode `startNextMatchingActivity` de la classe `Activity`.

Code 4-18 : Passer la main à un autre composant plus approprié

```
Intent intent = getIntent();
Uri data = intent.getData();
if(...)
   startNextMatchingActivity(getIntent());
```

Intent en mode déclaratif

Au delà du code, vous aurez peut-être l'occasion de retrouver une référence aux Intents dans d'autres fichiers que ceux de configuration des applications. Bien que cet usage soit plutôt anecdotique, il peut se révéler très efficace pour simplifier vos développements.

C'est le cas par exemple, des écrans de préférences (ce sujet est traité plus en détails dans le chapitre 6 sur les interfaces avancées). Pour rappel, un écran de préférences est un formulaire permettant de sauvegarder très simplement les paramètres de configuration d'une application. Un écran de préférences gère à lui seul le chargement, la modification, l'interaction avec l'utilisateur et la sauvegarde des paramètres de votre application.

Chaque préférence est déclarée au moyen d'une ou de plusieurs balises XML, chaque balise représentant une donnée à enregistrer. Vous allez pouvoir associer votre préférence à une action particulière sans avoir à écrire la moindre ligne de code, simplement en ajoutant une balise XML dans la description du menu. Lorsque l'utilisateur sélectionnera cette préférence, l'Intent avec l'action et les données spécifiées sera émis.

Dans le code 4-19, l'élément `intent` spécifie d'émettre un Intent (avec comme action et données, les valeurs des attributs `action` et `data`) lorsque l'utilisateur sélectionnera le sous-écran de préférences :

Code 4-19 : Déclarer un Intent depuis un PreferenceScreen :

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android" >
   <PreferenceCategory android:title="Exemple">
      <Preference
         android:title="Exemple 1"
```

```
            android:summary="Exemple 1"
            />
        <PreferenceScreen
            android:title="Exemple avec Intent"
            android:summary="Exemple avec Intent">
            <intent
                android:action="android.intent.action.VIEW"
                android:data="demo://ceci est une chaine de caractere"
                />
        </PreferenceScreen>
    </PreferenceCategory>
</PreferenceScreen>
```

Diffuser et recevoir des Intents

La notion de diffusion

Au-delà de demander la réalisation d'une action par une activité précise, l'échange d'Intents forme un moyen de communication évolué capable de gérer des actions au niveau de l'ensemble du système.

Un Intent peut être diffusé à l'ensemble des applications du système pour transmettre des informations à destination des autres applications, que ce soit pour demander la réalisation d'actions spécifiques, comme nous l'avons vu, ou pour fournir des informations sur l'environnement (appel entrant, réseau Wi-Fi connecté, etc.).

Ce mécanisme de diffusion permet à chaque processus de communiquer un état spécifique, une information ou une action de manière homogène à l'ensemble de ses pairs. Dans ce contexte, ces Intents sont alors appelés des *Broadcast Intents* alors que ses récepteurs sont appelés les *Broadcast Receiver ou* « récepteurs d'Intents ».

Android utilise très largement ce type d'Intent pour communiquer des informations système (niveau de la batterie, état du réseau, etc) aux applications. Le système de filtrage quant à lui reste identique mais nous allons voir que la manière de traiter l'information est quelque peu différente. Cela dit, elle ouvre des possibilités intéressantes pour la création de services.

Diffuser des Intents à but informatif

La première étape pour comprendre cette mécanique de diffusion consiste à envoyer un Intent au système grâce à la méthode `sendBroadcast` d'une activité. L'essentiel ici est de bien définir cet objet, sinon vous risquez de parasiter le système ou d'obtenir des comportements non désirés (ouverture d'applications non sollicitées, etc.).

Code 4-20 : Diffuser un Intent

```
    private void doBroadcast(){
        Intent intent = new Intent(MyBroadcastReceiver.VIEW);
        intent.putExtra( "extra", "hello !");
        sendBroadcast(intent);
    }
```

Recevoir et traiter des Intents diffusés

La seconde étape consiste à définir un filtre d'Intents dans le fichier de configuration de l'application puis à écouter le flux de messages grâce à un objet `BroadcastReceiver`. À chaque fois que cet écouteur recevra un message, le système appellera sa méthode `onReceive`.

L'application n'a pas besoin d'être en cours d'exécution pour que le filtre fonctionne. Si l'application n'est pas lancée mais qu'un Intent correspondant au filtre est diffusé, une nouvelle instance de l'application sera automatiquement démarrée par le système.

Code 4-21 : Création d'un *BroadcastReceiver*

```
public final class MyBroadcastReceiver extends BroadcastReceiver {
    public static final String VIEW =
        "eyrolles.intent.action.VIEW";

    @Override
    public void onReceive(Context context, Intent intent) {
        ... // Insérer le code de traitement de l'Intent ici.
    }
}
```

De la même manière que pour les activités, vous devez déclarer un filtre correspondant à l'action définie dans le `BroadcastReceiver` dans le fichier de configuration de l'application :

Code 4-22 : Déclaration du filtre d'Intents pour le BroadcastReceiver

```
<manifest ...>
    <application ...>
    ...
        <receiver android:name="MyBroadcastReceiver">
            <intent-filter>
              <action
                 android:name="eyrolles.intent.action.VIEW" />
              <category
                 android:name="android.intent.category.DEFAULT" />
```

```
        </intent-filter>
      </receiver>
      ...
   </application>
</manifest>
```

Notez la présence de la balise `<receiver>` indiquant que l'application dispose d'un écouteur « permanent ». Permanent car il est également possible d'ajouter et de supprimer un écouteur d'Intents dynamiquement pendant l'exécution d'une application. Cela offre l'opportunité de filtrer à la volée les messages et ainsi d'étendre les capacités de l'application.

Créer un récepteur d'Intents dynamiquement

Pour créer un récepteur d'Intents pendant l'exécution d'une application, créez tout d'abord une classe dérivant de `BroadcastReceiver`. Une fois cette classe créée et instanciée, vous devez encore enregistrer le récepteur auprès du système via la méthode `registerReceiver` de la classe `Activity`.

Cette méthode requiert deux arguments qui reprennent les paramètres de la balise `<receiver>` dans le fichier de configuration de l'application, à savoir spécifier une nouvelle instance de l'objet `BroadcastReceiver` ainsi que le filtre d'Intents associé. Le code suivant récapitule ces étapes :

Code 4-23 : Création d'un récepteur d'Intents dynamique

```
MyBroadcastReceiver receiver;

private void doAddReceiver(){
   receiver = new MyBroadcastReceiver();
   IntentFilter filter =
      new IntentFilter(MyBroadcastReceiver.CALL);
   registerReceiver(receiver, filter);
}
```

Lorsque l'activité a fini son exécution ou bien que le récepteur n'est plus utilisé, vous devez libérer les ressources correspondantes en appelant la méthode `unregisterReceiver` comme indiqué ci-dessous :

Code 4-24 : Libération des ressources prises par un récepteur d'Intents

```
MyBroadcastReceiver receiver;
...
unregisterReceiver(receiver);
```

> **À RETENIR** **Observez le cycle de vie du composant de réception d'intentions**
>
> Tout appel à la méthode `registerReceiver` doit invariablement s'accompagner de l'appel à la méthode `unregisterReceiver`.

Les messages d'informations natifs

Android, à l'instar des actions, propose plusieurs actions prédéfinies et utilisées pour communiquer sur l'état d'un service ou le changement d'un composant du système.

Tableau 4–2 Messages informatifs natifs sous Android

Action	Définition
`ACTION_BOOT_COMPLETED`	Est diffusée lorsque le système a fini de démarrer. La permission `RECEIVE_BOOT_COMPLETED`est requise.
`ACTION_SHUTDOWN`	Indique que le système est en train de s'éteindre. Seul le système peut envoyer ce type de message.
`ACTION_CAMERA_BUTTON`	Survient lorsque le bouton de l'appareil photo est appuyé.
`ACTION_DATE_CHANGED`	Survient lorsque la date a été changée.
`ACTION_TIME_CHANGED`	Survient lorsque l'heure a été changée.
`ACTION_SCREEN_ON / OFF`	Permet d'être informé lorsque que l'écran s'éteint et s'allume.
`ACTION_POWER_CONNECTED / DISCONNECTED`	Diffusée lorsque l'alimentation électrique a été branchée / débranchée.
`ACTION_PACKAGE_ADDED / REMOVED`	Permet de prendre en charge un appel entrant.
`ACTION_MEDIA_MOUNTED / UNMOUNTED`	Survient lorsque qu'un support de stockage, une carte SD par exemple, a été correctement monté ou démonté du système de fichier de l'appareil.

Vous trouverez d'autres messages dans la documentation officielle en ligne du SDK Android.

En résumé

Lancer une activité, naviguer entre écrans... tout ou presque passe par l'envoi d'un objet de la classe Intent au système. Ce mécanisme permet aussi de solliciter d'autres applications pour exécuter une action spécifique (appeler un correspondant, ouvrir une page Web, etc), ou de s'adresser à des composants créés pour recevoir des Intents, appelés récepteurs d'Intents.

La bonne maîtrise des Intents est un passage obligé pour tout développeur qui souhaite développer une application souple, communicante et tirant partie de l'ensemble des composants de la plate-forme Android. Veillez à bien appréhender ce concept qui offre une flexibilité et une puissance sans précédent sur les plates-formes mobiles.

5

Création d'interfaces utilisateur avancées

Android offre des possibilités en ergonomie pour capter l'attention de vos utilisateurs.

Ce chapitre prolonge le chapitre 3 sur les interfaces utilisateur, en montrant comment créer des interfaces beaucoup plus riches, notamment en créant des vues personnalisées. Les adaptateurs permettront la réalisation d'interfaces orientées données en associant une source de données à des contrôles d'interface, par exemple pour créer une liste de contacts ou de pays.

Le système de menu d'Android permet à l'utilisateur d'interagir avec l'application, qu'il s'agisse de naviguer d'une activité à une autre, ou de proposer des actions spécifiques dans des contextes particuliers. Nous verrons en particulier qu'il est aisé d'implémenter un menu global et des menus contextuels.

L'internationalisation des applications est également un point clé de la réussite de vos applications auprès des utilisateurs – tant pour leur diffusion que leur adoption. C'est ce que permet le complément ADT, en facilitant l'adaptation de l'application à la langue de l'utilisateur.

Ce chapitre traitera également de l'animation des vues, pour plus de dynamisme visuel, ainsi que de la création de gadgets.

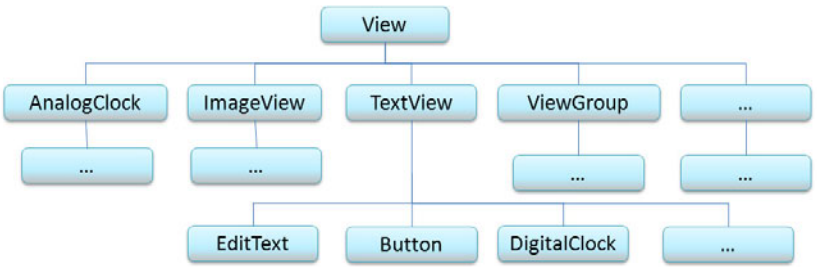
Créer des composants d'interface personnalisés

Si les différentes vues standards ne suffisent plus à réaliser tous vos projets, si vous avez des besoins plus complexes en matière d'interface utilisateur ou avec un style graphique radicalement différent, vous n'aurez d'autre choix que de les créer vous-même.

Les widgets, ou vues standards

Le mot *widget* désigne l'ensemble des vues standards incluses dans la plate-forme Android. Elles font partie du paquetage `android.widget`. Les vues héritant toutes de la classe `View`, chaque widget hérite aussi de cette classe. Ainsi l'élément `Button` hérite-t-il de `TextView`, qui lui-même hérite de classe `View`. L'élément `CheckBox`, quant à lui, hérite de la vue `Button`. Android tire profit des méthodes de chaque vue et de chaque widget pour former une plate-forme paramétrable et modulaire.

Figure 5–1
Structure de l'héritage des contrôles



Voici une liste non exhaustive des contrôles actuellement disponibles :

- `Button` : un simple bouton poussoir ;
- `CheckBox` : contrôle à deux états, coché et décoché ;
- `EditText` : boîte d'édition permettant de saisir du texte ;
- `TextView` : contrôle de base pour afficher un texte ;
- `ListView` : conteneur permettant d'afficher des données sous forme de liste ;
- `ProgressBar` : affiche une barre de progression ou une animation ;
- `RadioButton` : bouton à deux états s'utilisant en groupe.

Vous pouvez décrire une interface de deux façons : soit via une définition XML, soit directement depuis le code en instanciant directement les objets adéquats. La construction d'une interface au sein du code d'une activité, plutôt qu'en utilisant une définition XML, permet par exemple de créer des interfaces dynamiquement.

Code 5-1 : Instanciation d'un Widget au sein d'une activité

```
Button button = new Button(this);
button.setText("Cliquez ici");
button.setOnClickListener(this);
...
```

Il peut arriver qu'une vue ne réponde pas totalement aux besoins du développeur. Dans ce cas il est possible, au même titre que les autres widgets d'hériter d'une vue existante pour former un contrôle personnalisé.

Créer un contrôle personnalisé

Pour créer un contrôle personnalisé, la première étape est de déterminer quelle est la classe la plus adaptée pour former la base de votre contrôle. Dans l'exemple suivant, nous allons créer un contrôle affichant une grille. Nous décidons d'étendre la vue de base `TextView` pour afficher cette grille :

Code 5-2 : Création d'un composant personnalisé : classe CustomView

```
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.util.AttributeSet;
import android.widget.TextView;

public class CustomView
   extends TextView {

   private Paint mPaint;
   private int mColor;
   private int mDivider;

   // Constructeurs
   public CustomView(Context context, AttributeSet attrs, int defStyle) {
      super(context, attrs, defStyle);
      init(context);
   }

   public CustomView(Context context, AttributeSet attrs) {
      super(context, attrs);
      init(context);
   }

   public CustomView(Context context) {
      super(context);
```

```
        init(context);
    }

    // Initialisation du contrôle
    private void init(Context context){
        mDivider = 10;
        mColor = 0xff808080;
        mPaint = new Paint();
        mPaint.setColor(mColor);
    }

    // Écriture de la propriété "GridColor"
    public void setGridColor(int color){
        mColor = color;
        mPaint.setColor(mColor);
        invalidate();
    }

    // Lecture de la propriété "GridColor"
    public int getGridColor(){
        return mColor;
    }

    // Écriture de la propriété "Divider"
    public void setDivider(int divider){
        mDivider = divider;
        invalidate();
    }

    // Lecture de la propriété "Divider"
    public int getDivider(){
        return mDivider;
    }

    @Override
    protected void onDraw(Canvas canvas) {
        // On récupère la taille de la vue
        int x = getWidth();
        int y = getHeight();
        int n = x / mDivider;
        // On dessine des lignes verticales
        for(int i = 0; i<x ; i+=n){
            canvas.drawLine(i, 0, i, y, mPaint);
        }
        n = y / mDivider;
        // On dessine des lignes horizontales
        for(int j = 0; j<y ; j+=n){
            canvas.drawLine(0, j, x, j, mPaint);
        }
```

```
        // On dessine en dessous du texte
        super.onDraw(canvas);
    }
}
```

Vous noterez la déclaration des trois constructeurs de base d'une vue. Ils servent notamment à créer une vue à partir d'un gabarit d'interface au format XML. De cette façon, la création du contrôle et son utilisation dans un fichier de définition XML sont semblables à tout autre contrôle de la plate-forme Android.

Vous remarquerez également que le code 5-2 contient quelques méthodes supplémentaires `set`/`get` permettant respectivement de mettre à jour et de récupérer les propriétés du contrôle. Les deux propriétés ajoutées sont la couleur de grille et le nombre de cellules.

Pour mettre à jour l'affichage du contrôle nous emploierons conjointement les méthodes `invalidate` qui à son tour déclenchera un appel à la méthode `onDraw`. Le système nous fournira un `Canvas` sur lequel nous pourrons dessiner la grille de notre contrôle.

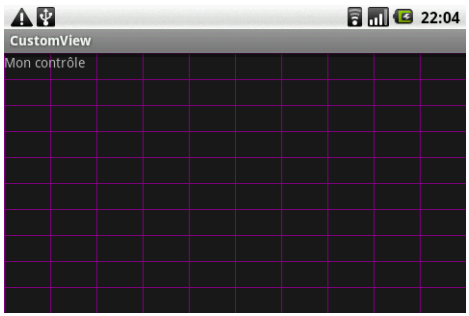
Afin d'éprouver le contrôle personnalisé que vous venez de créer, construisez une activité et définissez le contrôle comme contenu principal. Vous pourrez utiliser l'écran tactile pour changer la taille des cellules de la grille :

Code 5-3 : Utilisation de notre contrôle dans du code

```
import android.app.Activity;
import android.os.Bundle;
import android.view.MotionEvent;

public class MainActivity
   extends Activity
{

   private CustomView view;

   @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        view = new CustomView(this);
        view.setText("Mon contrôle");
        view.setGridColor(0xff800080);
        setContentView(view);
    }

   @Override
   public boolean onTouchEvent(MotionEvent event) {
      switch(event.getAction()){
      case MotionEvent.ACTION_DOWN:
         view.setDivider(20);
         break;
```

```
        case MotionEvent.ACTION_UP:
            view.setDivider(10);
            break;
        }
        return super.onTouchEvent(event);
    }
}
```

Figure 5–2
Rendu de notre contrôle



Déclarer un contrôle personnalisé dans les définitions d'interface XML

Créer une interface utilisateur directement dans le code d'une activité n'est pas une bonne pratique (même si cela peut se révéler nécessaire pour certains scénarios). Le chapitre 2 abordait l'utilisation des contrôles standards d'Android dans les définitions XML des interfaces des activités. Cette méthode a plusieurs avantages : séparation entre la logique fonctionnelle et la présentation, interface modifiable par un graphiste et non uniquement par un développeur, etc.

Contrairement à l'utilisation des contrôles Android standards, l'utilisation d'un contrôle personnalisé dans un fichier de description d'interface XML nécessite un peu de préparation en amont. En effet, pour pouvoir renseigner les propriétés d'un contrôle personnalisé d'une interface de manière déclarative, vous devez tout d'abord exposer ses attributs dans un fichier `res/attrs.xml` :

Code 5-4 : Fichier attrs.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <declare-styleable name="CustomView">
        <attr name="GridColor" format="color"/>
        <attr name="Divider">
            <enum name="big" value="3" />
            <enum name="normal" value="10" />
            <enum name="small" value="20" />
```

```
        </attr>
    </declare-styleable>
</resources>
```

Une fois les propriétés déclarées dans le fichier `res/attrs.xml`, celles-ci seront utilisables dans un fichier XML de description d'interface en ajoutant le nom du paquetage dans l'espace de nom comme indiqué ci-dessous :

Code 5-5 : Fichier main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res/
                ➥ com.eyrolles.android.customview"
     android:orientation="vertical"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     >
    <com.eyrolles.android.customview.CustomView
        android:layout_height="fill_parent"
        android:layout_width="fill_parent"
        android:id="@+id/CustomViewId"
        android:text="Cliquez ici"
        app:GridColor="#ffff0000"
        app:Divider="big"
        />
</LinearLayout>
```

Après avoir défini les propriétés dans un fichier d'attributs et avant de pouvoir profiter pleinement de votre contrôle personnalisé en mode déclaratif, il faut revoir les constructeurs contenus dans le code du contrôle personnalisé. En effet, chaque propriété personnalisée sera passée en paramètre du constructeur sous la forme d'un tableau d'objets. Vous devez donc manipuler ce tableau pour récupérer les valeurs de ces propriétés et les appliquer au contrôle personnalisé :

Code 5-6 : Chargement des propriétés depuis le fichier xml

```
public CustomView(Context context, AttributeSet attrs, int defStyle) {
    super(context, attrs, defStyle);
    mDivider = 10;
    mColor  = 0xff808080;
    mPaint   = new Paint();
    loadAttrFromXml(context, attrs, defStyle);
    mPaint.setColor(mColor);
}
```

```
public CustomView(Context context, AttributeSet attrs) {
   this(context, attrs, 0);
}

public CustomView(Context context) {
   this(context,null);
}

// Charge les propriétés à partir du fichier xml
private void loadAttrFromXml(Context context, AttributeSet attrs, int defStyle){
   TypedArray a = context.obtainStyledAttributes(attrs, R.styleable.CustomView,
                                                 defStyle, 0);
   int n = a.getIndexCount();
   for (int i = 0; i < n; i++) {
      int attr = a.getIndex(i);
      switch (attr) {
      case R.styleable.CustomView_Divider:
         mDivider = a.getInt(attr, mDivider);
         break;
      case R.styleable.CustomView_GridColor:
         mColor = a.getColor(attr, mColor);
         break;
      }
   }
   a.recycle();
}
```

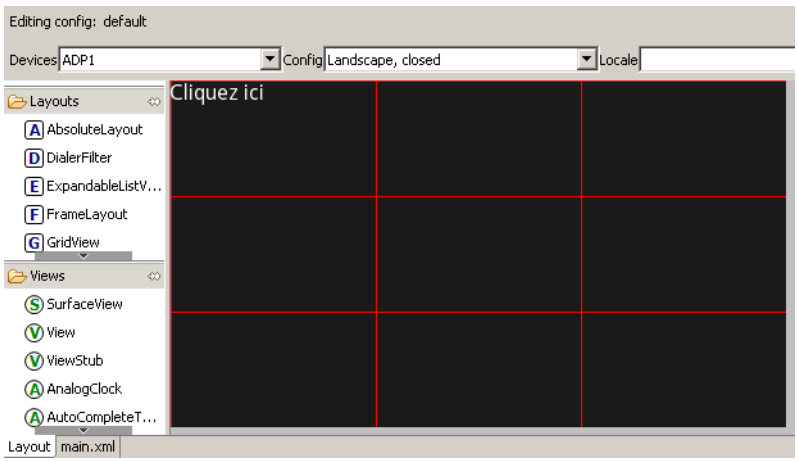


Figure 5–3
Rendu du contrôle dans le concepteur d'interface d'ADT(voir annexes).

La création de contrôles personnalisés est à la portée de tous les développeurs désirant créer des contrôles selon leurs propres besoins. Bien que permettant de créer des applications visuellement très riches, cette manipulation doit faire l'objet d'une réelle

analyse ergonomique de votre part. Un utilisateur ayant l'habitude de manipuler les contrôles standards de la plate-forme Android pourrait être perturbé par l'utilisation de vos contrôles personnalisés et votre application perdrait ainsi tout son attrait.

> **À CONSULTER Les guides d'ergonomie de la documentation Android**
>
> La richesse et la flexibilité de la plate-forme Android vous permettont de créer bon nombre de composants tous visuellement plus réussis les uns que les autres. N'hésitez pas à consulter la documentation du SDK Android, qui évolue chaque jour un peu plus, pour vous donner des directives sur les bonnes pratiques en termes d'ergonomie.
>
> ▸ http://developer.android.com/guide/practices/ui_guidelines/index.html

Les adaptateurs pour accéder aux données de l'interface

Les adaptateurs servent à gérer les sources de données qui seront notamment affichées dans les différentes listes de l'interface utilisateur. Ils réalisent la liaison entre vos sources de données (un simple tableau de chaînes de caractères, une base de données, un fournisseur de contenu, etc.) et les contrôles de votre interface utilisateur.

Techniquement, un adaptateur – représenté par l'objet `Adapter` – n'est ni plus ni moins qu'une interface entre vos données et leur affichage à l'écran. En d'autres termes c'est un tout-en-un vous permettant de séparer vos données brutes sous diverses formes (`array`, `cursor`, `bitmap`, etc.) en éléments visuels manipulables (navigation, sélection, clic). Pour peu que vous ayez déjà réalisé un tel objet, vous connaissez bien la complexité du mécanisme (la création d'un adaptateur personnalisé est traitée dans la suite de ce chapitre). Heureusement pour les autres, Android rend l'utilisation des adaptateurs aussi simple qu'une ligne de commande.

Code 5-7 : Première approche

```
list.setListAdapter(
   new ArrayAdapter<String>(
      context,layout,array));
```

Le rôle de l'adaptateur peut paraître ambigu, à mi-chemin entre un générateur de vue et un agrégateur de données. Mais bien comprendre son fonctionnement permet d'ouvrir de nombreuses perspectives. Pour afficher une liste d'éléments cliquables, nous avons besoin de trois choses :

- des données : `Array`, `ArrayList`, `Cursor`... ;
- un **`Adapter`** qui fera l'interface entre données et les vues ;
- une `ListView` qui fera l'interface entre l'adaptateur et l'utilisateur.

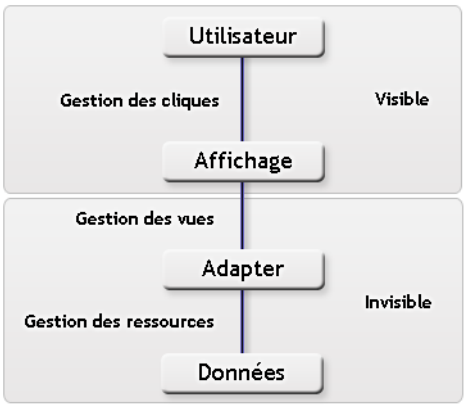


Figure 5–4
L'objet Adapter dans son environnement

Utiliser le bon adaptateur

Puisque `Adapter` est une interface, il n'existe pas d'objet `Adapter` utilisable en l'état. La raison est qu'il pourrait exister autant de type d'adaptateurs qu'il existe de type de données. Bien évidemment, la plate-forme Android contient plusieurs implémentations permettant de traiter les types les plus courants :

- `ArrayAdapter<T>` : pour tous les types de tableaux ;
- `BaseAdapter` : pour créer des adaptateurs personnalisés ;
- `CursorAdapter` : pour traiter les données de type `Cursor` ;
- `HeaderViewListAdapter` : pour ajouter des entêtes et pieds de page aux `ListView` ;
- `ResourceCursorAdapter` : pour la création de vues à partir d'une disposition XML ;
- `SimpleAdapter` : malgré son nom, cet adaptateur s'emploie pour des données complexes ;
- `SimpleCursorAdapter` : sur-couche du `CursorAdapter` permettant de lier un modèle XML aux données.

Prenons comme exemple un tableau de chaînes de caractères à afficher sous la forme d'une liste dans une activité.

Commençons par déclarer le tableau puis un `ArrayAdapter`. Pour simplifier la gestion des vues, nous appliquerons cet adaptateur dans une activité de type `ListActivity`. Ce type d'activité embarque en effet une `ListView` et des méthodes simplifiant la gestion des adaptateurs.

Code 5-8 : Création d'un adaptateur

```
String[] tableau = new String[]{
     "Un" ,"Deux" ,"Trois" ,"Quatre"
   ,"Cinq" ,"Six" ,"Sept" ,"Huit"
   ,"Neuf" ,"Dix"};

ArrayAdapter<String> adapter =
new ArrayAdapter<String>(
   this
   ,android.R.layout.simple_list_item_1, tableau);
this.setListAdapter(adapter);
```

Figure 5–5
Exemple de liste

Le constructeur utilisé pour instancier un nouvel objet `ArrayAdapter` prend trois paramètres :

- `Context context` : grâce à ce paramètre l'adaptateur sera capable de créer seul les objets nécessaires à la transcription des fichiers XML en ressources ;
- `int textViewResourceId` : l'identifiant du fichier XML à utiliser comme modèle pour la liste (en général contenu dans le dossier `/res/layout` du projet). Le modèle est la vue ou le groupe de vue qui sera utilisé par chaque entrée pour s'afficher dans la liste associée avec l'`ArrayAdapter` ;
- `String[] objects` : les données sous forme d'un tableau de chaînes de caractères.

Utiliser une base de données avec le CursorAdapter

Grâce au `CursorAdapter`, Android dispose d'un outil puissant pour gérer de façon simple et efficace des données complexes, qu'il s'agisse d'une liste de contacts, d'un journal d'appels ou bien de tout autre type de données issues d'une base de données.

Prenons comme exemple le journal des appels. Imaginons une liste dépouillée affichant les derniers numéros appelés ainsi que le nom associé à chaque numéro.

Code 5-9 : Création d'un adaptateur SimpleCursorAdapter

```
Cursor cursor = managedQuery(
   Calls.CONTENT_URI
   , null
   , null
   , null
   , null);
SimpleCursorAdapter adapter =
   new SimpleCursorAdapter(
          this
         ,android.R.layout.simple_list_item_2
         ,cursor
         ,new String[]{
              Calls.NUMBER
            , Calls.CACHED_NAME
            }
         ,new int[]{
            android.R.id.text1,
            android.R.id.text2
            }
      );
setListAdapter(adapter);
```

Dans l'exemple ci-dessus on voit clairement l'avantage de ce système. Imaginez un instant la taille et la complexité du code nécessaire à la gestion et l'affichage d'un tel contenu. En seulement quelques lignes de code, nous exécutons une requête pour récupérer la liste des numéros dans le journal des appels, créons un objet `Adapter` grâce aux données et aux dispositions fournies par les ressources de la plate-forme Android et enfin nous associons l'adaptateur à un contrôle `ListView`.

Nous ne rentrerons pas dans le détail pour récupérer les données du journal des appels car ce sujet sera traité dans le chapitre sur la téléphonie. Néanmoins voyons les autres paramètres envoyés au constructeur de l'objet `SimpleCursorAdapter`.

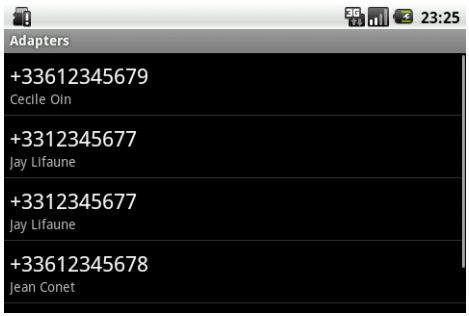


Figure 5–6
Rendu du journal d'appel

Code 5-10 : Détail de la méthode

```
public
   SimpleCursorAdapter(
        android.content.Context context
      , int layout
      , android.database.Cursor c
      , java.lang.String[] from
      , int[] to);
```

Le paramètre `layout` attend un identifiant de fichier XML comportant les éléments à afficher. Dans notre exemple nous avons utilisé `android.R.layout.simple_list_item_2`. Si l'on regarde de plus près le fichier XML, on constate qu'il comporte deux `TextView`, ce qui pourrait se résumer à ceci :

Code 5-11 : Exemple de fichier xml pour définir une vue sur deux lignes

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
      <TextView
android:id="@android:id/text1"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
         />
      <TextView
android:id="@android:id/text2"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
         />
</LinearLayout>
```

Vous noterez les deux identifiants des `TextView@android:id/text1` et `text2`. Ils seront utilisés dans le paramètre `to`, tableau contenant les identifiants des contrôles qui vont recevoir les données. En d'autres termes, nous souhaitons afficher le numéro et le nom du contact, soit deux champs. Nous avons donc une mise en page avec deux `TextView` et deux identifiants :

Code 5-12 : Tableau d'identifiants et déclaration des champs

```
new String[]{
     Calls.NUMBER
   , Calls.CACHED_NAME
}
```

```
new int[]{
     android.R.id.text1
   , android.R.id.text2
}
```

Le `SimpleCusorAdapter` va dans un premier temps charger la mise en page, trouver les contrôles associés aux identifiants puis dans un second temps, placer les données dans chacun des contrôles. Chaque adaptateur fonctionne sur un principe similaire : ils héritent tous de la même classe de base. Ainsi il nous est possible de créer notre propre adaptateur pour gérer un type de données en particulier.

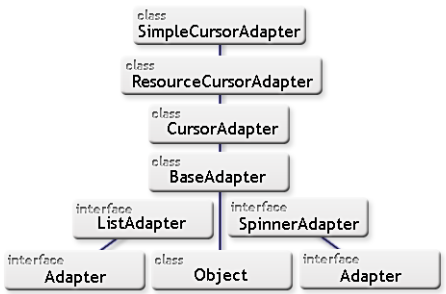


Figure 5–7
Schéma d'héritage du SimpleCursorAdapter

Personnaliser un adaptateur

Après quelques temps d'utilisation du système Android, vous vous rendrez compte que les adaptateurs sont partout. Voici quelques exemples d'applications utilisant des adaptateurs couplés à un objet `ListView` :

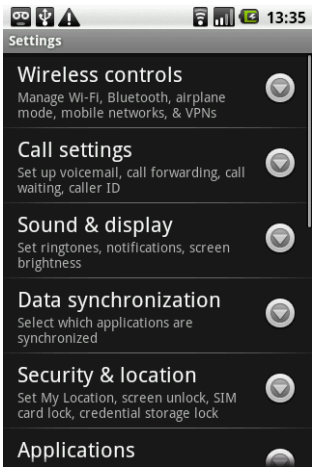


Figure 5–8

Figure 5–9

Figure 5–10

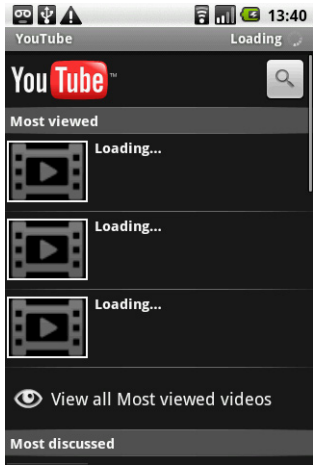


Figure 5–11

La flexibilité et la modularité du modèle objet des adaptateurs permet de bénéficier de la puissance de ceux existants pour former la base d'un nouvel adaptateur.

Tout au long de cette partie nous allons créer un adaptateur permettant de gérer des chaînes de caractères (stockées dans un simple tableau dans cet exemple). Afin de simplifier la conception de cet adaptateur personnalisé, il est important de pouvoir bénéficier des méthodes de base de l'objet `BaseAdapter` pour former la structure du futur adaptateur. Le code suivant commence la création de l'adaptateur personnalisé en créant d'abord une coquille vide :

Code 5-13 : Modèle d'adaptateur personnalisé

```
public class MyCustomeAdapter
   extends BaseAdapter{

   @Override
   public int getCount() {
         return 0;
   }

   @Override
   public Object getItem(int arg0) {
      return null;
   }

   @Override
   public long getItemId(
      int position) {
```

```
      // ...
      return 0;
   }

   @Override
   public View getView(
        int position
      , View convertView
      , ViewGroup parent) {
      // ...
      return null;
   }
}
```

Après avoir implémenté les méthodes requises, vous pouvez ajouter le constructeur qui se composera de deux paramètres essentiels : le contexte et les données.

Code 5-14 : Constructeur du modèle d'adaptateur personnalisé

```
...

LayoutInflater inflater;
String[] data;

...

MyCustomAdapter(Context context, String[] data)
{
   this.data = data;
   this.inflater =
      LayoutInflater.from(context);
}
```

Dans le code précédent, les propriétés sont enregistrées dans des variables locales pour un usage futur. Un nouvel objet `LayoutInflater` est instancié afin de pouvoir créer dynamiquement les vues à afficher dans la liste de destination.

> **MÉTHODE Bien penser son constructeur**
>
> L'efficacité d'un adaptateur n'est pas uniquement déterminée par le code qui le compose. Il faut avant tout réfléchir en amont au type des données qu'il traite et si possible, utiliser un adaptateur déjà existant dans le framework Android.

Une autre étape importante dans la création d'un adaptateur personnalisé est la récupération des données, constituée par l'implémentation des méthodes `getCount`, `getItem` et `getItemId` :

Code 5-15 : Récupérer les informations de l'adaptateur

```
public int getCount() {
   return data.length;
}

public Object getItem(int position) {
   return data[position];
}

public long getItemId(int position) {
   return position;
}
```

Voici le détail de chacune d'elles :

- `getCount` : renvoie le nombre d'éléments contenus dans la collection de données. Dans le cas d'un tableau ce sera la taille de celui-ci ;
- `getItem` : permet d'accéder aux données en indiquant une position absolue dans la collection. Dans le cas d'un tableau, la fonction retourne l'élément contenu à l'index-position donné ;
- `getItemId` : permet de séparer l'ordre naturel des données de leur position dans la liste. Cela permet de réaliser des tris sur les données par la suite (une entrée 'A' peut être placée en première ou dernière position dans la liste selon l'ordre du tri, cependant la méthode `getItemId` retournera toujours la même valeur). Dans le cas d'un tableau non trié, on renverra tout simplement l'index.

La dernière étape pour achever notre adaptateur personnalisé consiste à relier les données aux vues qui afficheront celles-ci. Pour cela, il va falloir identifier les vues qui seront alimentées par les données. Cette association est effectuée via la méthode `getView` :

Code 5-16 : Découverte des vues

```
public View getView(int position, View convertView,
   ViewGroup parent) {

   convertView =
   inflater.inflate(
   android.R.layout.simple_list_item_1, null);

   TextView t1 = (TextView)
   convertView.findViewById(
   android.R.id.text1);

   t1.setText(data[position]);
   return convertView;
}
```

Comme vous pouvez le constater, nous utilisons un objet `LayoutInflater` pour charger le fichier XML grâce à la méthode `inflate`. Puis nous retrouvons une instance d'un `TextView` – ici identifié par son nom `text1` – avec `findViewById`. Une fois l'instance de la boîte de texte récupérée, nous associons la donnée adéquate à son contenu en définissant sa propriété `Text`.

Néanmoins ce système n'est pas très efficace : nous chargeons en effet le fichier XML à chaque passage dans la méthode `getView`. Pour nous éviter ce chargement, l'objet `Adapter` dispose d'un mécanisme de cache permettant de limiter le nombre d'appels au `LayoutInflater`.

Optimiser l'adaptateur en utilisant le cache

Comme nous venons de le voir, nous avons besoin dans certains cas de conserver le contenu d'un fichier XML et nous allons pour cela utiliser le cache de l'adaptateur.

À l'origine de cette optimisation nous retrouvons la méthode `view.setTag`. Cette fonction permet de stocker des données dans une vue. Nous allons utiliser ce principe avec l'adaptateur et ainsi conserver une référence aux contrôles et économiser des ressources précieuses :

Code 5-17 : La méthode getView optimisée

```
// Nous créons une structure qui encapsulera les données
private class MyTag {
   TextView txtData;
   TextView txtExtra;
}

public View getView(int position, View convertView, ViewGroup parent)
   {
   MyTag holder; // Instanciation de notre enveloppe de données
   if (convertView == null){
      convertView = inflater.inflate(android.R.layout.simple_list_item_2, null);
      holder = new MyTag();
      // Nous remplissons l'enveloppe de données
      holder.txtData = (TextView)convertView.findViewById(android.R.id.text1);
      holder.txtExtra = (TextView)convertView.findViewById(android.R.id.text2);
      // Nous spécifions nos données au contrôle personnalisé
      convertView.setTag(holder);
   } else {
      holder = (MyTag)convertView.getTag();
   }
   holder.txtData.setText(data[position]);
   holder.txtExtra.setText(""+System.currentTimeMillis());
   return convertView;
}
```

Concernant les adaptateurs, rappelez vous qu'il n'est que rarement nécessaire d'en recréer un dans son intégralité. La plate-forme Android dispose bien souvent de l'adaptateur dont vous aurez besoin.

Créer un menu pour une activité

Tous les modèles Android possèdent un bouton *Menu*. Grâce à ce bouton, il vous est possible de proposer à vos utilisateurs des fonctionnalités supplémentaires n'apparaissant pas par défaut à l'écran et d'ainsi mieux gérer la taille limitée de l'écran d'un appareil mobile.

Chaque menu est propre à une activité, c'est pourquoi toutes les opérations que nous allons traiter dans cette partie se réfèrent à une activité. Pour proposer plusieurs menus vous aurez donc besoin de le faire dans chaque activité de votre application.

Création d'un menu

Pour créer un menu il vous suffit de surcharger la méthode `onCreateOptionsMenu` de la classe `Activity`. Cette méthode est appelée la première fois que l'utilisateur appuie sur le bouton menu de son téléphone. Elle reçoit en paramètre un objet de type `Menu` dans lequel nous ajouterons nos éléments ultérieurement.

Si l'utilisateur appuie une seconde fois sur le bouton *Menu*, cette méthode ne sera plus appelée tant que l'activité ne sera pas détruite puis recréée. Si vous avez besoin d'ajouter, de supprimer ou de modifier un élément du menu après coup, il vous faudra surcharger une autre méthode, la méthode `onPrepareOptionsMenu` que nous aborderons plus tard dans cette partie.

Pour créer un menu, commencez par créer un fichier de définition d'interface nommé `main.xml`:

Code 5-18 : Définition de l'interface de l'exemple avec menu

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content">
   <TextView
      android:id="@+id/TextView01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Appuyez sur la touche menu">
   </TextView>
</LinearLayout>
```

Créez une nouvelle classe d'activité, dans un fichier `main.java` par exemple, dans lequel vous placerez le contenu suivant :

Code 5-19 : Création d'un menu

```
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;

public class Main extends Activity {

   // Pour faciliter la gestion des menus
   // nous créons des constantes
   private final static int MENU_PARAMETRE = 1;
   private final static int MENU_QUITTER = 2;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
   }

   @Override
   public boolean onCreateOptionsMenu(Menu menu) {
      // Nous créons un premier menu pour accéder aux paramètres.
      // Pour ce premier élément du menu, nous l'agrémentons
      // d'une image.
      menu.add(0, MENU_PARAMETRE, Menu.NONE, "Paramètres").setIcon(R.drawable.icon);
      // Nous créons un second élément.
      // Celui-ci ne comportera que du texte.
      menu.add(0, MENU_QUITTER, Menu.NONE, "Quitter");

      return true;
   }

   @Override
   public boolean onOptionsItemSelected(MenuItem item) {
      switch (item.getItemId()) {
      case MENU_PARAMETRE:
         // Lorsque l'utilisateur cliquera sur le menu 'paramètres',
         // un message s'affichera.
         Toast.makeText(Main.this, "Ouverture des paramètres",
                        Toast.LENGTH_SHORT).show();
         // onOptionsItemSelected retourne un booléen,
         // nous lui envoyons la valeur "true" pour signaler
         // que tout s'est correctement déroulé.
         return true;
```

```
        case MENU_QUITTER:
           // Lorsque l'utilisateur cliquera sur le menu 'Quitter',
           // nous fermerons l'activité avec la méthode finish().
           finish();
           return true;
        default:
           return true;
        }
     }
}
```

Dans le code précédent, nous avons surchargé deux méthodes, l'une responsable de la création du menu de l'activité, l'autre de la logique lors de la sélection par l'utilisateur d'un élément du menu.

La méthode `onCreateOptionsMenu` sera appelée uniquement la première fois lorsque l'utilisateur appuiera sur le bouton *Menu*. Dans cette méthode, nous créons deux éléments de menu *Paramètres* et *Quitter*.

La méthode `onOptionsItemSelected` est appelée lorsque l'utilisateur clique sur l'un des éléments du menu. Dans notre cas, si l'utilisateur clique sur l'élément *Paramètres* l'application affiche un message, alors qu'un clic sur *Quitter* ferme l'activité grâce à la méthode `finish`.

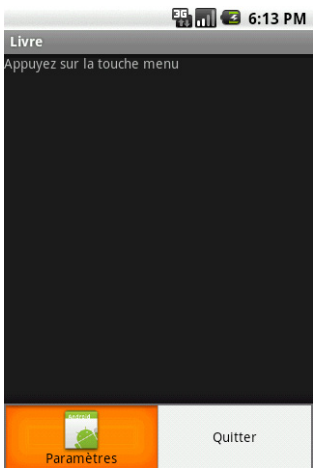


Figure 5–12
Exécution du code 5-19 : création d'un menu

Mettre à jour dynamiquement un menu

Il vous sera peut-être nécessaire de changer l'intitulé, de supprimer ou de rajouter un élément du menu en cours de fonctionnement. La méthode `onCreateOptionsMenu` n'étant appelée qu'une fois, la première fois où l'utilisateur clique sur le bouton *Menu*, vous ne pourrez pas mettre à jour votre menu dans cette méthode.

Pour modifier le menu après coup, par exemple dans le cas où votre menu propose de s'authentifier et qu'une fois authentifié, vous souhaitez proposer à l'utilisateur de se déconnecter, vous devrez surcharger la méthode `onPrepareOptionsMenu`.

Un objet de type `Menu` est envoyé à cette méthode qui est appelée à chaque fois que l'utilisateur cliquera sur le bouton *Menu*. Modifiez le code 5-19 pour rajouter la redéfinition de la méthode `onPrepareOptionsMenu` :

Code 5-20 : Modification dynamique du menu

```
@Override
public boolean onPrepareOptionsMenu(Menu menu) {
   // Nous modifions l'intitulé de notre premier menu
   // Pour l'exemple nous lui donnerons un intitulé
   // différent à chaque fois que l'utilisateur cliquera
   // sur le bouton menu à l'aide de l'heure du système
   menu.getItem(0).setTitle(SystemClock.elapsedRealtime()+"");

   return super.onPrepareOptionsMenu(menu);
}
```

Dans cet exemple, lorsque l'utilisateur appuie sur le bouton *Menu*, l'intitulé du premier menu afficher une valeur différente à chaque fois. Vous pourrez placer à cet endroit le code nécessaire pour adapter l'intitulé que vous souhaitez afficher à l'utilisateur s'il est nécessaire de changer le menu. Si l'utilisateur appuie à nouveau, l'intitulé changera de concert. S'il rappuie encore, l'intitulé changera de même.

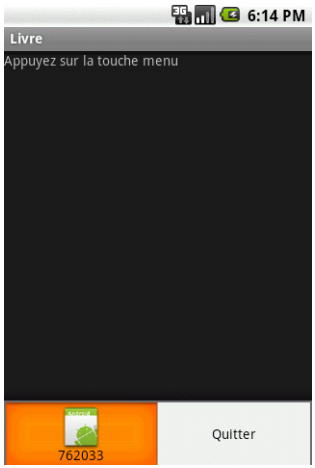


Figure 5–13
Un bouton mis à jour une première fois avec la méthode onPrepareOptionsMenu

De la même façon, vous pouvez changer l'image du menu comme ceci :

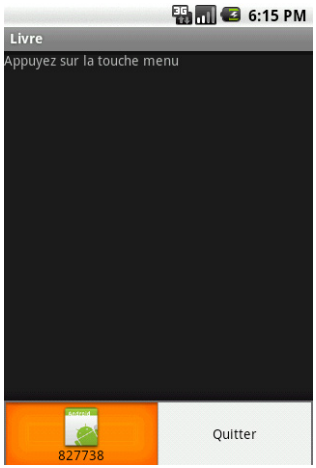


Figure 5–14
Un bouton mis à jour une seconde fois avec la méthode onPrepareOptionsMenu

Code 5-21 : Changer l'image d'un élément de menu

```
menu.getItem(0).setIcon(android.R.drawable.icon_secure);
```

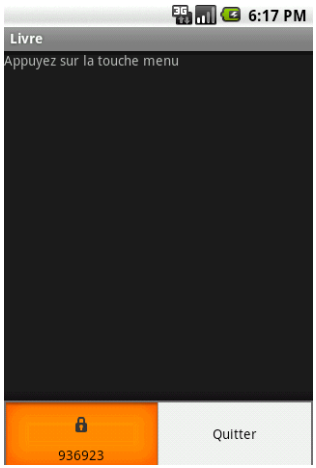


Figure 5–15
L'icône d'un élément du menu changée dynamiquement

Créer des sous-menus

Céer des sous-menus peut se révéler utile pour proposer plus d'options sans encombrer l'interface utilisateur.

Pour ajouter un sous-menu, vous devrez ajouter un menu de type `SubMenu` et des éléments le composant. Le paramétrage est le même que pour un menu, à l'exception de l'image : vous ne pourrez pas ajouter d'image sur les éléments de vos sous-menus.

Néanmoins, il sera possible d'ajouter une image dans l'entête du sous-menu. Remplacez la méthode `onCreateOptionsMenu` du code 5-19 par celle-ci :

Code 5-22 : Création de sous-menus

```
private final static int SOUSMENU_VIDEO= 1000;
private final static int SOUSMENU_AUDIO= 1001;

...

@Override
public boolean onCreateOptionsMenu(Menu menu) {
   // Nous créons un premier menu sous forme de sous-menu
   // Les sous menu associés à ce menu seront ajoutés à ce sous-menu
   SubMenu sousMenu = menu.addSubMenu(0, MENU_PARAMETRE, Menu.NONE,
                                      "Paramètres").setIcon(R.drawable.icon);
   // Nous ajoutons notre premier sous-menu
   sousMenu.add(0, SOUSMENU_AUDIO, Menu.NONE, "Audio").setIcon(R.drawable.icon);
   // Nous ajoutons notre deuxième sous-menu
   sousMenu.add(0, SOUSMENU_VIDEO, Menu.NONE, "Vidéo");

   // Il est possible de placer une
   // image dans l'entête du sous-menu
   // Il suffit de le paramétrer avec la méthode
   // setHeaderIcon de notre objet SubMenu
   sousMenu.setHeaderIcon(R.drawable.icon);

   // Nous créons notre deuxième menu
   menu.add(0, MENU_QUITTER, Menu.NONE, "Quitter");

   return true;
}
```

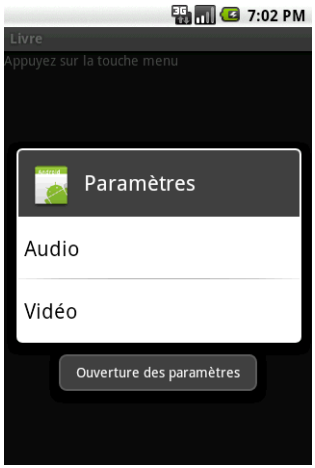


Figure 5–16
Utilisation d'un sous menu

Lorsque l'utilisateur clique sur le bouton *Menu*, le menu qui s'affiche est identique à l'exemple précédent. Néanmoins en cliquant sur le menu *Paramètres* un sous-menu s'affiche sous forme de liste.

Créer un menu contextuel

La plate-forme Android autorise également la création des menus contextuels, autrement dit de menus dont le contenu change en fonction du contexte. Les menus contextuels fonctionnent d'ailleurs de façon similaire aux sous-menus.

> **B.A.-BA Appeler un menu contextuel dans Android**
>
> Rappelons que sous Android, l'appel d'un menu contextuel se fait lorsque l'utilisateur effectue un clic de quelques secondes sur un élément d'interface graphique.

La création d'un menu contextuel se fait en deux étapes : tout d'abord la création du menu proprement dit, puis son enregistrement auprès de la vue, avec la méthode `registerForContextMenu` ; en fournissant en paramètre la vue concernée.

Concrètement, redéfinissez la méthode `onCreateContextMenu` et ajoutez-y les menus à afficher en fonction de la vue sélectionnée par l'utilisateur. Pour sélectionner les éléments qui s'y trouveront, redéfinissez la méthode `onContextItemSelected`.

Créez un nouveau fichier de définition d'une interface comme ceci :

Code 5-23 : Définition de l'interface de l'exemple avec menu contextuel

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical"
   android:gravity="center_vertical|center_horizontal">
   <TextView
      android:id="@+id/monTexteContextMenu"
      android:layout_height="wrap_content"
      android:text="Cliquez ici 3 sec"
      android:textSize="30dip"
      android:layout_width="fill_parent"
      android:gravity="center_horizontal">
   </TextView>
</LinearLayout>
```

Effectuez les modifications du code 5-19 suivantes :

Code 5-24 : Création d'un menu contextuel

```
...
import android.view.ContextMenu;
import android.view.MenuItem;
import android.view.ContextMenu.ContextMenuInfo;
...

   // Pour faciliter notre gestion des menus
   // nous créons des variables de type int
   // avec des noms explicites qui nous aideront
   // à ne pas nous tromper de menu
   private final static int MENU_CONTEXT_1= 1;
   private final static int MENU_CONTEXT_2= 2;


   @Override
   public void onCreate(Bundle savedInstanceState) {
      ...

      // Nous enregistrons notre TextView pour qu'il réagisse
      // au menu contextuel
      registerForContextMenu((TextView)findViewById(R.id.monTexteContextMenu));
   }


   @Override
   public void onCreateContextMenu(ContextMenu menu, View v,
         ContextMenuInfo menuInfo) {
      // Nous vérifions l'identifiant de la vue.
      // Si celle-ci correspond à une vue pour laquelle nous souhaitons
      // réagir au clic long, alors nous créons un menu.
      // Si vous avez plusieurs vues à traiter dans cette méthode,
      // il vous suffit d'ajouter le code nécessaire pour créer les
      // différents menus.
      switch (v.getId()) {
      case R.id.monTexteContextMenu:
         {
               menu.setHeaderTitle("Menu contextuel");
               menu.setHeaderIcon(R.drawable.icon);
               menu.add(0, MENU_CONTEXT_1, 0, "Mon menu contextuel 1");
               menu.add(0, MENU_CONTEXT_2, 0, "Mon menu contextuel 2");
               break;
           }
      }
      super.onCreateContextMenu(menu, v, menuInfo);
   }
```

```
    @Override
    public boolean onContextItemSelected(MenuItem item) {
       // Grâce à l'identifiant de l'élément
       // sélectionné dans notre menu nous
       // effectuons une action adaptée à ce choix
       switch (item.getItemId()) {
       case MENU_CONTEXT_1:
          {
                Toast.makeText(Main.this, "Menu contextuel 1 cliqué !",
                               Toast.LENGTH_SHORT).show();
                break;
          }
       case MENU_CONTEXT_2:
          {
                Toast.makeText(Main.this, "Menu contextuel 2 cliqué !",
                               Toast.LENGTH_SHORT).show();
                break;
          }
       }
       return super.onContextItemSelected(item);
    }
}
```

Figure 5–17
Une vue possédant un menu contextuel

Si l'utilisateur clique pendant trois secondes sur la vue `TextView` centrale, le menu contextuel s'affiche.

Figure 5–18
Affichage du menu contextuel de l'exemple 5-24

Pour insérer une image dans le gabarit de mise en page de l'interface, ajoutez les lignes suivantes dans le fichier XML après la balise `<TextView>` :

Code 5-25 : Ajout d'une image dans le menu contextuel de l'exemple

```
...
</TextView>
<ImageView
   android:id="@+id/ImageView01"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:src="@drawable/icon">
</ImageView>
...
```

À ce stade, si l'utilisateur clique quelques secondes sur l'image, il ne se produira rien car la vue de l'image n'a pas enregistré de menu contextuel.

Vous devez ajouter la ligne suivante dans la méthode `onCreate` de votre fichier `main.java` :

Code 5-26 : Enregistrer le contrôle de l'image pour réagir au menu contextuel

```
registerForContextMenu((TextView)findViewById(R.id.monTexteContextMenu));
...
registerForContextMenu((ImageView)findViewById(R.id.monImageContext));
```

L'image est désormais enregistrée et un clic dessus déclenchera l'apparition du menu contextuel. Il reste à adapter les méthodes `onCreateContextMenu` et

onContextItemSelected pour que les menus contextuels destinés à cette image s'affichent correctement :

Code 5-27 : Modification du code 5-24 pour gérer le menu contextuel de l'image

```
…

private final static int MENU_CONTEXT_IMAGE_1= 3;
private final static int MENU_CONTEXT_IMAGE_2= 4;

@Override
public void onCreateContextMenu(ContextMenu menu, View v,
      ContextMenuInfo menuInfo) {
   // Nous vérifions l'identifiant de la vue envoyée.
   // Si celle-ci correspond à celle que nous souhaitons,
   // nous créons nos menus.
   // Si vous avez plusieurs vues à faire réagir, il vous suffira
   // d'ajouter les conditions et le code nécessaire
   // pour afficher ces même menus ou d'autre menus.
   switch (v.getId()) {
   case R.id.monTexteContextMenu:
      {
         menu.setHeaderTitle("Menu contextuel");
         menu.setHeaderIcon(R.drawable.icon);
         menu.add(0, MENU_CONTEXT_1, 0, "Mon menu contextuel 1");
         menu.add(0, MENU_CONTEXT_2, 0, "Mon menu contextuel 2");
         break;
      }
   case R.id.monImageContext:
      {
         menu.setHeaderTitle("Menu contextuel Image");
         menu.setHeaderIcon(R.drawable.icon);
         menu.add(0, MENU_CONTEXT_IMAGE_1, 0, "Mon menu contextuel IMAGE 1");
         menu.add(0, MENU_CONTEXT_IMAGE_2, 0, "Mon menu contextuel IMAGE 2");
         break;
      }
   }
   super.onCreateContextMenu(menu, v, menuInfo);
}

@Override
public boolean onContextItemSelected(MenuItem item) {
   // Grâce à l'identifiant de l'élément
   // sélectionné dans notre menu nous
   // effectuons une action adaptée à ce choix
   switch (item.getItemId()) {
   case MENU_CONTEXT_1:
      {
```

```
            Toast.makeText(Main.this, "Menu contextuel 1 cliqué !",
                           Toast.LENGTH_SHORT).show();
            break;
         }
      case MENU_CONTEXT_2:
         {
            Toast.makeText(Main.this, "Menu contextuel 2 cliqué !",
                           Toast.LENGTH_SHORT).show();
            break;
         }
      case MENU_CONTEXT_IMAGE_1:
         {
            Toast.makeText(Main.this, "Menu contextuel IMAGE 1 cliqué !",
                           Toast.LENGTH_SHORT).show();
            break;
         }
      case MENU_CONTEXT_IMAGE_2:
         {
            Toast.makeText(Main.this, "Menu contextuel IMAGE 2 cliqué !",
                          Toast.LENGTH_SHORT).show();
            break;
         }
      }
      return super.onContextItemSelected(item);
}

...
```

Désormais, en cliquant quelques secondes sur l'image, un menu contextuel dédié s'affichera et les clics de l'utilisateur sur les éléments de ce menu seront gérés.

Internationalisation des applications

Pour que vos applications rencontrent le plus grand succès, il faut les rendre compréhensibles par le plus grand nombre d'utilisateurs, et donc les traduire. Les auteurs de la plate-forme Android ont bien pris en compte cette nécessité et offrent un mécanisme assez facile à prendre en main.

BON À SAVOIR **i18n et l10n**

On voit de temps en temps ces deux termes quand on parle de l'internationalisation et de la régionalisation des applications ou des systèmes. En fait, rien de sorcier derrière ces termes : il y a 18 lettres entre le *i* et le *n* d'internationalisation et 10 entre le *l* et le *n* de localisation (le mot anglais pour régionalisation). C'est plus court à écrire !

Étape 0 : créer une application à manipuler

Créons tout d'abord une application standard sans penser à une éventuelle internationalisation. Par contre intégrons deux contraintes : des chaînes de caractères et des images affichées.

Code 5-28 : Mise en page d'une application avec une image et une case à cocher

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout (1) xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<ImageButton (2)
    android:id="@+id/android_button"
    android:layout_width="100dip"
    android:layout_height="wrap_content"
    android:src="@drawable/android" />

<CheckBox android:id="@+id/checkbox" (3)
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/checkboxtext" />
</LinearLayout>
```

Cette mise en page est simple avec une disposition linéaire (1), une image du bonhomme Android (2) et une case à cocher (3) contenant un texte à traduire.

Il faut également renseigner le fichier ressource qui contient les chaînes de caractères.

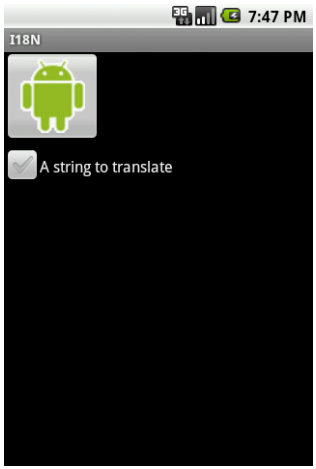


Figure 5–19
Capture d'écran de l'application sans traduction

Code 5-29 : Fichier strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">I18N</string>
    <string name="checkboxtext">A string to translate</string>
</resources>
```

Internationaliser des chaînes de caractères

Maintenant, commençons à personnaliser l'application en fonction de la langue de l'appareil. Cliquez sur l'icône de l'assistant de création de fichiers XML ou parcourez les menus *File > New > Android XML File*.

Figure 5–20
Icône de l'assistant de création de fichiers XML

Grâce au formulaire qui apparaît, sélectionnez votre projet puis entrez un nom de fichier à traduire (`strings.xml` si vous n'avez pas modifié le projet). Spécifiez ensuite *Values* comme type de ressource à créer. Sélectionnez *Language* dans la liste gauche et appuyez sur la flèche. Ensuite dans la boîte *Language* qui doit apparaître, à droite de la fenêtre, saisissez le code à deux lettres qui correspond au langage que vous souhaitez utiliser. Prenons *fr* comme exemple.

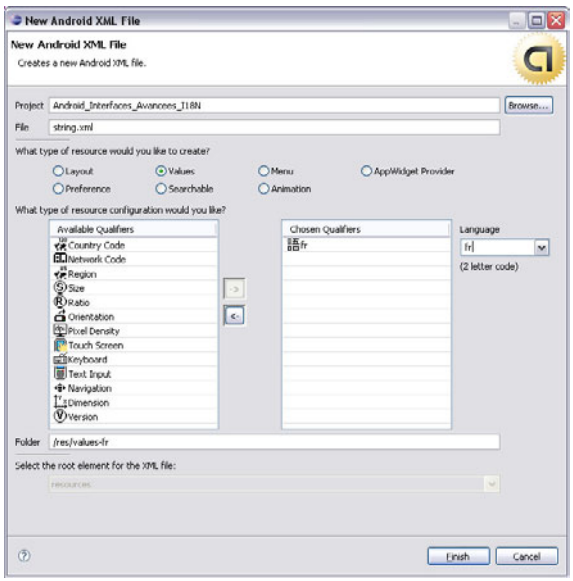


Figure 5–21
Ajout d'un fichier de traduction en français

> **À CONSULTER Liste des langues et des pays**
>
> Vous pouvez consulter la liste des codes de langues et de pays aux adresses suivantes :
> - http://www.loc.gov/standards/iso639-2/php/code_list.php
> - http://www.iso.org/iso/country_codes/iso_3166_code_lists/english_country_names_and_code_elements.htm

Enfin, appuyez sur *Finish*. Votre projet devrait contenir un répertoire supplémentaire : `/res/values-fr`.

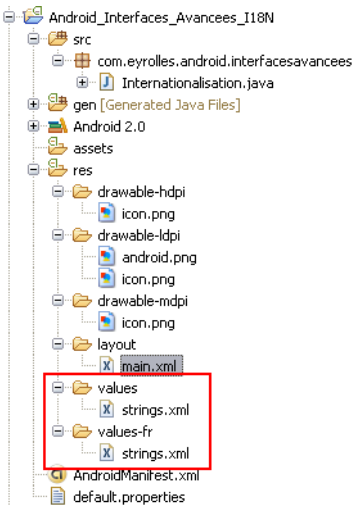


Figure 5–22
Impact sur le projet

Éditez le fichier `strings.xml` de ce nouveau répertoire et placez-y les chaînes traduites en français.

Code 5-30 : Chaînes traduites en français dans le répertoire res/values-fr

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">I18N France</string>
    <string name="checkboxtext">Une traduction pour la France</string>
</resources>
```

Ces modifications effectuées, relancez votre émulateur et vous devriez constater... que rien n'a changé ! En effet, cela est tout à fait normal étant donné que nous n'avons pas modifié la langue du système qui n'est pas le français par défaut.

Pour changer ce paramètre, ouvrez le volet des applications et sélectionnez *Custom Locale*. Le réglage affiché doit être *en_US*, sélectionnez *fr_FR* grâce à un appui long et à la validation demandée.

Figure 5–23
Réglage de la langue du système

Relancez l'application, elle est bien en français !

Voici un tableau qui met en évidence la logique entre le code pays et les fichiers et répertoires nécessaires. Tous les codes pays possibles sont consultables grâce à l'application *Custom Locale*.

Tableau 5–1 Les différents pays avec leurs codes et l'emplacement de leurs ressources

Code pays	Pays/Langue concernés	Répertoire du fichier string.xml
Défaut	Angleterre/Anglais	`res/values/`
en-rUS	Etats-Unis/Anglais	`res/values/`
fr-rFR	France/Français	`res/values-fr/`
Ja-rJP	Japon/Japonnais	`res/values-ja/`
...	...	...

Internationaliser les images

Comme nous l'avons laissé entendre tout au long du chapitre, il est également possible de choisir des images différentes en fonction de la langue du système (utile par exemple si l'on veut afficher le drapeau de la langue courante).

Pour cela, créons une seconde image de la mascotte Android francisée, par exemple en lui rajoutant quelques couleurs. Il ne reste qu'à la placer au bon endroit. La logique est d'ailleurs similaire à celle qui dicte la gestion des chaînes de caractères.

Tableau 5–2 Les différents pays avec leurs codes et l'emplacement de leurs ressources images

Code pays	Pays/Langue concernés	Répertoire des images
Défaut	Angleterre/Anglais	`res/drawable/`
en-rUS	Etats-Unis/Anglais	`res/drawable-en-rUS/`
fr-rFR	France/Français	`res/drawable-fr-rFR/`
Ja-rJP	Japon/Japonnais	`res/drawable-ja-rJP/`
...	...	...

En consultant rapidement le tableau présenté ci-dessus, il faut placer cette nouvelle image – mais avec le même nom de fichier `android.png` – dans le répertoire `res/drawable-fr-FR`.

Relancez l'application et voilà, le tour est joué !

Figure 5–24
Résultat de notre internationalisation

Animation des vues

Les animations font aujourd'hui partie de notre quotidien que ce soit sur le Web (via les technologies XHTML/CSS/JavaScript, Adobe Flash/Flex, Microsoft Silverlight, etc.) ou dans les applications bureautiques. Face à cela, la plate-forme Android n'est pas en reste puisqu'elle propose deux mécanismes pour animer vos interfaces :

- les animations image par image (dédiées comme son nom l'indique, aux images) ;
- les animations de positionnement (dédiées à l'animation des vues).

Cette partie présente ces différentes méthodes.

La plate-forme Android permet là encore de définir des animations des deux façons habituelles : par le code ou via l'utilisation d'un fichier XML. Définir les animations dans des fichiers de ressources externes offre deux atouts : d'une part, une personne de l'art tel qu'un designer peut modifier le fichier indépendamment du code et d'autre part, Android peut alors appliquer automatiquement l'animation la plus appropriée selon le matériel (taille et orientation de l'écran).

Les animations d'interpolation

Les animations d'interpolation permettent d'effectuer une rotation et/ou de changer la position, la taille, l'opacité d'une vue. Avec ce type d'animation, la plate-forme s'occupera pour vous de définir les étapes intermédiaires par interpolation en fonction du temps et des états d'origine et de fin que vous aurez spécifiés.

Ce type d'animation est souvent utilisé pour réaliser des transitions entre activités ou pour mettre en exergue un élément de l'interface à l'utilisateur (saisie incorrecte, options à la disposition de l'utilisateur, etc).

Les interpolations possibles sont les suivantes :

- opacité (*Alpha*) : permet de jouer sur la transparence/opacité de la vue ;
- échelle (*Scale*) : permet de spécifier l'agrandissement/réduction sur les axes et X et Y à appliquer à la vue ;
- translation (*Translate*) : permet de spécifier la translation/déplacement à effectuer par la vue ;
- rotation (*Rotate*) : permet d'effectuer une rotation selon un angle en degré et un point de pivot.

Animer par le code

Chaque type d'animation par interpolation possède une sous-classe dérivant de `Animation` : `ScaleAnimation`, `AlphaAnimation`, `TranslateAnimation` et `RotateAnimation`. Chacune de ces animations peut être créée, paramétrée, associée et exécutée directement depuis le code :

Code 5-31 : Animer une vue directement depuis le code

```
TextView vueTexte = ...;

// Animation d'une alpha
Animation animationAlpha = new AlphaAnimation(0.0f, 1.0f);
animationAlpha.setDuration(100);
vueText.startAnimation(animationAlpha);
```

```
// Animation de translation d'une vue
TranslateAnimation animationTranslation = new TranslateAnimation(
   Animation.RELATIVE_TO_SELF, 0.0f, Animation.RELATIVE_TO_SELF, 0.0f,
Animation.RELATIVE_TO_SELF, -1.0f, Animation.RELATIVE_TO_SELF, 0.0f);
animationTranslation.setDuration(500);
vueText.startAnimation(animationTranslation);
```

Chaque type d'animation possède des constructeurs propres prenant des paramètres qui leur sont spécifiques. D'une manière globale, la plupart des méthodes sont communes. Par exemple, pour spécifier la durée de l'animation, utilisez la méthode `setDuration` spécifiant le nombre de millisecondes en paramètre.

Enfin, pour démarrer une animation, appelez la méthode `startAnimation` de la vue cible et spécifiez l'objet `Animation` que vous souhaitez utiliser.

Nous verrons plus loin comment créer une série d'animation par le code, c'est-à-dire un enchaînement d'animations élémentaires.

Animer par le XML

Le tableau suivant présente les attributs XML des paramètres de chaque type d'animation (vous remarquerez que ces propriétés XML retrouvent toutes un équivalent sous la forme de méthodes d'accès `get`/`set` dans les classes dérivées d'`Animation`).

Tableau 5–3 Les propriétés des animations par interpolation

Type d'animation	Propriété	Description
ScaleAnimation	`fromXScale`	`float`
	`toXScale`	`float`
	`fromYScale`	`float`
	`toYScale`	`float`
	`pivotX` / `pivotY`	Valeur en pourcentage du centre de rotation, ou pivot, relatif à la vue sur l'axe des ordonnées. Chaîne dont la valeur est comprise entre '0 %' et '100 %'.
AlphaAnimation	`fromAlpha`	Valeur de l'opacité de départ. Valeur comprise entre 0.0 et 1.0 (0.0 est transparent).
	`toAlpha`	Valeur de l'opacité d'arrivée. Valeur comprise entre 0.0 et 1.0 (0.0 est transparent).
TranslateAnimation	`fromX`	`float`
	`toX`	`float`

Tableau 5–3 Les propriétés des animations par interpolation (suite)

Type d'animation	Propriété	Description
RotateAnimation	`fromDegrees`	Orientation de départ en degrés. Valeur comprise entre 0 et 360.
	`toDegrees`	Orientation d'arrivée en degrés. Valeur comprise entre 0 et 360.
	`PivotX / pivotY`	Valeur en pourcentage du centre de rotation, ou pivot, relatif à la vue sur l'axe des abscisses. Chaine dont la valeur est comprise entre '0 %' et '100 %'.

Les fichiers XML de description d'animation doivent être placés dans le répertoire `res/anim` afin que le complément ADT génère un identifiant qui sera ensuite utilisé pour y faire référence dans le code.

L'extrait de fichier XML suivant présente la définition d'animations de chaque type. Un certain nombre d'éléments et d'attributs seront expliqués dans la partie suivante :

Code 5-32 : Ensemble d'animations

```
// Une animation sur l'opacité de la vue.
<alpha
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:interpolator="@android:anim/accelerate_interpolator"
   android:fromAlpha="0.0"
   android:toAlpha="1.0"
   android:duration="1000" />

// Animation de translation
<translate android:fromXDelta="100%p"
   android:toXDelta="0"
   android:duration="1000"/>

// Animation d'échelle
<scale android:fromXScale="0.0"
   android:toXScale="1.0"
   android:fromYScale="0.0"
   android:toYScale="1.0"
   android:pivotX="50%"
   android:pivotY="50%"
   android:duration="1000" >

// Animation de rotation
<rotate android:fromDegrees="0"
   android:toDegrees="360"
   android:pivotX="50%"
   android:pivotY="50%"
   android:duration="1000" >
```

Une fois votre animation décrite, vous devrez charger et associer l'animation à une vue – ou un groupe de vues – afin d'animer celle-ci. Le chargement s'effectue à l'aide de la méthode statique `loadAnimation` de la classe `AnimationUtils` en spécifiant l'identifiant de la ressource d'animation. Pour terminer, associez l'animation ainsi chargée à la vue en appelant sa méthode `setAnimation` et en spécifiant l'objet `Animation` précédemment chargé à partir de la définition XML de l'animation.

Code 5-33 : Charger et exécuter une animation décrite en XML

```
Animation animation = AnimationUtils.loadAnimation(this, R.anim.animation1);
animation.setRepeatMode(Animation.RESTART);
animation.setRepeatCount(Animation.INFINITE);
maVueAAnimer.startAnimation(animation);
```

Créer une série d'animations

Vous pouvez créer une série d'animations en utilisant la classe `AnimationSet` ou l'attribut `set` dans la définition XML. Vous allez pouvoir exécuter ces animations soit simultanément soit en décalé dans le temps.

Pour vous aider à planifier l'exécution des différentes animations vous pouvez utiliser les attributs XML (ou leur méthode équivalente au niveau des classes) communs à tous les types d'animations par interpolation.

Tableau 5–4 Propriétés communes des animations par interpolation pour l'exécution planifiée

Nom de la propriété	Description
`duration`	La durée en millisecondes de l'exécution totale de l'animation.
`fillAfter`	Une valeur booléenne pour spécifier si la transformation doit s'effectuer avant l'animation.
`fillBefore`	Une valeur booléenne pour spécifier si la transformation doit s'effectuer après l'animation.
`interpolator`	L'interpolateur utilisé pour gérer l'animation (cf. ci-après dans ce chapitre).
`startOffset`	Le délai en millisecondes avant le démarrage de l'animation. La référence à l'origine de ce délai est le début de la série d'animation. Si vous ne spécifiez pas cette propriété, l'animation sera exécutée immédiatement, de la même façon que vous auriez spécifié la valeur 0.
`repeatCount`	Définit le nombre de fois où l'animation doit être jouée.
`repeatMode`	Définit le mode de répétitions si le nombre de répétitions est supérieur à 0 ou infini. Les valeurs possibles sont `restart` (valeur 1) pour recommencer l'animation, et `reverse` (valeur 2) pour jouer l'animation à l'envers.

Code 5-34 : Réalisation d'une série d'animation en code Java

```
// Création d'une série d'animations. La valeur true spécifie que
// chaque animation utilisera l'interpolateur de l'AnimationSet.
// Si false, chaque animation utilisera l'interpolateur
// définit dans chacune d'elle.
AnimationSet serieAnimations = new AnimationSet(true);

AlphaAnimation animationAlpha = new AlphaAnimation(0.0f, 1.0f);
animationAlpha.setDuration(500);
// Ajout de l'animation à la série d'animation
serieAnimation.addAnimation(animation);

TranslateAnimation animationTranslation = new TranslateAnimation(
   Animation.RELATIVE_TO_SELF, 0.0f,
   Animation.RELATIVE_TO_SELF, 0.0f,
   Animation.RELATIVE_TO_SELF, -1.0f,
   Animation.RELATIVE_TO_SELF, 0.0f
);
animationTranslation.setDuration(300);
animationTranslation.setStartOffset(500);
// Ajout de l'animation à la série d'animations
serieAnimation.addAnimation(animation);

// Exécution de la série d'animations
vueAAnimer.startAnimation(serieAnimation);
```

Code 5-35 : Réalisation d'une série d'animation en XML

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
   android:interpolator="@android:anim/accelerate_interpolator">
   <alpha
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:interpolator="@android:anim/accelerate_interpolator"
      android:fromAlpha="0.0" android:toAlpha="1.0" android:duration="500" />
   <translate android:fromXDelta="0"
      android:toXDelta="100%p"
      android:startOffset="500"
      android:duration="300" />
</set>
```

Animer un groupe de vues

Animer une vue permet de créer des effets très saisissants mais limités à leur périmètre. Si vous souhaitez animer la page complète, par exemple pour réaliser une page qui se tourne ou un effet d'apparition digne des transitions des meilleurs logiciels de présentation, Android permet d'appliquer une transformation à une mise en page complète.

Pour animer un groupe de vues (toute mise en page dérivant de la classe `ViewGroup`) vous devrez spécifier l'animation à la méthode `setLayoutAnimation` du groupe de vues.

Code 5-36 : Chargement et exécution d'une animation d'un groupe de vues

```
AnimationSet serieAnimations = new AnimationSet(true);
...
serieAnimation.addAnimation(animation);

TranslateAnimation animationTranslation = ...
...
serieAnimation.addAnimation(animation);

// Crée un gestionnaire d'animation qui appliquera l'animation à
// l'ensemble des vues avec un délai - ici nul - entre chaque vue.
LayoutAnimationController controller =
                  new LayoutAnimationController(serieAnimation, 0.f);
// Spécifie l'animation de groupe de vues à utiliser.
groupeVues.setLayoutAnimation(controller);
// Démarre l'animation du groupe de vues.
groupeVues.startLayoutAnimation();
```

Gérer l'accélération des animations avec les interpolateurs

Le délai d'exécution de l'animation permet de paramétrer la vitesse des animations. Pour spécifier une accélération ou une décélération, vous devez spécifier un interpolateur. Un interpolateur sert à calculer la position de l'objet en fonction du temps et de la durée de l'animation.

Par défaut, le système utilise un interpolateur linéaire. Ce dernier est dit « linéaire » car le positionnement de la vue est rigoureusement proportionnel au temps d'exécution de l'animation. Si vous êtes à 50% du temps d'exécution, la position d'une vue lors d'une translation sera de 50% du chemin, lors d'une rotation de 50% l'angle de rotation, etc.

Si vous souhaitez donner un effet de vitesse au départ suivi d'une décélération à l'arrivée, vous pouvez utiliser un interpolateur différent :

- `AccelerateDecelerateInterpolator` : permet à une animation de démarrer et de s'arrêter lentement avec une accélération entre les deux ;
- `AccelerateInterpolator` : permet à une animation de démarrer lentement puis d'accélérer jusqu'à la fin ;
- `AnticipateInterpolator` : permet à l'animation de démarrer à l'envers puis de revenir à l'endroit ;
- `AnticipateOvershootInterpolator` : permet à l'animation de démarrer à l'envers puis de revenir à l'endroit tout en dépassant la valeur cible pour finir à la valeur finale ;

- `BounceInterpolator` : l'animation se terminera avec des « sautes » comme si l'objet rebondissait ;
- `CycleInterpolator` : répète l'animation le nombre de fois spécifié selon un modèle sinusoïdal.
- `DecelerateInterpolator` : permet à une animation de démarrer rapidement puis de décélérer jusqu'à la fin ;
- `OvershootInterpolator` : l'animation dépasse la valeur cible avant de revenir à la valeur finale ;
- `LinearInterpolator` : l'animation est homogène dans le temps et l'espace.

Vous pouvez spécifier l'interpolateur dans votre code ou directement dans le fichier XML décrivant votre animation :

Code 5-37 : Spécifier l'interpolateur

```
TextView vueTexte = (TextView)findViewById(R.id.textDemo);

// Charge et paramètre l'animation
Animation animation = AnimationUtils.loadAnimation
   (this, R.anim.animation1);
animation.setRepeatMode(Animation.RESTART);
animation.setRepeatCount(Animation.INFINITE);
// Spécifie un interpolateur à 'rebondissement'.
BounceInterpolator interpolateurRebondissement =
                new BounceInterpolator();
animation.setInterpolator(interpolateurRebondissement);
// Démarre l'animation
vueTexte.startAnimation(animation);
```

Dans une définition XML d'animations, vous pouvez spécifier un interpolateur en utilisant l'attribut `android:interpolator`. La valeur de cet attribut doit être de la forme `@android:anim/nomInterpolateur` (vous pouvez trouver la liste de ces noms dans `android.R.anim`).

Code 5-38 : Spécifier un interpolateur dans la définition XML d'une animation

```
...
<Translate ...
   android:interpolator="@android:anim/bounce_interpolator"
... />
...
```

Vous pouvez spécifier un interpolateur différent pour chaque animation. Dans le cadre d'une série d'animations, vous pouvez également spécifier que vous souhaitez utiliser un interpolateur commun en le définissant avec l'attribut `shareInterpolator`, qui

attend un booléen. Ajoutez l'interpolateur que vous souhaitez utiliser à l'instar des animations avec l'attribut `interpolator` :

Code 5-39 : Définir l'interpolateur d'une série d'animations XML

```
<set android:shareInterpolator="true"
   android:interpolator="@android:anim/accelerate_interpolator">
...
</set>
```

Recourir aux événements pour l'animation

Afin de pouvoir gérer de façon plus subtile les animations par interpolation, la plateforme Android propose, sous la forme d'un écouteur (listener), de gérer les événements du cycle de vie d'une animation. À l'aide de ces événements, vous allez pouvoir exécuter des actions avant le démarrage de l'animation, après son exécution et entre chaque répétition d'une animation.

Pour gérer ces événements, implémentez l'interface `AnimationListener` puis spécifiez-en une instance à la méthode `setAnimationListener` de votre objet `Animation`. Les méthodes à surcharger sont indiquées dans le tableau ci-après.

Tableau 5–5 Méthodes de l'interface AnimationListener

Nom de la méthode	Description
`onAnimationStart`	Notifie le début de l'animation.
`onAnimationEnd`	Notifie la fin de l'animation.
`onAnimationRepeat`	Notifie la répétition de l'animation. À chaque boucle, cette méthode sera appelée.

Code 5-40 : Réalisation d'un écouteur d'animation

```
Animation animation = AnimationUtils.loadAnimation(...)
...
animation.setAnimationListener(
   new Animation.AnimationListener() {

   @Override
   public void onAnimationStart(Animation animation) {
      // Notifie le début de l'animation
   }

   @Override
   public void onAnimationRepeat(Animation animation) {
      // Notifie la fin de l'animation
   }
```

```
    @Override
    public void onAnimationEnd(Animation animation) {
        // Notifie la répétition de l'animation.
        // À chaque boucle, cette méthode sera appelée.
    }
});
```

Les animations image par image

Les animations image par image permettent de spécifier une séquence d'images qui seront affichées les unes à la suite des autres selon un délai spécifié, un peu à la manière d'un dessin animé.

Comme les animations par interpolation, une séquence d'animation image par image peut être définie soit dans le code soit dans une ressource XML externe. La séquence d'images est décrite dans une liste d'éléments `item` contenant dans un élément parent `animation-list`. Pour chaque élément, vous spécifiez l'identifiant de la ressource et sa durée d'affichage :

Code 5-41 : Séquence d'une animation image par image

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    <!-- Ce paramètre est utilisé pour spécifier si l'animation doit être
jouée une ou plusieurs fois -->
    android:oneshot="false">
    <!-- Chaque image est déclarée dans un élément item -->
    <itemandroid:drawable="@drawable/android1" android:duration="500" />
    <item android:drawable="@drawable/android2" android:duration="500" />
    <item android:drawable="@drawable/android3" android:duration="500" />
</animation-list>
```

Au niveau du code vous devez spécifier l'animation à la vue via sa méthode `setBackgroundResource` en précisant l'identifiant de l'animation séquentielle d'images. Pour démarrer l'animation, récupérez l'objet `AnimationDrawable` en appelant la méthode `getBackground` de la vue, puis appelez la méthode `start` de l'animation.

> **À SAVOIR Ne pas démarrer l'animation dans la méthode onCreate**
>
> Attention à l'endroit où vous appelez la méthode `start`. En effet, si vous appelez cette méthode pour lancer votre animation depuis la méthode `onCreate`, l'animation ne démarrera pas !

Code 5-42 : Animation d'une vue avec une animation image par image

```
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main);

   ImageView image = (ImageView) findViewById(R.id.image);
   // Spécifie l'animation comme fond de l'image
   image.setBackgroundResource(R.drawable.animation_images);
   image.setOnClickListener(new View.OnClickListener() {

      @Override
      public void onClick(View v) {
         ImageView image = (ImageView) AnimationImageParImage.this
            .findViewById(R.id.image);
         // Récupère l'animation à animer
         AnimationDrawable animation = (AnimationDrawable) image
            .getBackground();
         // Démarre l'animation
         animation.start();
      }
   });
}
```

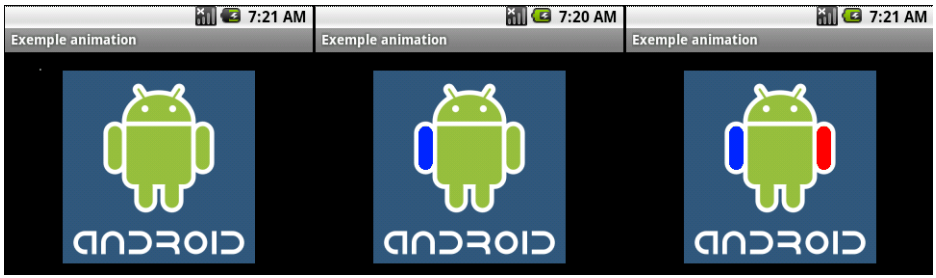


Figure 5–25 Résultat de l'animation du code 5-42

Les AppWidgets

Hormis les icônes des applications, le bureau (ou écran d'accueil) peut également contenir ce que l'on appelle les *AppWidgets*. Leur utilisation consiste à exporter votre application sous la forme d'un contrôle personnalisable disponible directement sur le bureau de l'utilisateur.

> **JARGON Widget, appwidget et gadget**
>
> Rappelons que le mot *widget* vient de la contraction des mots anglais window et gadget. À l'origine, il est utilisé pour désigner un programme miniature que l'on dispose sur son espace de travail. Tout au long de ce chapitre nous appellerons les AppWidgets des *gadgets*.

Les gadgets sont une très bonne manière d'étendre vos applications. Ils offrent un accès direct aux fonctionnalités de celles-ci, tout en gardant un format discret. Nous pouvons les comparer à de petits programmes embarqués, tels que des extensions, comme le proposent aujourd'hui les systèmes d'exploitation de dernière génération.

Les avantages sont nombreux :

- il est inutile de lancer l'application à proprement parler ;
- l'application est toujours disponible sur le bureau ;
- plusieurs gadgets peuvent se partager un même écran ;
- les ressources de fonctionnement ne sont allouées que lorsque cela est nécessaire ;
- chaque gadget peut sauvegarder ses propres paramètres.

À titre d'exemple l'horloge analogique fait partie des gadgets, ainsi que le lecteur média, l'album photo et le panneau de contrôle comme le montre l'écran ci-joint.

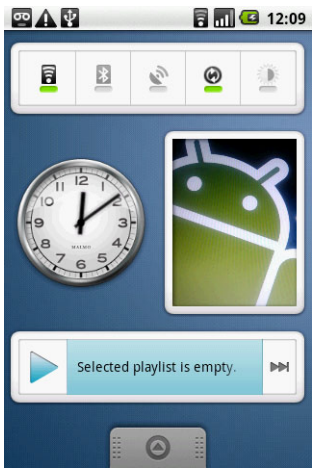


Figure 5–26
Exemples de gadgets (source Google)

Création d'un gadget

Pour concevoir un gadget nous allons avoir besoin de réaliser plusieurs tâches :

1. créer un objet hérité de la classe `AppWidgetProvider` ;
2. définir l'aspect visuel du gadget dans un autre fichier XML ;

3 définir les paramètres du `AppWidgetProvider` dans un fichier XML ;
4 ajouter une activité pour personnaliser le gadget ;
5 déclarer le gadget dans le fichier de configuration de l'application.

Conception du fournisseur

La tâche la plus importante revient sans aucun doute à la conception de l'`AppWidgetProvider`, en d'autres termes au fournisseur de gadget. Attention son nom pourrait laisser sous-entendre que cette classe pourrait s'occuper de la création du gadget, or il n'en n'est rien. La classe `AppWidgetProvider` hérite directement du `BroadcastReceiver` et n'est donc rien d'autre qu'un simple écouteur de messages.

Une fois ce constat réalisé, plusieurs possibilités s'offrent alors à vous : soit gérer ces messages à la manière d'un simple `BroadcastReceiver` en réécrivant la méthode `onReceive`, soit en utilisant les méthodes simplifiées de l'`AppWidgetProvider` (méthodes `onUpdate` et `getSetting`). Bien évidemment, nous ne pouvons que vous conseiller d'utiliser ces dernières.

> **ATTENTION Les gadgets utilisent des contrôles RemoteViews**
>
> Sachez que les gadgets n'utilisent pas des contrôles standards : vous constaterez que les contrôles récupérés sont de type `RemoteViews`.

Un gadget peut avoir plusieurs instances en cours d'utilisation : c'est pourquoi chaque gadget possède un identifiant propre. Le système garde également un compteur du nombre d'instances par type de gadget.

L'exécution du code ci-dessous s'effectue alors que le gadget est déjà créé (la création aura lieu avec une activité spécifique, détaillée plus loin dans ce chapitre), par conséquent un identifiant lui est déjà attribué.

Dès qu'une modification s'opère sur un des gadgets de notre application, le système envoie un message à notre implémentation `AppWidgetProvider` pour l'en informer.

Code 5-43 : Conception du WidgetProvider

```
import android.app.PendingIntent;
import android.appwidget.AppWidgetManager;
import android.appwidget.AppWidgetProvider;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.net.Uri;
import android.preference.PreferenceManager;
import android.widget.RemoteViews;
```

```
public final class WidgetProvider
   extends AppWidgetProvider {

   @Override
   public void onUpdate(Context context,
      AppWidgetManager appWidgetManager, int[] appWidgetIds) {
      super.onUpdate(context, appWidgetManager, appWidgetIds);

      // On récupère le texte dans les préférences
      String text = getSetting(context);
      // Nous effectuons une copie du nombre de widgets à mettre à jour
      int n = appWidgetIds.length;
      // On boucle
      for(int i=0; i<n; i++){
         // Crée une nouvelle vue
         RemoteViews views = new RemoteViews(context.getPackageName()
            , R.layout.widgetlayout);
         // Définit le texte
         views.setTextViewText(android.R.id.text1, text);
         // Crée un nouvel événement pour le clic
         Intent intent = new Intent(context,WidgetConfigActivity.class);
         intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
         intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID
            ,appWidgetIds[i]);
         intent.setData(Uri.parse("androidwidget://widget/id/"
                                  + appWidgetIds[i]));
         PendingIntent pendingIntent = PendingIntent.getActivity(context,
                                                                  0, intent, 0);
         // On ajoute cet événement sur la vue
         views.setOnClickPendingIntent(android.R.id.text1, pendingIntent);
         // On met à jour la widget
         appWidgetManager.updateAppWidget(appWidgetIds[i], views);
      }
   }

   private String getSetting(Context context){
      SharedPreferences settings =
         PreferenceManager.getDefaultSharedPreferences(context);
      return settings.getString("text",context.getString(R.string.hello));
   }
}
```

Définir l'interface du gadget

Comme vous pouvez le voir dans le code précédent, nous utilisons les paramètres envoyés à la méthode `onUpdate` pour instancier un nouvel objet `RemoteViews`. Cette vue sera chargée depuis un fichier XML que nous aurons préalablement créé dans le dossier `res/layout` du projet. Ce fichier XML ressemble à n'importe quelle autre

déclaration d'interface à la différence que les composants utilisables, des `RemoteViews`, sont limités.

Voici les contrôles dont vous disposez :

- `FrameLayout`
- `LinearLayout`
- `RelativeLayout`
- `AnalogClock`
- `Button`
- `Chronometer`
- `ImageButton`
- `ImageView`
- `ProgressBar`
- `TextView`

Nous utiliserons pour l'exemple suivant un simple bouton imbriqué dans un `LinearLayout` comme défini ci-dessous.

Code 5-44 : Conception du widgetLayout.xml

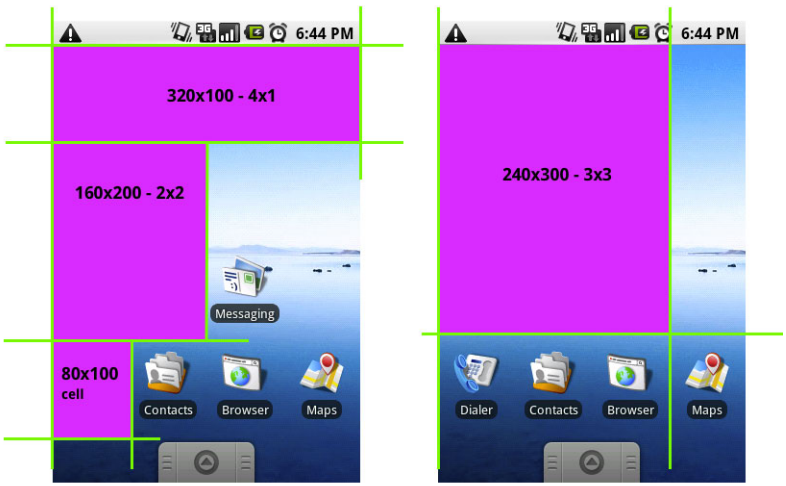
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:gravity="center"
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@drawable/background">
    <Button
        android:layout_width="283dip"
        android:layout_height="54dip"
        android:id="@android:id/text1"
        android:textColor="#ff000000"
        android:text="@string/hello"
        android:background="@drawable/button"
        android:gravity="center"
        />
</LinearLayout>
```

Notez la taille du bouton en *dip*. Vous savez que chaque icône sur l'écran d'accueil du système Android tient une place bien précise. La formule de calcul de la taille d'une cellule est la suivante.

À RETENIR Taille minimale d'un gadget

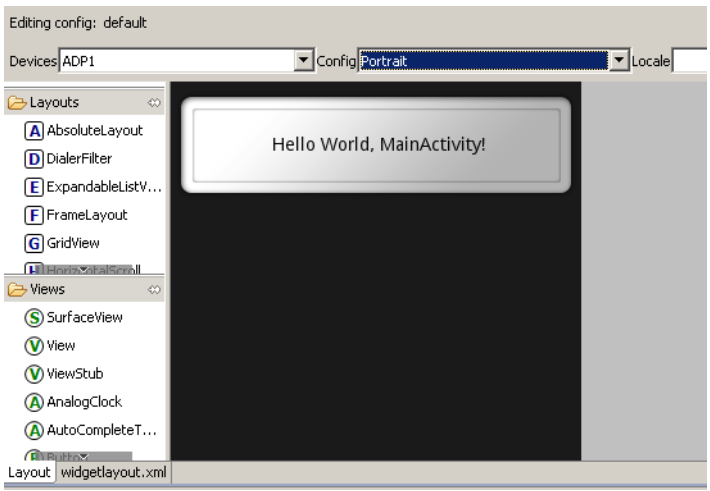
Taille minimale d'un gadget en dip/dp = (Nombre de cellules * 74dip) – 2dip

Figure 5–27
Exemples de taille de gadget (source Google)



Si vous utilisez le concepteur d'interface d'ADT (voir annexes), votre gadget devrait ressembler à celui ci-dessous.

Figure 5–28
Exemple de gadget



Définir les paramètres du gadget

Vous devez stocker les paramètres du gadget (tailles minimales, fréquence de rafraîchissement, etc.) dans un fichier XML à part du récepteur et de l'interface. C'est aussi dans ce fichier que vous attribuerez au gadget une activité pour le configurer pendant l'exécution (cf. plus loin dans ce chapitre).

Créez ce fichier de paramétrage XML dans `res/xml/widget.xml` contenant la structure suivante :

Code 5-45 : Conception du fichier widget.xml pour stocker les paramètres du gadget

```
<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="294dip"
    android:minHeight="72dip"
    android:updatePeriodMillis="86400000"
    android:initialLayout="@layout/widgetlayout"
    android:configure="com.eyrolles.android.widget.WidgetConfigActivity" >
</appwidget-provider>
```

Ainsi fait, notre gadget pourrait très bien fonctionner. Néanmoins il manque encore plusieurs choses. En effet, nous n'avons pas déclaré le gadget dans le fichier de configuration de l'application, ni d'activité pour créer ou configurer le gadget pendant l'exécution.

Associer le gadget à une activité

Nous allons effectuer d'une pierre deux coups et lancer une activité qui va à la fois créer le gadget et permettre sa configuration pendant l'exécution.

Pour ajouter un gadget, l'utilisateur doit appuyer sur l'écran d'accueil pour faire apparaître un menu. Ce menu offre plusieurs possibilités dont l'ajout de gadget. Lorsque l'utilisateur choisit *widget* le système renvoie une liste d'applications ayant déclaré un gadget dans son fichier de configuration (nous le verrons en section suivante). Lorsque l'utilisateur choisit un des éléments dans cette liste, le système va démarrer l'activité qui lui est associée et ainsi demander la création d'un gadget. Le système attendra la fermeture de cette activité pour obtenir l'autorisation de créer le gadget. Ce mécanisme s'appuie sur l'emploi des méthodes `startActivityForResult` et `setResult` que nous avons détaillées au chapitre 3.

Le système envoie également un extra de type `AppWidgetManager.INVALID_APPWIDGET_ID` permettant d'identifier le gadget qui va être créé. Il ne nous reste plus qu'à charger et sauver les paramètres personnalisés liés à cet identifiant.

Code 5-46 : Conception de l'activité de configuration

```
package com.eyrolles.android.widget;

import android.app.Activity;
import android.appwidget.AppWidgetManager;
import android.content.Intent;
import android.os.Bundle;
import android.preference.Preference;
```

```
import android.preference.PreferenceActivity;
import android.preference.Preference.OnPreferenceChangeListener;

public class WidgetConfigActivity
   extends PreferenceActivity
   implements OnPreferenceChangeListener {

   int mAppWidgetId;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      // Nous ajoutons les contrôles via le fichier XML
      addPreferencesFromResource(R.xml.preferences);
      findPreference("text").setOnPreferenceChangeListener(this);

      // Nous récupérons l'identifiant du widget
      mAppWidgetId = getIntent().getIntExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
                                             AppWidgetManager.INVALID_APPWIDGET_ID);

        // Nous testons la validité de l'identifiant
        if (mAppWidgetId == AppWidgetManager.INVALID_APPWIDGET_ID){
           // L'identifiant n'est pas valide on quitte
           setResult(Activity.RESULT_CANCELED);
           finish();
        }else{
           // L'identifiant est valide : on renvoie OK
           Intent resultValue = new Intent();
           resultValue.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, mAppWidgetId);
           setResult(RESULT_OK, resultValue);
        }
   }

   @Override
   public boolean onPreferenceChange(Preference preference, Object newValue) {
      // Mise à jour de la widget
      Intent intent = new Intent(AppWidgetManager.ACTION_APPWIDGET_UPDATE);
      intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_IDS,
                      new int[]{mAppWidgetId});
      sendBroadcast(intent);
      return true;
   }
}
```

Dans nos exemples nous avons sauvegardé/chargé une préférence dont voici le contenu, que vous placerez dans un fichier `preferences.xml` dans `res/xml` :

Code 5-47 : Conception du fichier preferences.xml

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
   <EditTextPreference
      android:summary="Cliquez pour modifier"
      android:title="Titre"
      android:key="text"
      />
</PreferenceScreen>
```

Paramétrer le fichier de configuration de l'application

Une fois toutes ces étapes réalisées, vous devez encore déclarer l'existence du gadget dans le fichier de configuration de l'application en spécifiant :

- le récepteur ;
- l'activité de création et de configuration.

Pour déclarer le récepteur, ajoutez un élément `<receiver>` comme nœud de l'élément`<application>`. La différence par rapport à un composant de type écouteur standard est l'action du filtre et ses métadonnées. Le filtre doit capturer les actions de type `android.appwidget.action.APPWIDGET_UPDATE`. Vous ajouterez des informations supplémentaires dans l'élément XML enfant `<meta-data>` pour spécifier le fichier de paramétrage du gadget (situé dans `res/xml/widget.xml`). La paire de cette métadonnée est composée du nom `android.appwidget.provider` et de sa valeur `@xml/widget`.

Pour déclarer l'activité de création et de configuration du gadget, insérez un composant activité dans le fichier de configuration et déclarez un filtre sur l'action `android.appwidget.action.APPWIDGET_CONFIGURE` :

Code 5-48 : Modification du fichier de configuration pour déclarer le gadget

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.eyrolles.android.widget"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        // L'activité de création et de configuration du gadget
        <activity android:name=".WidgetConfigActivity">
            <intent-filter>
               <action android:name="android.appwidget.action.APPWIDGET_CONFIGURE" />
            </intent-filter>
        </activity>
```

```
        // La déclaration du récepteur du gadget
         <receiver android:name=".WidgetProvider">
             <intent-filter>
                 <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
             </intent-filter>
             <meta-data
                 android:name="android.appwidget.provider"
                 android:resource="@xml/widget" />
         </receiver>
    </application>
    <uses-sdk android:minSdkVersion="4" />
</manifest>
```

Vous remarquerez qu'il n'y a pas d'activité par défaut dans ce fichier de configuration – pas de présence de filtre sur une action de type LAUNCHER – c'est pourquoi notre application ne démarrera pas si l'on clique sur son icône.

L'implémentation que nous venons de réaliser peut paraître simpliste et l'est dans la réalité. Néanmoins, elle regroupe toutes les fonctions de base d'un gadget et servira de modèle pour vos futures créations.

DU CÔTÉ DE L'UTILISATEUR Ajouter un gadget sur le bureau

Pour activer un gadget, vous devez passer par le menu d'ajout des gadgets en laissant votre doigt appuyé sur une zone vierge de l'écran d'accueil. Le menu d'ajout apparaît alors. Choisissez *Widgets* puis sélectionnez le gadget que vous souhaitez ajouter à votre page d'accueil. Un écran de configuration s'ouvre alors afin de paramétrer le gadget et notamment son titre. Confirmez vos paramètres pour voir apparaître le gadget sur la page d'accueil de l'appareil.

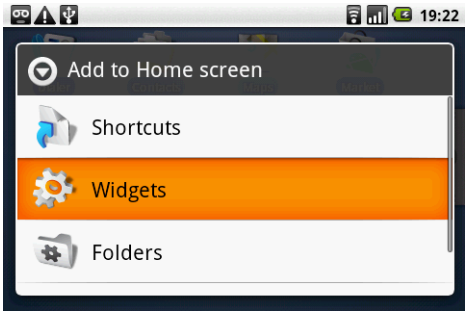


Figure 5–29 Menu d'ajout du bureau

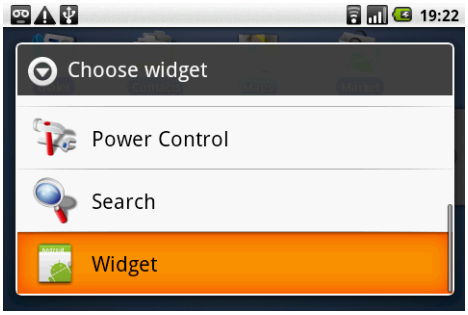


Figure 5–30 Choix d'un widget

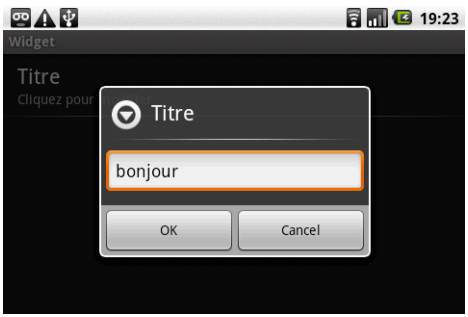


Figure 5–31 Spécifier le titre du widget

Figure 5–32 Résultat de l'ajout du widget sur le bureau

En résumé

Vous pouvez désormais proposer des interfaces graphiques riches à vos utilisateurs, que ce soit en créant des composants personnalisés, en utilisant des adaptateurs, des animations ou en créant des menus. Vous conviendrez que la plate-forme Android est relativement bien fournie et aisée à appréhender. Sachez utiliser ces possibilités de façon harmonieuse, l'ergonomie de l'application étant un des aspects cruciaux dans le choix et l'adoption des applications par les utilisateurs.

Rendre accessible votre application à des millions de personnes ne parlant pas votre langue est aussi une excellente façon d'étendre son rayonnement et de diffuser vos applications au delà des frontières. Là aussi, la plate-forme Android apporte sa simplicité, et le complément ADT les outils nécessaires à votre productivité.

Si créer une application n'est pas suffisant pour vos utilisateurs et si vous souhaitez la mettre à disposition de l'utilisateur directement sur son bureau, la possibilité de créer des gadgets sera alors un excellent choix. Cependant, rien ne sert de créer un gadget mal réfléchi, ou juste parce que vous savez développer un gadget Android... Ne créez pas de gadgets qui soient trop... « gadget » !

6

Persistance des données

Une application qui ne peut garder trace des interactions avec l'utilisateur et conserver des données d'une session à une autre est une application sans vie, que l'utilisateur s'empressera souvent de désinstaller. Toute application doit pouvoir charger et enregistrer des données.

Quatre techniques sont à la disposition du développeur pour faire persister des données dans le temps, chacune offrant un compromis entre facilité de mise en œuvre, rapidité et flexibilité, tout cela de façon unifiée quelle que soit l'application :

- l'enregistrement du parcours de l'utilisateur dans l'application (écrans rencontrés avec leur contexte) ou *persistance des activités* (un écran étant lié à une activité). Lorsqu'un utilisateur navigue dans une application, il est important de pouvoir conserver l'état de l'interface utilisateur pour préparer son retour sur les écrans potentiellement déchargés par le système pour gagner des ressources.
- le *mécanisme de préférence clé/valeur*. Les fichiers de préférences sont avant tout utilisés pour stocker des préférences utilisateur, la configuration d'une application ou l'état de l'interface d'une activité. Ce mécanisme fournit un stockage simple et efficace par paire clé/valeur de valeurs primitives. Il existe également un système d'écran de préférences qui permet de créer des écrans de configuration rapidement et simplement ;
- l'utilisation d'un *système de fichiers*. Les fichiers sont le support de stockage élémentaire pour lire et écrire des données brutes dans le système de fichiers d'Android. Vous pourrez stocker des fichiers sur le support de stockage interne ou externe d'Android ;

- l'utilisation d'une *base de données SQLite*. Les bases de données SQLite sont réservées pour le stockage et la manipulation des données structurées.

Chaque méthode possède des caractéristiques et des usages propres. Il vous reviendra de choisir celle qui semblera la plus adaptée au contexte de votre application.

Persistance de l'état des applications

Le mode de fonctionnement du cycle de vie d'une activité vous empêchera souvent de savoir à quel moment précis une activité d'arrière-plan sera déchargée de la mémoire. Dès lors, il était indispensable que la plate-forme Android propose un moyen d'enregistrer l'état des activités afin que l'utilisateur retrouve une interface graphique identique entre chacune de ses sessions.

Il existe deux mécanismes de sauvegarde de l'état de l'activité : le premier consiste à gérer la persistance grâce aux méthodes du cycle de vie de l'activité et le second permet de la gérer de façon manuelle à l'aide des classes de persistance de l'API.

La plupart du temps la persistance de l'interface utilisateur se fera via les méthodes du cycle de vie de l'activité. Ce système a été spécifiquement élaboré pour faciliter l'enregistrement et la gestion de l'état de l'interface tout au long du cycle de vie de l'activité. La seconde méthode sera traitée plus loin dans ce chapitre.

La méthode `onSaveInstanceState` d'une activité est appelée lorsque le système a besoin de libérer des ressources et de détruire l'activité (si vous appuyez sur *Retour* pour quitter l'application, l'activité n'est pas enregistrée puisqu'elle ne sera pas mise dans la pile de l'historique des activités). L'objet utilisé pour stocker les données est de type `Bundle` et sera passé comme paramètre aux méthodes `onCreate` et `onRestoreInstanceState` afin de pouvoir rétablir l'interface utilisateur lors de la création ou restauration de l'activité.

Par défaut, les méthodes `onSaveInstanceState`, `onRestoreInstanceState` et `onCreate` de la classe mère fonctionnent selon le principe que les valeurs de toutes les vues possédant un attribut `id` renseigné sont enregistrées, puis restaurées.

L'implémentation par défaut de la méthode `onSaveInstanceState` enregistre l'état des vues identifiées dans un objet `Bundle`. L'objet ainsi sauvegardé est ensuite passé aux méthodes `onCreate` et `onRestoreInstanceState` pour restaurer l'ensemble.

Afin d'illustrer ce mécanisme, créons une application élémentaire possédant une simple activité composée de quelques champs.

Création d'une application élémentaire pour l'exemple

```
<LinearLayout android:id="@+id/layoutPrincipal"
   android:layout_width="fill_parent" android:layout_height="fill_parent"
   android:orientation="vertical"
   xmlns:android="http://schemas.android.com/apk/res/android">

   <TextView android:id="@+id/nomDescription" android:layout_width="wrap_content"
   android:layout_height="wrap_content" android:text="Saisissez votre nom :">
   </TextView>

   <EditText android:id="@+id/nom" android:layout_width="fill_parent"
      android:layout_height="wrap_content" android:textSize="18sp">
   </EditText>

   <TextView android:id="@+id/messageDescription"
      android:layout_width="fill_parent" android:layout_height="wrap_content"
      android:text="Saisissez un message (qui ne sera pas enregistré) :">
   </TextView>

   <!-- Cette vue ne possède pas d'attribut id, elle ne sera pas enregistrée par
défaut. -->
   <EditText
      android:layout_width="fill_parent"
      android:layout_height="wrap_content" android:textSize="18sp">
   </EditText>
</LinearLayout>
```

Remarquez que le champ de saisie du message ne comporte volontairement pas d'identifiant ; de cette façon, Android n'enregistrera pas la valeur de cette vue.

Nous créons ensuite l'activité suivante.

Code 6-1 : Squelette d'une application de persistance

```
public class SauvegardeEtatActivite extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

   @Override
   protected void onSaveInstanceState(Bundle savedInstanceState) {
      super.onSaveInstanceState(savedInstanceState);
      Toast.makeText(this, "État de l'activité sauvegardé",
                     Toast.LENGTH_SHORT).show();
   }
```

```
    @Override
    protected void onDestroy() {
        super.onDestroy();
        Toast.makeText(this, "L'activité est détruite", Toast.LENGTH_SHORT).show();
    }
}
```

N'oubliez pas de configurer l'application de façon à ce que l'activité soit lancée au démarrage de celle-ci, dans sa section `application` :

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.eyrolles.persistance"
      android:versionCode="1"
      android:versionName="1.0.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".SauvegardeEtatActivite"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Exécutez cette application et remplissez les champs. Une fois cette opération effectuée, cliquez sur le bouton *Home* de l'émulateur. Si seul le message *État de l'activité sauvegardé* apparaît, cela signifie que l'activité n'a pas été détruite et que l'interface utilisateur possède toujours les valeurs que vous avez précédemment saisies. Si vous relancez l'application, ces valeurs réapparaîtront.

Si vous ne souhaitez pas que les écrans soient conservés après la fermeture de l'application, utilisez l'une des options des outils développeur de l'émulateur. Naviguez dans *Dev Tools*>*Development Settings* et cochez la case *Immediately destroy activities*.

Si vous répétez les étapes précédentes et que vous appuyez sur le bouton *Home* de l'émulateur, vous enregistrerez l'état de l'interface et détruirez du même coup l'activité. Une fois l'activité enregistrée et effectuée, si vous relancez l'application, vous observerez que seul le champ du nom a été restauré et que le champ du message – ne possédant pas d'identifiant – est quant à lui vide.

Configurer le mode de conservation des activités

Le mécanisme par défaut présente de nombreux avantages, le premier étant qu'aucune action ou code spécifique n'est requis pour le développeur afin de faire persister l'interface des activités. Néanmoins, ce comportement trouve vite ses limites, notamment quand vous souhaitez ne pas enregistrer un champ confidentiel qui pour des raisons de logique applicative doit posséder un identifiant. Vous vous trouverez dans la même situation si vous souhaitez traiter les valeurs en amont ou en aval du rendu de l'interface ou alors ajouter des valeurs supplémentaires.

Pour personnaliser l'enregistrement de l'état de l'activité ou enregistrer des informations supplémentaires, redéfinissez ou modifiez simplement la méthode `onSaveInstanceState` pour l'enregistrement et les méthodes `onCreate` ou `onRestoreInstanceState` pour la restauration. Utilisez ensuite l'objet de type `Bundle` passé en paramètre de ces méthodes pour lire ou écrire des valeurs.

Reprenez le code utilisé dans l'exemple précédent et modifiez-le pour qu'il ressemble au code suivant :

Code 6-2 : Squelette d'une application de persistance (suite)

```
public class SauvegardeEtatActivite extends Activity {

   private final static String MA_CLE = "MaCle";

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      if (savedInstanceState != null && && savedInstanceState.containsKey(MA_CLE)) {
         String val = savedInstanceState.getString(MA_CLE);
         Toast.makeText(this, "onCreate() : " + val,
               Toast.LENGTH_SHORT).show();
      }
      setContentView(R.layout.persistance_etat_activite1);
   }

   @Override
   protected void onSaveInstanceState(Bundle savedInstanceState) {
      super.onSaveInstanceState(savedInstanceState);
      savedInstanceState.putString(MA_CLE, "Ma valeur !");
      Toast.makeText(this, "État de l'activité sauvegardé",
            Toast.LENGTH_SHORT).show();
   }

   @Override
   protected void onDestroy() {
      super.onDestroy();
```

```
            Toast.makeText(this, "L'activité est détruite", Toast.LENGTH_SHORT)
                  .show();
        }
    }
```

Exécutez l'application à nouveau, remplissez les champs, puis appuyez sur le bouton *Home* de l'émulateur. Relancez l'application une seconde fois, vous devriez alors voir apparaître le message `onCreate() : Ma valeur !` et la valeur du champ `Nom` restaurée.

De la même façon, vous pourriez placer le code de la méthode `onCreate` dans une redéfinition de la méthode `onRestoreInstanceState`. Sachez que les méthodes `onCreate` et `onRestoreInstanceState` sont uniquement appelées si l'activité a été précédemment détruite.

> **BONNE PRATIQUE** **Stockage de données peu liées à l'état de l'interface**
>
> Le mécanisme que nous venons de voir est utilisé pour préserver une interface utilisateur. Il n'a pas été conçu pour enregistrer des préférences n'ayant pas ou peu de rapport avec l'interface, surtout si vous voulez conserver des valeurs même quand l'utilisateur annule une activité en appuyant sur le bouton *Retour*. Pour cela, il vous faudra utiliser les préférences partagées.

Les préférences partagées

Au-delà de vouloir simplement enregistrer l'état de l'application, vous envisagez sûrement d'enregistrer et de retrouver d'autres valeurs propres à votre application, par exemple les paramètres de l'utilisateur, la configuration de l'application, etc. Android fournit un moyen d'enregistrer d'autres préférences que celles des activités et d'autoriser un accès partagé à ces informations à travers les différents composants de l'application.

Ce mécanisme d'enregistrement, appelé *préférences partagées*, permet la persistance de propriétés sous la forme d'un ensemble de paires clé/valeur. Que ce soit pour stocker les paramètres d'exécution d'une application ou les préférences de l'utilisateur, ce système de stockage des informations est relativement simple et léger à mettre en œuvre. Les préférences partagées peuvent stocker des données de types `boolean`, `int`, `long`, `float` et `String`.

Récupérer les préférences partagées

Complètement indépendant du cycle de vie de l'activité, ce mécanisme est surtout utilisé pour enregistrer des données qui doivent être partagées entre deux sessions utilisateur et accessibles par l'ensemble des composants de l'application.

Vous pouvez récupérer les préférences en effectuant un appel à la méthode `getPreferences` de l'activité courante. Cette méthode vous renverra une instance d'une classe `SharedPreferences` à partir de laquelle vous pourrez extraire les valeurs.

Vous pouvez créer plusieurs ensembles de préférences partagées, chacun identifié par un nom unique. La méthode `getSharedPreferences(String, int)` propose de spécifier le nom dans le premier argument.

> La méthode `getPreferences(int)` est un appel à la méthode surchargée `getSharedPreferences(String, int)` en utilisant le nom de classe de l'activité courante comme paramètre de nom de préférence.

Une fois l'instance de la classe `SharedPreferences` récupérée, la lecture des données s'effectue via l'utilisation des différentes méthodes `getBoolean`, `getString`, `getFloat`, `getInt` et `getLong`, respectivement pour récupérer des valeurs de types `boolean`, `String`, `float`, `Int` et `Long`. Pour chacune de ces méthodes, vous spécifierez en premier paramètre le nom de la clé pour laquelle vous souhaitez récupérer la valeur et en second paramètre la valeur à fournir par défaut si la valeur correspondant à la clé n'est pas trouvée.

Code 6-3 : Récupération des préférences partagées

```
SharedPreferences prefs = getPreferences(Context.MODE_PRIVATE);
String nom = prefs.getString("nomUtilisateur", null);
if (nom != null){
   // Utiliser la valeur.
}
float age = prefs.getFLoat("ageUtilisateur", 0.0f);
```

Si vous souhaitez extraire toutes les paires clé/valeur des préférences en une seule opération, vous disposez de la méthode `getAll` qui vous retournera un objet de type `Map<String, ?>`.

Enregistrer ou mettre à jour des préférences partagées

L'enregistrement des préférences est rendu possible avec l'utilisation de l'interface `Editor` de la classe `SharedPreferences` qui vous offrira toutes les méthodes pour pouvoir ajouter ou modifier les valeurs. Pour récupérer une instance de la classe `Editor`, vous devez appeler la méthode `edit` de votre instance `SharedPreferences`.

Une fois en possession de ladite instance, vous utiliserez les méthodes `putBoolean`, `putString`, `putInt`, `putFloat` et `putLong` pour ajouter des valeurs respectivement de type `String`, `Int`, `Float` et `Long`. Chacune de ces méthodes prend en paramètres la

clé sous forme d'une chaîne de caractères ainsi que la valeur associée. Si une valeur est spécifiée avec une clé déjà existante, la valeur sera simplement mise à jour.

Une fois vos ajouts et modifications effectués, une dernière opération est nécessaire pour que vos données soient enregistrées. En effet, vous devrez expressément spécifier l'enregistrement des données avec l'appel à la méthode `commit` de votre objet `Editor`.

Dans l'exemple suivant, le code crée un ensemble de préférences en mémoire et ajoute des valeurs.

Code 6-4 : Enregistrer des préférences partagées

```
SharedPreferences preferences = getPreferences(Context.MODE_PRIVATE);
// Récupération d'un éditeur pour modifier les préférences.
SharedPreferences.Editor editor = preferences.edit();
editor.putString("nomUtilisateur", getNomUtilisateur());
editor.commit();
```

En fin d'opération, n'oubliez pas de demander à l'instance `Editor` d'enregistrer les données avec sa méthode `commit`.

Les permissions des préférences

Vous remarquerez que la méthode `getPreferences` de l'exemple précédent possède un paramètre définissant la permission donnée au fichier des préférences : `MODE_PRIVATE`, `MODE_WORLD_READABLE` et `MODE_WORLD_WRITABLE`. Cette option définit les droits liés aux fichiers.

La permission que vous spécifierez n'aura que peu d'influence sur le fonctionnement de l'application, mais elle en aura sur la sécurité d'accès des données enregistrées :

- `MODE_PRIVATE` est le mode par défaut et signifie que le fichier créé sera accessible uniquement à l'application l'ayant créé ;
- `MODE_WORLD_READABLE` permet aux autres applications de lire le fichier mais non de le modifier ;
- `MODE_WORLD_WRITABLE` permet aux autres applications de modifier le fichier.

Revoyons l'exemple précédent pour utiliser des préférences partagées :

Code 6-5 : l'exemple 6-3 revisité

```
public static final String MES_PREFERENCES = "MesPreferencesUtilisateur";
...
SharedPreferences prefs = getSharedPreferences(MES_PREFERENCES,
            Context.MODE_PRIVATE);
String nom = prefs.getString("nomUtilisateur", null);
```

```
   if (nom != null){
      // Utiliser la valeur.
   }
float age = prefs.getFLoat("ageUtilisateur", 0.0f);
```

EN COULISSES Chemin de stockage des préférences

Les préférences sont stockées dans des fichiers du système de fichiers interne de Android. Si nous prenons le cas de l'exemple précédent, le fichier est enregistré à l'emplacement suivant :
`/data/data/com.eyrolles.android/shared_prefs/MesPreferencesUtilisateur.xml`

En utilisant un nom pour votre ensemble de valeurs, vous avez la possibilité d'utiliser plusieurs portées pour stocker vos données et de mieux organiser ces dernières, notamment si vous avez des paires possédant le même nom de clé.

Réagir aux modifications des préférences avec les événements

La classe `SharedPreferences` propose un mécanisme d'événements pour vous permettre de maîtriser les changements de données effectués par l'application. La classe `SharedPreferences` possède une méthode `registerOnSharedPreferenceChangeListener` permettant de spécifier une méthode de rappel/écouteur qui sera appelée à chaque modification de vos préférences.

Code 6-6 : Gérer les événements des préférences partagées

```
SharedPreferences prefs = getSharedPreferences(MES_PREFERENCES,
            Context.MODE_PRIVATE);
String nom = prefs.getString("nomUtilisateur", null);
if (nom != null)
   setNomUtilisateur(nom);

// Enregistrement des écouteurs de changement de valeurs.
prefs
   .registerOnSharedPreferenceChangeListener(
            new OnSharedPreferenceChangeListener() {

      @Override
      public void onSharedPreferenceChanged(
            SharedPreferences sharedPreferences, String key) {
            // Mettez votre traitement ici.
   }
);
```

De la même façon que vous avez enregistré l'écouteur, vous pouvez le désinscrire en utilisant la méthode `unregisterOnSharedPreferenceChangeListener` en spécifiant

l'instance de l'écouteur passée en paramètre lors de l'enregistrement, avec la méthode `registerOnSharedPreferenceChangeListener`.

Code 6-7 : Gérer les événements des préférences partagées (suite)

```
SharedPreferences preferences = getSharedPreferences(MES_PREFERENCES,
            Context.MODE_PRIVATE);
      if (preferences != null)
         preferences.unregisterOnSharedPreferenceChangeListener(...);
```

Les menus de préférences prêts-à-l'emploi

Afin de faciliter la vie des développeurs et de ne pas réinventer la roue à chaque application, les développeurs de la plate-forme Android ont créé un type d'activité de présentation et de persistance des paramètres de l'application. Cette activité spécifique, qui hérite de `ListActivity`, permet de créer très rapidement des menus similaires à ceux des paramètres système d'Android.

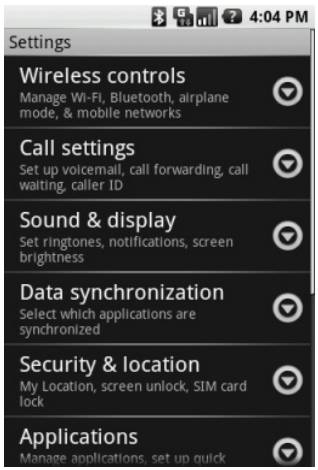


Figure 6–1
Le menu de préférences de l'émulateur Android

L'activité `PreferenceActivity` permet de construire l'interface de votre menu de préférences de plusieurs façons :

- directement dans un fichier XML que vous placerez dans les ressources de votre projet, par exemple à l'emplacement `/res/xml/preferences.xml`, et dans lequel vous spécifiez la hiérarchie des préférences ;
- en utilisant plusieurs activités pour lesquelles vous avez spécifié des métadonnées dans le fichier `manifest.xml` ;
- depuis une hiérarchie d'objets dont la racine est de type `PreferenceScreen`.

En utilisant ce type d'activité, vous facilitez l'évolution de votre application et réduisez l'effort que vous allez devoir fournir pour enregistrer et lire les paramètres de l'application. Chaque paramètre est enregistré automatiquement et vous pourrez retrouver chacune des valeurs en appelant la méthode statique `getDefaultSharedPreferences` du gestionnaire de préférences `PreferenceManager`.

Les étapes nécessaires pour implémenter une activité de préférences (via l'utilisation d'un fichier XML) sont les suivantes.

1. Créer un fichier XML avec la structure des paramètres.
2. Créer une classe dérivant de `PreferenceActivity`.
3. Charger la structure XML des paramètres.
4. Récupérer les valeurs de chaque paramètre à l'aide de `getDefaultSharedPreferences`.

Nous ne pouvons que vous conseiller de réaliser vos menus de préférences de cette façon, cela permet de garder une homogénéité visuelle des menus de paramètres pour l'utilisateur.

Créer un menu de préférences

La première étape consiste à créer un fichier XML décrivant comment sera présenté votre menu de préférences à l'utilisateur. L'exemple suivant décrit la hiérarchie de préférences utilisateur contenant une liste de pays, une case à cocher et une zone de texte :

Code 6-8 : Description d'une activité de préférences

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
   <ListPreference android:key="preferences_pays"
      android:title="Liste de pays" android:summary="Veuillez sélectionner votre
pays"
      android:dialogTitle="Sélectionner votre pays :"
      android:entries="@array/pays_liste_preferences"
      android:entryValues="@array/pays_liste_valeurs_preferences"/>
   <CheckBoxPreference android:key="checkBoxDemo"
      android:title="Démo case à cocher" android:summary="Checkbox summary"
      android:summaryOn="Cette case est activée, décochez-la pour la désactiver."
      android:summaryOff="Cette case est désactivée, cochez-la pour l'activer."/>
   <EditTextPreference android:key="editTextDemo"
      android:title="Démo texte" android:summary="Démo zone de texte"
      android:dialogTitle="Veuillez saisir un texte"
android:dependency="checkBoxDemo"/>
</PreferenceScreen>
```

La racine des préférences XML est un élément `PreferenceScreen`, représentant un écran – nous détaillerons ce concept plus loin – de paramètres. Tous les éléments que vous allez trouver en dessous de cette racine sont des déclarations de différents types

de préférences : `CheckBoxPreference`, `EditTextPreference`, `ListPreference`, `RignTonePreference`, etc. Chacun de ces éléments affichera un contrôle du type approprié avec un titre et une description spécifiés dans les attributs de l'élément dans l'écran des préférences.

Chaque préférence, quel que soit son type, doit posséder une clé qui sert d'identifiant pour l'enregistrement de chaque valeur. C'est l'attribut `android:key` de la préférence qui permet de spécifier cette propriété. N'hésitez pas à mettre la valeur de la clé dans un fichier de valeur de type `String` afin de pouvoir retrouver cette valeur plus facilement directement depuis le code :

```
<EditTextPreference android:key="@string/preferences_nom_cle" … />
```

Les attributs `android:title` et `android:summary` permettent respectivement de spécifier le titre et la description de la préférence. En plus de ces propriétés communes, il existe un ensemble de propriétés spécifiques que nous allons détailler.

Pour pouvoir utiliser la description XML d'un menu dans votre application, vous devez créer une classe héritant de `PreferenceActivity` et charger le menu de préférences depuis la méthode de création de l'activité :

Code 6-9 : Création d'une activité de préférences

```
public class PreferenceActivityImpl extends PreferenceActivity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      addPreferencesFromResource(R.xml.preferences);
   }
}
```

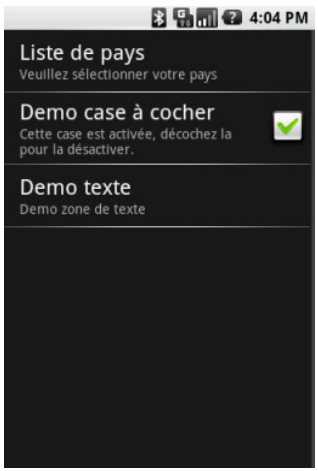


Figure 6–2
Le menu de l'exemple

La méthode `addPreferencesFromRessource` ajoute le menu de préférences à l'activité en s'appuyant sur le fichier XML spécifié.

Changez le fichier de configuration de l'application de façon à lancer l'activité de préférences et exécutez l'application pour afficher le menu de la figure 6-2.

Préférences de type liste

L'objectif du type `ListPreference` est de proposer une liste de valeurs à l'utilisateur parmi lesquelles il ne pourra en sélectionner qu'une seule. Une liste possède des propriétés permettant de définir quels seront les éléments présentés à l'utilisateur :

- `dialogTitle` spécifie le titre de la boîte de dialogue affichant la liste à l'utilisateur. Vous pouvez spécifier une chaîne ou une référence vers une ressource de type chaîne. Cette propriété est commune à d'autres types de préférences ;
- `entries` spécifie les éléments qui seront présentés à l'utilisateur. Vous devez spécifier une référence vers une ressource de type tableau ;
- `entryValues` spécifie les valeurs qui seront enregistrées dans les préférences.

Voici un exemple de `ListPreference` permettant de sélectionner un pays :

Code 6-10 : Ajout d'une liste de choix de pays

```
<ListPreference android:key="preferences_pays"
android:title="Liste de pays" android:summary="Veuillez sélectionner
votre pays" android:dialogTitle="Sélectionner votre pays :"
android:entries="@array/pays_liste_preferences"
android:entryValues="@array/pays_liste_valeurs_preferences"/>
```

Les éléments de la liste de l'exemple font référence à une ressource localisée dans `res/values/array.xml` contenant le fichier XML suivant :

```
<resources>
    <array name="pays_liste_preferences">
        <item>Angleterre</item>
        <item>Australie</item>
        <item>France</item>
        <item>Nouvelle-Calédonie</item>
        <item>Nouvelle-Zélande</item>
    </array>

    <array name="pays_liste_valeurs_preferences">
        <item>England</item>
       <item>Australia</item>
        <item>France</item>
        <item>New Caledonia</item>
        <item>New Zeland</item>
```

```
    </array>
</resources>
```

Figure 6–3
La liste des pays du menu d'exemple

Paramètre de type case à cocher

La préférence de ce type propose à l'utilisateur une case à cocher pouvant prendre deux valeurs : activée ou désactivée.

Code 6-11 : Ajout d'une préférence de type case à cocher

```
<CheckBoxPreference android:key="checkBoxDemo"
      android:title="Démo case à cocher"
      android:summaryOn="Cette case est activée, décochez-la pour la désactiver."
      android:summaryOff="Cette case est désactivée, cochez-la pour l'activer."/>
```

Comme le montre le code 6-11, ce type de préférence possède des propriétés spécifiques permettant de personnaliser l'affichage de façon plus intuitive pour l'utilisateur :

- `summaryOn` spécifie la description lorsque la case à cocher est activée ;
- `summaryOff` spécifie la description lorsque la case à cocher est désactivée.

Ce type de préférence est opportun pour utiliser la propriété de dépendance `android:dependency`. Cette propriété permet à une préférence de spécifier la clé d'une autre préférence. Si la valeur de la préférence cible est remplie ou vraie - dans le cadre d'une case à cocher, par exemple -, la préférence dépendante sera alors active, sinon elle sera désactivée.

Voici un exemple rapide de cette propriété :

Code 6-12 : Dépendance d'une préférence envers une autre

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
   <CheckBoxPreference android:key="checkBoxDemo"
      android:title="Démo case à cocher" android:summary="Checkbox summary"
      android:summaryOn="Cette case est activée, décochez-la pour la désactiver."
      android:summaryOff="Cette case est désactivée, cochez'la pour l'activer."/>
   <EditTextPreference android:key="editTextDemo"
      android:title="Démo texte" android:summary="Démo zone de texte"
      android:dialogTitle="Veuillez saisir un texte"
android:dependency="checkBoxDemo"/>
</PreferenceScreen>
```

Cette dépendance active ou désactive la préférence l'ayant spécifiée en fonction de la valeur de la préférence ciblée. Dans l'exemple du code 6-12, lorsque la case est cochée, la zone de texte est active, sinon elle est désactivée.

Figure 6–4 Case à cocher activée

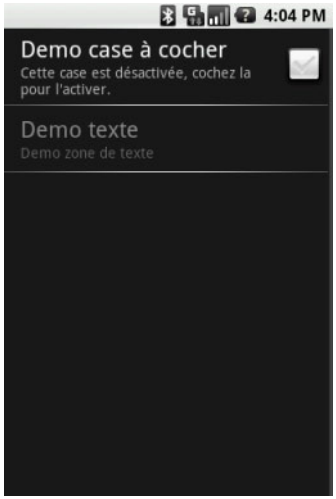


Figure 6–5 Case à cocher désactivée - la préférence de type texte est maintenant désactivée.

Autres types de préférences : zone de texte, sélection de sonnerie...

La zone de texte

Ce type de paramètre permet à l'utilisateur de saisir une valeur de type chaîne de caractères. Lorsqu'il clique sur cette préférence, une boîte de dialogue apparaît avec une zone de saisie de texte.

Code 6-13 : Ajout d'une préférence de type zone de texte

```
<EditTextPreference android:key="editTextDemo"
android:title="Démo texte" android:summary="Démo zone de texte"
android:dialogTitle="Veuillez saisir un texte"/>
```

Figure 6–6
Saisie d'une préférence de type texte

Sélection de sonnerie

Quelquefois vous aurez besoin de paramètres nécessitant la sélection d'une sonnerie, par exemple pour attirer l'attention de l'utilisateur lors d'un événement précis de votre application. L'élément à déclarer dans la structure XML du menu de préférences sera celui du code 6-14 suivant.

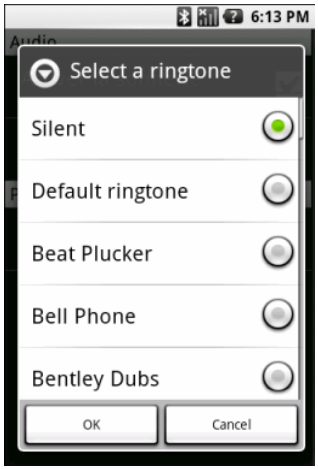


Figure 6–7
Sélection d'une sonnerie dans le menu de préférences

Code 6-14 : Ajout d'une préférence de type sélection de sonnerie

```
<RingtonePreference android:key="Sonnerie"
   android:title="Sonnerie" android:showDefault="true"
   android:dependency="AutoriserSonnerie"
   android:showSilent="true"
   android:summary="Choisissez une sonnerie" />
```

Diviser pour optimiser la navigation parmi les préférences

Si votre application comporte beaucoup de préférences, proposer un seul et unique menu incluant l'ensemble des paramètres de votre application risque de donner une fausse impression de complexité. L'utilisateur ne verra pas forcément la logique entre vos préférences et sera ainsi contraint de faire défiler l'écran pour trouver le paramètre qu'il souhaite modifier. Heureusement, la plate-forme Android offre la possibilité de découper visuellement l'affichage des préférences en les scindant en catégories et en écrans imbriqués.

Les catégories de préférences

Les catégories sont utilisées pour regrouper les préférences entre elles et rendre l'organisation des menus plus logiques et ergonomiques. Au niveau de la description XML de votre menu, au lieu d'avoir toutes vos préférences rattachées à la racine `PreferenceScreen`, vous pourrez les imbriquer dans un élément `PreferenceCategory` :

Code 6-15 : Ajout d'une catégorie dans une activité de préférences

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
   <PreferenceCategory android:title="Audio">
      <CheckBoxPreference android:key="AutoriserSonnerie"
         android:title="Activer la sonnerie" android:summary="Activez la
sonnerie pour être notifié !" />
      <RingtonePreference android:key="Sonnerie"
         android:title="Sonnerie" android:showDefault="true"
         android:dependency="AutoriserSonnerie" android:showSilent="true"
         android:summary="Choisissez une sonnerie" />
   </PreferenceCategory>
   <PreferenceCategory android:title="Préférences avancées">
      <CheckBoxPreference android:key="PreferenceAvancee"
         android:title="Préférences avancées" android:summary="Ma préférence
avancée ..." />
   </PreferenceCategory>
</PreferenceScreen>
```

Spécifiez le titre de la catégorie, qui s'affiche à l'écran, à l'aide de l'attribut `title`.

Visuellement, cela ajoute une séparation entre les différentes catégories ce qui permet à l'utilisateur de mieux identifier et retrouver les différents paramètres qui lui sont proposés par thème.

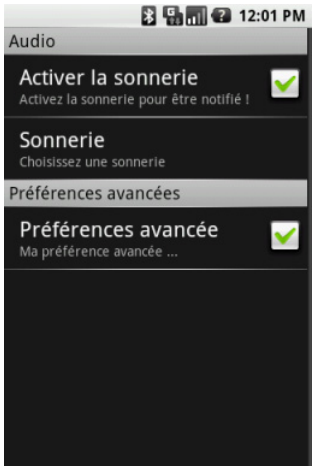


Figure 6–8
Écran de préférences avec catégories

Les écrans de préférences imbriqués

Si les catégories ne suffisent plus ou que vous souhaitez diviser votre menu de préférences de façon logique en plusieurs écrans, vous avez la possibilité d'utiliser des éléments `PreferenceScreen` imbriqués. Chaque groupe de préférences ainsi assemblé dans un élément `PreferenceScreen` imbriqué sera affiché dans son propre écran. L'écran du parent affichera un élément avec un lien vers ces écrans de préférences détaillées.

Voici un exemple de la structure XML de préférences possédant un écran principal des catégories et un écran imbriqué.

Code 6-16 : Exemple d'imbrication d'écrans de préférences

```
<?xml version="1.0" encoding="UTF-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
   <PreferenceCategory android:title="Audio">
      <CheckBoxPreference android:key="AutoriserSonnerie"
         android:title="Activer la sonnerie" android:summary="Activez la
sonnerie pour être notifié !" />
      <RingtonePreference android:key="Sonnerie"
         android:title="Sonnerie" android:showDefault="true"
         android:dependency="AutoriserSonnerie" android:showSilent="true"
         android:summary="Choisissez une sonnerie" />
   </PreferenceCategory>
   <PreferenceCategory android:title="Préférences avancées">
      <PreferenceScreen android:key="EcranDetails" android:title="Avancé"
```

```
        android:summary="Préférences avancées">
        <CheckBoxPreference android:key="PreferenceAvancee"
          android:title="Préférences avancées" android:summary="Ma préférence
avancée ..." />
      </PreferenceScreen>
   </PreferenceCategory>
</PreferenceScreen>
```

Le résultat se traduit par l'apparition d'une icône permettant à l'utilisateur de naviguer dans le sous-écran de préférences.

Figure 6–9 Écran principal des paramètres de l'application

Figure 6–10 L'écran des paramètres imbriqués, fils du menu principal

Stockage dans des fichiers

Le fichier représente l'élément de base du système Android pour stocker n'importe quel type de données : applications, ressources, bases de données, etc.

Le paradigme de développement des applications Android prévoit que toutes les ressources soient stockées et regroupées dans le dossier `/res`. Malgré cela, il y aura toujours des cas où vous ne pourrez pas éviter l'utilisation de fichiers de données bruts (fichier de données sérialisées, texte brut, etc.). Heureusement, la plate-forme Android prévoit des API pour faciliter la vie des développeurs.

Lire, écrire et supprimer dans le système de fichiers

En plus des API standards de manipulation de l'espace de noms `java.io`, Android propose deux méthodes, `openFileOutput` et `openFileInput`, pour simplifier la manipulation des fichiers depuis un contexte local.

La méthode `openFileOutput` permet d'ouvrir le fichier en écriture ou de le créer s'il n'existe pas. Si le fichier existe, le comportement par défaut consiste à l'écraser. Si vous souhaitez ajouter du contenu à un fichier déjà existant, vous devez spécifier le mode `MODE_APPEND`.

Code 6-17 : Ouverture d'un fichier en écriture

```
private static final String NOM_FICHIER = "MonFichier.dat";
...
try {
   // Crée un flux de sortie vers un fichier local.
   FileOutputStream ous = openFileOutput(NOM_FICHIER, MODE_PRIVATE);
   ...
} catch (FileNotFoundException e) {
   // Traitement des exceptions ...
}
```

La méthode `openFileInput` permet d'ouvrir le fichier en lecture.

Code 6-18 : Ouverture d'un fichier en lecture

```
private static final String NOM_FICHIER = "MonFichier.dat";
...
try {
   // Crée un flux d'entrée depuis un fichier local.
   FileInputStream ins = openFileInput(NOM_FICHIER);
   ...
} catch (FileNotFoundException e) {
   // Traitement des exceptions ...
}
```

Ces méthodes ne prennent en charge que les noms de fichiers localisés dans le répertoire de l'application. Il est impossible de spécifier des séparateurs de chemin, sauf à risquer de voir une exception se lever.

> **VOUS VENEZ DE JAVA Méthodes et classes de manipulation de fichiers**
>
> Un développeur Java ne sera pas dérouté par les méthodes ou classes pour manipuler les fichiers avec Android, il s'agit des mêmes API - dans une version légèrement limitée - que celles de Java Standard Edition.

Pour supprimer un fichier du système de fichiers, utilisez la méthode `deleteFile` qui permet d'effacer un fichier à partir de son nom.

Partager un fichier avec d'autres applications

Par défaut, les fichiers créés par la méthode `openFileOutput` sont restreints à l'application. Si vous voulez partager un fichier avec une autre application, la bonne pratique reste l'utilisation d'un fournisseur de contenu. Si une autre application essaye d'accéder aux fichiers locaux de votre application, son accès lui sera refusé.

Néanmoins, vous pouvez spécifier un mode d'ouverture tel que `MODE_WORLD_READABLE` ou `MODE_WORLD_WRITABLE` permettant aux autres applications de pouvoir accéder à vos fichiers.

Afin de mieux comprendre le rôle de chaque permission, voici un résumé des modes d'accès possibles (qui ne sera pas sans vous rappeler celle des préférences partagées, également stockées dans un fichier) :

- `MODE_PRIVATE` est le mode par défaut et signifie que le fichier créé sera accessible uniquement à l'application l'ayant créé ;
- `MODE_WORLD_READABLE` permet aux autres applications de pouvoir lire le fichier mais pas de pouvoir le modifier ;
- `MODE_WORLD_WRITABLE` permet aux autres applications de pouvoir lire et également modifier le fichier ;
- `MODE_APPEND` permet d'écrire les données en fin de fichier au lieu de l'écraser. Vous pouvez combiner ce mode avec un autre (par exemple, `MODE_PRIVATE | MODE_APPEND`).

Intégrer des ressources dans vos applications

Une application nécessite quelquefois d'embarquer des fichiers de façon à pouvoir charger des données pendant son exécution. Par exemple, dans le cadre d'un jeu, pouvoir lire les données des niveaux de jeu, les menus, etc., devient vite indispensable.

Android répond élégamment à cette exigence en vous permettant d'embarquer des données directement dans le fichier `.apk` de votre application. L'utilisation de ce mécanisme vous permet également de garder l'avantage de la gestion des ressources alternatives : langue, régionalisation et configuration matérielle. Vous pourrez de cette façon proposer des sons ou d'autres ressources qui s'adapteront automatiquement au matériel et à la langue de vos utilisateurs.

Pour embarquer des données dans votre application, placez simplement les ressources brutes dans le répertoire `res/raw` de votre projet Eclipse. Les fichiers embar-

qués seront ainsi accessibles en lecture seule par votre application grâce à la méthode `openRawResource`.

Code 6-19 : Ouvrir une ressource d'une application

```
Resources res = getResources();
InputStream ins = res.openRawResource(R.raw.niveau_1_obstacles);
```

> **ATTENTION Taille du fichier .apk**
>
> Ajouter des fichiers bruts est utile lorsque vous souhaitez embarquer des données déjà traitées (dictionnaires, contenu en texte brut, etc.). Cependant, gardez bien à l'esprit que plus vos données sont volumineuses, plus le fichier `.apk` que vous distribuerez le sera également.

> **ERGONOMIE Tester les performances sur un téléphone de test plutôt que l'émulateur**
>
> Si vos opérations sur les fichiers sont complexes et fréquentes, surveillez de près les performances car vous risqueriez de rendre l'application trop peu réactive et de frustrer l'utilisateur. Pour ces tests, l'utilisation d'un matériel physique plutôt que l'émulateur est fortement recommandé.

Gérer les fichiers

La plate-forme Android propose plusieurs méthodes utilitaires pour manipuler les fichiers depuis le contexte de l'application :

- la méthode `fileList` liste tous les fichiers locaux de l'application ;
- la méthode `getFileDir` permet d'obtenir le chemin absolu du répertoire dans le système de fichiers où tous les fichiers créés avec `openFileOutput` sont stockés ;
- la méthode `getFileStreamStore` retourne le chemin absolu du répertoire dans le système de fichiers où est stocké le fichier créé avec `openFileOutput` et dont le nom est passé en paramètre.

L'utilisation des fichiers dans une application Android n'est souvent pas le meilleur moyen d'arriver à vos fins. Néanmoins, il y a bien des cas où vous ne pourrez pas faire autrement. Si vous souhaitez exposer le contenu d'un fichier, ou d'une base de données (détaillée dans la prochaine partie), privilégiez le recours à un fournisseur de contenu qui offrira plus de sécurité et de contrôle sur les données exposées.

Quelquefois, il sera plus opportun pour vous de ne pas embarquer les données brutes directement dans vos applications mais de les télécharger après coup à la demande et d'enregistrer celles-ci dans le système de fichiers. Cela évitera à l'utilisateur d'installer une application trop lourde qui pourrait le rebuter au premier abord, et d'optimiser l'espace disque en ne téléchargeant que les données dont l'utilisateur a véritablement besoin (uniquement sa langue, etc).

Stockage dans une base de données SQLite

L'avantage d'une base de données est qu'elle permet de manipuler et de stocker des données complexes et structurées, ce qui serait impossible, ou du moins difficile à faire, avec les autres moyens de persistance décrits précédemment.

Android fournit un support de bases de données relationnelles au travers de SQLite, une base de données très utilisée dans le domaine des appareils mobiles (lecteurs mp3, lecteurs de salon, etc.).

> **CULTURE SQLite**
>
> SQLite est une base de données relationnelle dont les principales caractéristiques sont les suivantes : elle est légère, gratuite et Open Source.
> ▸ http://www.sqlite.org

À la différence de bon nombre d'autres bases de données, SQLite s'exécute sans nécessiter de serveur, ce qui implique que l'exécution des requêtes sur la base de données s'effectue dans le même processus que l'application. Conjugué au fait qu'une base de données est réservée à l'application créatrice, cela rend une base de données SQLite performante au niveau de la gestion des transactions et de la synchronisation.

Vous pouvez créer plusieurs bases de données par application. Néanmoins, chaque base de données est dédiée à l'application, c'est-à-dire que seule l'application qui en est à l'origine pourra y accéder. Si vous souhaitez exposer les données d'une base de données particulière à d'autres applications, vous pourrez utiliser un fournisseur de contenu, comme nous le verrons au chapitre 7.

> **EN COULISSES Chemin de stockage des bases de données par défaut**
>
> Toutes les bases de données sont stockées par défaut dans `/data/data/<espace de noms>/databases` sur votre appareil. Le fichier de bases de données est automatiquement créé en `MODE_PRIVATE`, d'où le fait que seule l'application l'ayant créé peut y accéder.

Concevoir une base de données SQLite pour une application Android

On ne conçoit pas et on n'utilise pas une base de données SQLite de la même façon que l'on pourrait le faire avec une base de données dédiée à un serveur (PostgreSQL, MySQL, etc.). Gardez en tête que vous développez pour des appareils mobiles, avec peu d'espace de stockage, de mémoire vive et de puissance. Évitez donc de mettre des volumes de données importants dans vos bases ou d'effectuer des requêtes fréquentes (attention à la gestion des ressources des curseurs).

De structures simples résultent des requêtes simples et des données facilement identifiables. Faites en sorte de concevoir une base avec une structure simple et extensible, comportant des données facilement identifiables de façon à ce qu'une requête ne renvoie pas de données inutiles en créant du « bruit » dans vos données. Il en va de la performance de votre application.

SQLite étant une base de données légère, vous devrez éviter d'y enregistrer des données binaires (images, par exemple). À la place, créez des fichiers sur le support de stockage et faites-y référence dans la base. Puisque vous pourriez avoir besoin d'exposer ces données binaires à d'autres applications, une bonne pratique consiste à nommer ces fichiers avec un URI de façon à être directement renvoyé par un fournisseur de contenu et exploité par l'application demandeuse.

Autre élément à prendre en considération, l'ajout systématique d'une colonne d'index auto-incrémentable de façon à présenter chaque ligne de façon unique. C'est un bon conseil, surtout si vous envisagez par la suite de partager vos tables via un fournisseur de contenu (voir le chapitre 7 à ce sujet).

Créer et mettre à jour la base de données SQLite

Créer une base de données SQLite est une opération simple, commune à toutes les bases de données. Néanmoins, mettre à jour le schéma de la base de données est une opération plus délicate. Afin de simplifier le code de votre application pour gérer ces opérations, le SDK Android offre une classe d'aide : `SQLiteOpenHelper`.

Pour pouvoir utiliser cette classe d'aide, celle-ci doit être dérivée de façon à personnaliser les méthodes nécessaires à votre application. Parmi ces méthodes, vous trouverez une méthode de création `onCreate`, une méthode de mise à jour `onUpgrade` et une méthode pour ouvrir la base de données en toute simplicité.

La force de ce modèle de conception est qu'une fois que vous aurez dérivé cette classe, toutes ces opérations seront transparentes pour vous.

Code 6-20 : Squelette de la classe d'aide d'une base de données SQLite

```
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase.CursorFactory;

class MaBaseOpenHelper extends SQLiteOpenHelper {

   public MaBaseOpenHelper(Context context, String nom, CursorFactory
cursorfactory, int version)
   {
```

```
      super(context, nom, cursorfactory, version);
   }

   @Override
   public void onCreate(SQLiteDatabase db) {
      // TODO Ajoutez votre code de création ici ...
   }

   @Override
   public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
      // TODO Ajoutez votre code de mise à jour ici ...
   }
}
```

En utilisant la classe dérivée de `SQLiteOpenHelper`, si vous essayez d'ouvrir une base alors qu'elle n'existe pas encore, la classe la créera pour vous en appelant la méthode `onCreate` que vous aurez redéfinie. Si la version de la base de données a changé, alors la méthode `onUpgrade` sera aussi appelée.

Code 6-21 : Création d'une classe d'aide à la création/mise à jour d'une base de données

```
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase.CursorFactory;

class MaBaseOpenHelper extends SQLiteOpenHelper {

   private static final String TABLE_PLANETES = "table_planetes";

   private static final String COLONNE_ID = "id";
   private static final String COLONNE_NOM = "nom";
   private static final String COLONNE_RAYON = "rayon";

   private static final String REQUETE_CREATION_BD = "create table "
         + TABLE_PLANETES + " (" + COLONNE_ID
         + " integer primary key autoincrement, " + COLONNE_NOM
         + " text not null, " + COLONNE_RAYON + " text not null);";

   public MaBaseOpenHelper(Context context, String nom,
         CursorFactory cursorfactory, int version) {
      super(context, nom, cursorfactory, version);
   }

   @Override
   public void onCreate(SQLiteDatabase db) {
      db.execSQL(REQUETE_CREATION_BD);
   }
```

```
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // Dans notre cas, nous supprimons la base et les données pour en créer
        // une nouvelle ensuite. Vous pouvez créer une logique de mise à jour
        // propre à votre base permettant de garder les données à la place.
        db.execSQL("DROP TABLE" + TABLE_PLANETES + ";");
        // Création de la nouvelle structure.
        onCreate(db);
    }
}
```

Pour accéder à une base de données à l'aide de la classe `SQLiteOpenHelper`, vous pouvez appeler les méthodes `getReadableDatabase` et `getWritableDatabase` et ainsi obtenir une instance de la base de données respectivement en lecture seule et en lecture/écriture.

Code 6-22 : Ouvrir une base de données en lecture et lecture/écriture

```
MaBaseOpenHelper monHelper = new MaBaseOpenHelper(context,
NOM_BASE_DONNEES, null, VERSION_BASE_DONNEES);
SQLiteDatabase db = monHelper.getWritableDatabase();
...
SQLiteDatabase db = monHelper.getWReadableDatabase();
```

Accéder à une base de données

La partie précédente nous a montré comment le SDK Android nous facilite la tâche pour créer une classe d'aide à l'ouverture, à la création et à la mise à jour d'une base de données SQLite. Maintenant, nous allons utiliser cette classe d'aide pour travailler avec la base qui sera créée automatiquement grâce à celle-ci.

La manipulation d'une base de données est toujours quelque peu fastidieuse et la bonne pratique veut que vous proposiez une couche d'abstraction, afin de proposer des méthodes d'accès et de manipulation simplifiées à la base, et d'apporter une indépendance vis-à-vis de la source de données. Si, par exemple, vous souhaitez par la suite changer de type de source de données et travailler avec des services Internet ou des fichiers, vous ne devrez changer que le code de cette classe d'abstraction.

Ce modèle de conception est appelé *adaptateur* et propose d'offrir au développeur des méthodes fortement typées pour effectuer des requêtes, insérer, supprimer et mettre à jour des données de la base mais aussi pour gérer les opérations d'ouverture et de fermeture de la base de données. Nous avons déjà les derniers éléments grâce à notre classe dérivée de `SQLiteOpenHelper`, nous allons maintenant encapsuler toutes ces actions dans notre adaptateur.

Code 6-23 : Création de l'adaptateur de la base de données

```
public class PlanetesDBAdaptateur {
   private static final int BASE_VERSION = 1;
   private static final String BASE_NOM = "planetes.db";

   private static final String TABLE_PLANETES = "table_planetes";

   private static final String COLONNE_ID = "id";
   private static final int COLONNE_ID_ID = 0;
   private static final String COLONNE_NOM = "nom";
   private static final int COLONNE_NOM_ID = 1;
   private static final String COLONNE_RAYON = "rayon";
   private static final int COLONNE_RAYON_ID = 2;

   // La requête de création de la structure de la base de données.
   private static final String REQUETE_CREATION_BD = "create table "
         + TABLE_PLANETES + " (" + COLONNE_ID
         + " integer primary key autoincrement, " + COLONNE_NOM
         + " text not null, " + COLONNE_RAYON + " text not null);";

   // L'instance de la base qui sera manipulée au travers de cette classe.
   private SQLiteDatabase maBaseDonnees;

   private MaBaseOpenHelper baseHelper;

   public PlanetesDBAdaptateur(Context ctx) {
      baseHelper = new MaBaseOpenHelper(ctx, BASE_NOM, null, BASE_VERSION);
   }

   /**
    * Ouvre la base de données en écriture.
    */
   public SQLiteDatabase open() {
      maBaseDonnees = baseHelper.getWritableDatabase();
      return maBaseDonnees;
   }

   /**
    * Ferme la base de données.
    */
  public void close() {
      maBaseDonnees.close();
   }

   public SQLiteDatabase getBaseDonnees() {
      return maBaseDonnees;
   }
```

```
 /**
  * Récupère une planète en fonction de son nom.
  */
 public Planete getPlanete(String nom) {
    // Insérer le code de requête d'une planète.
 }

 /**
  * Récupère une planète en fonction de son id.
  */
public Planete getPlanete(int id) {
    // Insérer le code de requête d'une planète.
 }

 /**
  * Retourne toutes les planètes de la base de données.
  */
public ArrayList<Planete> getAllPlanetes() {
    // Insérer le code de requête de l'ensemble des planètes de la base.
 }

 /**
  * Insère une planète dans la table des planètes.
  */
public long insertPlanete(Planete planete) {
    // Insérer le code d'insertion.
 }

 /**
  * Met à jour une planète dans la table des planètes.
  */
 public int updatePlanete(int id, Planete planeteToUpdate) {
    // Insérer le code de mise à jour de la base.
 }

 /**
  * Supprime une planète à partir de son nom.
  */
 public boolean removePlanete(String nom) {
    // Insérer le code de suppression d'une planète.
 }

 /**
  * Supprime une planète à partir de son id.
  */
 public boolean removePlanete(int id) {
    // Insérer le code de suppression d'une planète.
 }
```

```
    private class MaBaseOpenHelper extends SQLiteOpenHelper {
        ...
    }
}
```

De la même façon qu'il est une bonne pratique de proposer un adaptateur pour les accès aux sources de données, il en est aussi de pouvoir proposer des classes typées afin de manipuler les données de façon naturelle dans vos applications. Dans le code précédent, nous faisions référence à une classe `Planete` représentant un objet métier décrivant les propriétés, non exhaustives, d'une planète.

Code 6-24 : Création d'une classe typée pour manipuler les données issues de la base

```
public class Planete {

...private int id;

...private String nom;

...private float rayon;

...public Planete() { }

...public Planete(String nom, float rayon)
...{
......this.nom = nom;
......this.rayon = rayon;
...}

...public int getId() {
......return id;
...}

...public void setId(int id) {
......this.id = id;
...}

...public String getNom() {
......return nom;
}

...public void setNom(String nom) {
......this.nom = nom;
...}

...public float getRayon() {
......return rayon;
...}
```

```
...public void setRayon(float rayon) {
......this.rayon = rayon;
...}
}
```

Effectuer une requête dans une base SQLite

Toutes les requêtes de sélection SQLite s'effectuent via la méthode `query` d'une instance de `SQLiteDatabase`. Cette méthode retourne un curseur permettant ensuite de naviguer dans les résultats. Une attention particulière est nécessaire pour la bonne gestion des ressources allouées au travers de ce dernier.

À la différence de nombreuses API pour d'autres bases de données, les requêtes de sélection ne se font pas à partir d'une chaîne `SELECT` mais par l'utilisation de la méthode `query` – et de ses surcharges – proposant directement les critères de sélection au développeur.

La requête de sélection SQL sera construite par la méthode, puis compilée pour interroger la base de données. Les paramètres de la requête sont donc très proches d'une requête `SELECT` standard.

Tableau 6–1 Paramètre de la méthode de requête de sélection query()

Nom du paramètre	Description
`distinct`	Spécifie si le résultat doit contenir ou non des éléments uniques. Optionnel.
`table`	Spécifie le nom de la table à requêter.
`columns`	Spécifie la projection de la requête, c'est-à-dire le nom des colonnes de la table à inclure dans le résultat. Si vous spécifiez une valeur `null`, toutes les colonnes de la table seront retournées. Cependant, essayez de réduire au maximum votre projection aux seules colonnes qui seront utilisées de façon à réduire les ressources nécessaires. Vous spécifierez un tableau de `String` pour nommer les colonnes.
`selection`	Spécifie la clause de filtre `WHERE` de la requête. Le format doit être identique à une requête `WHERE` standard. Si vous spécifiez la valeur `null`, tous les éléments seront retournés. Vous pouvez utiliser des valeurs `'?'` dans la sélection, chacune sera remplacée par la valeur spécifiée dans le paramètre `selectionArgs`.
`selectionArgs`	Spécifie les valeurs de remplacement des `'?'` dans le paramètre `selection` dans l'ordre d'apparition.
`groupBy`	Un filtre de regroupement des lignes, similaire à la clause `GROUP BY`. Si vous spécifiez la valeur `null`, les lignes ne seront pas regroupées.
`having`	Un filtre de condition d'apparition des lignes en relation avec le paramètre `groupBy` et similaire à la clause `GROUP BY` standard. Si vous spécifiez la valeur `null`, toutes les lignes seront incluses dans le résultat. La valeur `null` est requise si la valeur du paramètre `groupBy` est également `null`.

Tableau 6–1 Paramètre de la méthode de requête de sélection query() (suite)

Nom du paramètre	Description
`orderBy`	Spécifie l'ordre de tri des lignes selon le format de la clause `ORDER BY` standard. Si vous spécifiez la valeur `null`, l'ordre de tri par défaut sera appliqué.
`limit`	Limite le nombre de lignes retournées par la requête en utilisant le format de la clause `LIMIT` standard. Si vous spécifiez une valeur `null`, toutes les lignes seront retournées.

Le code suivant utilise la méthode `query` pour retrouver les différentes planètes de notre exemple :

Code 6-25 : Requête des données de la base de données

```
...

private static final String TABLE_PLANETES = "table_planetes";

public static final String COLONNE_ID = "id";
public static final int COLONNE_ID_ID = 0;
public static final String COLONNE_NOM = "nom";
public static final int COLONNE_NOM_ID = 1;
public static final String COLONNE_RAYON = "rayon";
public static final int COLONNE_RAYON_ID = 2;

/**
* Récupère une planète en fonction de son nom.
*/
public Planete getPlanete(String nom) {
   Cursor c = maBaseDonnees.query(TABLE_PLANETES, new String[] {
         COLONNE_ID, COLONNE_NOM, COLONNE_RAYON }, null, null, null,
         COLONNE_NOM + " LIKE " + nom, null);
   return cursorToPlanete(c);
}

/**
* Récupère une planète en fonction de son id.
*/
public Planete getPlanete(int id) {
   Cursor c = maBaseDonnees.query(TABLE_PLANETES, new String[] {
         COLONNE_ID, COLONNE_NOM, COLONNE_RAYON }, null, null, null,
         COLONNE_ID + " = " + id, null);
   return cursorToPlanete(c);
}

/**
* Retourne toutes les planètes de la base de données.
*/
```

```
public ArrayList<Planete> getAllPlanetes() {
   Cursor c = maBaseDonnees.query(TABLE_PLANETES, new String[] {
         COLONNE_ID, COLONNE_NOM, COLONNE_RAYON }, null, null, null,
         null, null);
   return cursorToPlanetes(c);
}

...
```

Les méthodes `cursorToPlanetes` et `cursorToPlanete` permettent de transformer le curseur de résultat en objet métier. Ces deux méthodes sont décrites dans la partie suivante.

Récupérer des éléments issus du résultat d'une requête de sélection se fait au travers de l'objet `Cursor` retourné par la méthode `query`. Au lieu de retourner tous les résultats, le `Cursor` agit comme un curseur de base de données accessible directement depuis les API. De cette façon, vous vous déplacez à votre guise au sein des résultats en modifiant la position de celui-ci, via les méthodes fournies.

Tableau 6–2 Les méthodes de navigation de la classe Cursor

Nom	Description
`moveToFirst`	Déplace le curseur à la première position pour lire les données de la première ligne.
`moveToLast`	Déplace le curseur à la dernière position pour lire les données de la dernière ligne de la requête.
`moveToNext`	Déplace le curseur d'une position pour lire la ligne suivante.
`moveToPrevious`	Déplace le curseur d'une position pour lire la ligne précédente.
`moveToPosition(int)`	Déplace le curseur à la position indiquée.

D'autres méthodes sont nécessaires pour pouvoir récupérer les informations des résultats retournés.

Tableau 6–3 Méthodes d'information de la classe Cursor

Nom	Description
`getCount`	Retourne le nombre de lignes qui sont renvoyées par la requête.
`getColumnName(int)`	Retourne le nom de la colonne spécifiée par son index.
`getColumnNames`	Retourne un tableau de chaînes de caractères avec le nom de toutes les colonnes retournées.
`getColumnCount`	Retourne le nombre de colonnes renvoyées par la requête.

Pour récupérer une valeur depuis un curseur, naviguez au sein de celui-ci, puis utilisez l'une des méthodes `get` pour retrouver la valeur à partir de l'index de la colonne. Les valeurs n'étant que faiblement typées dans SQLite, vous pouvez utiliser n'importe quelle méthode `get` pour récupérer une valeur :

Code 6-26 : Extraction de valeurs d'un curseur

```
// Chaîne de caractères.
String maChaine = curseur.getString(COLONNE_N);
// Entier 32 bits.
int monEntier = curseur.getInt(COLONNE_N);
// Entier long 64 bits.
long monEntierLong = curseur.getLong(COLONNE_N);
// Tableau de bytes.
byte[] monBlob = curseur.getBlob(COLONNE_N);
// Nombre décimal flottant 32 bits.
float monFlottant = curseur.getFloat(COLONNE_N);
// Nombre décimal flottant 64 bits.
double monDouble = curseur.getDouble(COLONNE_N);
// Nombre entier court 16 bits.
short monEntierCourt = curseur.getShort(COLONNE_N);
```

De façon générale, vous aurez à parcourir l'intégralité du curseur pour retrouver tous les éléments retournés par la requête. Le squelette suivant vous permettra d'y parvenir.

Code 6-27 : Parcourir l'intégralité des lignes retournées par une requête SQLite

```
Cursor c = maBaseDeDonnees.query(..);
…
// On place le curseur au début en vérifiant qu'il contient des résultats.
if (c.moveToFirst() {
   do {
      ...
      int index = c.getInt(COLONNE_ID_ID);
      String nom = c.getString(COLONNE_NOM_ID);
      ...
   } while (c.moveToNext());
}
```

Agrémentons maintenant les méthodes de transformation de l'objet `Cursor` en objet métier de l'adaptateur de l'exemple de cette partie.

Code 6-28 : Transformation des données de la base vers une classe typée

```
/**
* Transforme un Cursor en objet de type 'Planete'.
*
```

```
* @param c
*            Le curseur à utiliser pour récupérer les données de la planète.
* @return Une instance d'une planète avec les valeurs du curseur.
*/
private Planete cursorToPlanete(Cursor c) {
   // Si la requête ne renvoie pas de résultat.
   if (c.getCount() == 0)
      return null;

   Planete retPlanete = new Planete();
   // Extraction des valeurs depuis le curseur.
   retPlanete.setId(c.getInt(COLONNE_ID_ID));
   retPlanete.setNom(c.getString(COLONNE_NOM_ID));
   retPlanete.setRayon(c.getFloat(COLONNE_RAYON_ID));
   // Ferme le curseur pour libérer les ressources.
   c.close();
   return retPlanete;
}

private ArrayList<Planete> cursorToPlanetes(Cursor c) {
   // Si la requête ne renvoie pas de résultat.
   if (c.getCount() == 0)
      return new ArrayList<Planete>(0);

   ArrayList<Planete> retPlanetes = new ArrayList<Planete>(c.getCount());
   c.moveToFirst();
   do {
      Planete planete = new Planete();
      planete.setId(c.getInt(COLONNE_ID_ID));
      planete.setNom(c.getString(COLONNE_NOM_ID));
      planete.setRayon(c.getFloat(COLONNE_RAYON_ID));
      retPlanetes.add(planete);
   } while (c.moveToNext());
   // Ferme le curseur pour libérer les ressources.
   c.close();
   return retPlanetes;
}
```

Afin de gérer au mieux les ressources prises par un `Cursor` en fonction de l'activité de l'utilisateur et donc du cycle de vie d'une activité, la classe `Activity` fournit une méthode nommée `startManagingCursor`. Lorsque vous n'avez plus besoin de gérer les résultats d'un `Cursor`, appelez la méthode `stopManagingCursor`, puis la méthode `close` du `Cursor` pour libérer les ressources du curseur.

Insérer des données

Insérer ou mettre à jour des données dans une base SQLite repose sur l'utilisation de la méthode `insert` de la classe `SQLiteDatabase`. Pour spécifier les valeurs de la ligne à insérer, la méthode accepte un objet de type `ContentValues`. Cet objet stocke les valeurs de chaque colonne de la ligne à insérer sous la forme d'une collection d'associations entre le nom de la colonne et la valeur.

Code 6-29 : Insertion de données dans une base de données SQLite

```
...
public long insertPlanete(Planete planete) {
   ContentValues valeurs = new ContentValues();
   valeurs.put(COLONNE_NOM, planete.getNom());
   valeurs.put(COLONNE_RAYON, planete.getRayon());
   return maBaseDonnees.insert(TABLE_PLANETES, null, valeurs);
}

public long insertPlanete(ContentValues valeurs) {
   return maBaseDonnees.insert(TABLE_PLANETES, null, valeurs);
}
...
```

> **ALTERNATIVE**
>
> Vous pouvez aussi utiliser la méthode `execSQL` en spécifiant une requête SQL standard `INSERT` pour arriver au même résultat. L'utilisation de la méthode `insert` permet de faciliter l'action en proposant une approche objet.

Mettre à jour des données

Pour mettre à jour des données dans SQLite, utilisez la méthode `update` de la classe `SQLiteDatabase` en spécifiant un objet `ContentValues` contenant les nouvelles valeurs et la valeur de la clause de condition `WHERE`.

Le contenu de la méthode `updatePlanete` de l'adaptateur de notre exemple se présente ainsi.

Code 6-30 : Mise à jour de données dans une base de données SQLite

```
...
public int updatePlanete(int id, Planete planeteToUpdate) {
   ContentValues valeurs = new ContentValues();
   valeurs.put(COLONNE_NOM, planeteToUpdate.getNom());
   valeurs.put(COLONNE_RAYON, planeteToUpdate.getRayon());
   return maBaseDonnees.update(TABLE_PLANETES, valeurs, COLONNE_ID + " = "
         + id, null);
}
```

```
public int updatePlanete(ContentValues valeurs, String where,
      String[] whereArgs) {
   return maBaseDonnees.update(TABLE_PLANETES, valeurs, where, whereArgs);
}
...
```

Le dernier paramètre de la méthode `update` est la condition – clause identique au paramètre de la clause standard `SQL UPDATE` – permettant de spécifier les éléments à mettre à jour. Seuls les éléments répondant à cette condition seront modifiés.

> **ALTERNATIVE**
>
> Vous pouvez aussi utiliser la méthode `execSQL` en spécifiant une requête SQL standard `UPDATE` pour arriver au même résultat. L'utilisation de la méthode `update` permet de faciliter l'action en proposant une approche objet.

Supprimer des données

Pour supprimer des données d'une table, utilisez la méthode `delete` de la classe `SQLiteDatabase` en spécifiant le nom de la table ciblée et le critère permettant à la base d'identifier les éléments à supprimer.

Voici le contenu des méthodes de suppression de l'adaptateur de notre exemple :

Code 6-31 : Suppression de données dans une base de données SQLite

```
...
public int removePlanete(String nom) {
   return maBaseDonnees.delete(TABLE_PLANETES, COLONNE_NOM + " LIKE "
         + nom, null);
}

public int removePlanete(int id) {
   return maBaseDonnees.delete(TABLE_PLANETES, COLONNE_ID + " = " + id,
         null);
}

public int removePlanete(String where, String[] whereArgs) {
   return maBaseDonnees.delete(TABLE_PLANETES, where, whereArgs);
}
```

Le dernier paramètre, spécifiant le critère de sélection des lignes à supprimer, doit être formaté de la même manière que le paramètre `WHERE` de la clause SQL standard `DELETE`.

ALTERNATIVE

Vous pouvez aussi utiliser la méthode `execSQL` en spécifiant une requête SQL standard `DELETE` pour arriver au même résultat. L'utilisation de la méthode `delete` permet de faciliter l'action en proposant une approche objet.

En résumé

Dans ce chapitre, nous avons abordé différentes façons de stocker, de manipuler et d'accéder à des données depuis vos applications : préférences, fichiers et bases de données. Chaque mécanisme possède ses propres avantages et inconvénients. Pour choisir le plus approprié, vous pourrez vous baser sur plusieurs critères : portée de l'accès aux données, difficulté/délai d'implémentation, structuration ou non des données, rapidité en lecture et enfin, nécessité ou non de devoir effectuer des requêtes sur ces données.

La nécessité de partager et d'exposer vos données aux autres applications vous fera également pencher pour un stockage en particulier ou un autre. Vous vous poserez peut-être la question suivante : pourquoi est-il intéressant de partager vos données avec d'autres applications ? Sachez que c'est grâce au partage de données que fonctionnent les applications essentielles d'Android : contacts, photos, musiques, etc. Alors pourquoi ne pas rendre vos applications également indispensables ?

Le prochain chapitre abordera les différentes possibilités offertes par la plate-forme Android pour partager ses données avec notamment les fournisseurs de contenu et les dossiers dynamiques, ou *live folders*.

7

Partager des données

Une application non communicante ne peut rester longtemps un centre d'intérêt si elle n'expose pas ses données aux autres applications.

Internet a définitivement modifié le paysage de l'information. Aujourd'hui, l'information circule vite et il y a peu de chances pour qu'elle ne soit pas reprise par une autre source dans les heures qui suivent sa publication. En proposant des applications trop fermées, c'est-à-dire en n'exposant pas vos données aux autres applications, qui pourront à leur tour en tirer partie, vous risquez d'avantager vos concurrents. En effet, pour eux, mieux vaut être la seule source d'information et rester incontournable.

Au-delà de cette considération, vous pouvez décider d'étendre votre application, ou celle d'un concurrent, en proposant des extensions utilisant les mécanismes de partage de données mis à disposition par un service déjà installé sur le téléphone de l'utilisateur.

Le partage des données d'Android via les fournisseurs de contenu est un excellent moyen de diffuser de l'information selon une interface standard d'échange. Quant aux dossiers dynamiques (*live folders*), ils vous permettront de créer une interface directement accessible depuis le Bureau de l'utilisateur pour visualiser les informations d'un fournisseur de contenu, sans même démarrer votre application.

Les fournisseurs de contenu

L'utilisation des bases de données vous permet de stocker des données complexes et structurées. L'accès à une base de données n'est possible que depuis l'application à partir de laquelle elle a été créée. Si vous souhaitez exposer les données d'une application à d'autres applications, par exemple des photos prises par votre application, Android prévoit un mécanisme plutôt élégant pour parvenir à cette fin : la notion de fournisseur de contenu, sous la forme d'une interface générique permettant d'exposer les données d'une application en lecture et/ou en écriture. Ce mécanisme permet de faire une scission claire entre votre application et les données exposées.

En rendant vos données disponibles au travers d'une telle interface, vous rendez votre application accessible et extensible à d'autres applications en tant que fournisseur de contenu, que ces applications soient créées par vous-même ou des tiers. C'est d'ailleurs le cas des dossiers dynamiques qui utilisent des données exposées de cette façon pour les afficher sur le Bureau de l'utilisateur.

Accéder à un fournisseur de contenu

Pour accéder à un fournisseur de contenu, vous devrez utiliser la classe `android.content.ContentResolver`. Cette classe est un véritable utilitaire qui sera votre principal point d'accès vers les fournisseurs de contenu Android.

Portez également une attention particulière à l'espace de noms `android.provider` dans lequel vous trouverez un ensemble de classes facilitant l'accès aux fournisseurs de contenu natifs de la plate-forme Android (appels, contacts, etc.). Nous reviendrons sur ce point plus loin dans ce chapitre.

Vous pouvez récupérer une instance – unique pour l'application – de la classe `ContentResolver` en utilisant la méthode `getContentResolver` du `Context`.

```
ContentResolver contentResolver = getContentResolver();
```

Pour accéder à un fournisseur de contenu en particulier, vous devez utiliser l'URI associé, ou plus précisément l'autorité de l'URI pour identifier le fournisseur de contenu.

> **VOCABULAIRE Autorité d'un URI**
>
> L'autorité d'un URI est une portion de celle-ci définissant l'adresse principale de la ressource. Par exemple, dans l'URI `content://com.eyrolles.provider.films/tous`, l'autorité est `com.eyrolles.provider.films`. Le `ContentResolver` utilise cette partie de l'URI pour retrouver le fournisseur de contenu ciblé.

Chaque fournisseur de contenu expose publiquement une propriété `CONTENT_URI`, qui stocke l'URI de requête du contenu, comme le montrent les fournisseurs de contenu natifs d'Android suivants :

```
android.provider.CallLog.Calls.CONTENT_URI
android.provider.Calendar.CONTENT_URI
android.provider.MediaStore.Images.Media.EXTERNAL_CONTENT_URI
```

Un fournisseur de contenu fonctionne un peu à la manière d'un service web accessible en REST (que nous verrons au chapitre 9) : vous exposez un ensemble d'URI capable de vous retourner différents ensembles d'éléments (tous les éléments, un seul ou un sous-ensemble) en fonction de l'URI et des paramètres.

Quand une requête cible un fournisseur de contenu, c'est le système qui gère l'instanciation de celui-ci. Un développeur n'aura jamais à instancier les objets d'un fournisseur de contenu lui-même.

Effectuer une requête

Tout comme pour les bases de données abordées au chapitre précédent, les fournisseurs de contenu retournent leurs résultats sous la forme d'un `Cursor`. L'utilisation d'un curseur a largement été détaillée dans le précédent chapitre. Sachez que vous pouvez effectuer les mêmes opérations que lorsque vous manipuliez des bases de données, l'utilisation dans le cadre d'un fournisseur de contenu ne limite en rien ses fonctionnalités.

Une requête s'effectuera en utilisant la méthode `query` du `ContentResolver` en passant les paramètres listés dans le tableau 7-1.

Tableau 7–1 Paramètres de la méthode query de la classe ContentResolver

Nom du paramètre	Description
`uri`	L'URI du fournisseur de contenu à requêter.
`projection`	La projection, soit les colonnes que vous souhaitez voir retournées. Cette valeur peut aussi être `null`.
`where`	Une clause `where` pour filtrer les éléments retournés. Dans cette chaîne de caractères, l'utilisation du `'?'` est possible ; chacun de ces caractères sera remplacé par les valeurs du paramètre de sélection (le paramètre suivant). Cette valeur peut aussi être égale à `null`.
`selection`	Un tableau de sélections qui remplaceront les caractères `'?'` placés dans la clause `where`. Ceci permet de rendre dynamiques les requêtes de façon à ne pas avoir à manipuler une chaîne de caractères pour construire la requête mais de le faire via l'ensemble de valeurs d'un tableau remplie dynamiquement à l'exécution. Cette valeur peut aussi être `null`.

Tableau 7–1 Paramètres de la méthode query de la classe ContentResolver (suite)

Nom du paramètre	Description
`order`	Le nom de la colonne associée à une chaîne de tri (`'DESC'` ou `'ASC'`). Cette valeur peut aussi être égale à `null`.

Voici une façon de lire les données d'un fournisseur de contenu :

Code 7-1 : Lire les données d'un fournisseur de contenu

```
// Requêter toutes les données du fournisseur de contenu.
Cursor mesInformations = getContentResolver().query(MonFournisseur.CONTENT_URI,
null, null, null, null);

// Filtrer les données retournées et trier par ordre croissant sur le nom.
String[] projection = new String[] {
                _ID,
                COL_PRENOM,
                COL_NOM
            };
String filtre = COL_PRENOM + " = Julien";
String ordre = COL_NOM + " ASC";
Cursor monCurseur = getContentResolver().query(MonFournisseur.CONTENT_URI,
projection, filtre, null, ordre);
```

Une fois le curseur récupéré, vous opérez de la même façon qu'avec une base de données.

La partie concernant la création d'un fournisseur de contenu expliquera plus en détail comment fonctionne le paramètre d'URI d'une requête.

Pour restreindre la requête à un élément en particulier, vous pouvez ajouter la valeur de son `_ID` correspondant. Par exemple, si celui-ci est 10, l'URI sera `content://com.eyrolles.android.fournisseurDemo/10`.

Il existe plusieurs méthodes d'aide, telles que `ContentUris.withAppendedId` et `Uri.withAppendedPath`, qui vous faciliteront la vie. Pour continuer notre exemple, le code correspondant pour construire la requête adéquate permettant de ne cibler qu'un élément en particulier s'écrit ainsi :

Code 7-2 : Récupérer une donnée spécifique d'un fournisseur de contenu

```
Uri uri = Uri.parse("content://com.eyrolles.android.fournisseurDemo");

// Utilisation de la méthode ContentUris pour créer une URI de requête
// permettant de récupérer l'élément _ID=10.
Uri personneID10 = ContentUris.withAppendedId(uri, 10);
```

```
// Utilisation de la méthode ContentUris pour créer une URI de requête
// permettant de récupérer l'élément _ID=10. Prend une chaîne de
// caractères au lieu d'un entier.
Uri personneID10 = Uri.withAppendedPath(uri, "10");

// Requête pour l'élément _ID=10 en particulier.
Cursor cur = managedQuery(personneID10, null, null, null, null);
```

Les fournisseurs de contenu natifs

Android possède quelques fournisseurs de contenu disponibles nativement permettant d'exposer certaines informations contenues dans les bases de données du système.

Tableau 7–2 Les fournisseurs de contenu de la plate-forme Android

Fournisseur de contenu	Description
Contacts	Retrouvez, ajoutez ou modifiez les informations des contacts.
Magasin multimédia	Le magasin multimédia stocke l'ensemble des fichiers multimédias de votre appareil. Ajoutez ou supprimez les fichiers multimédias depuis ce fournisseur de contenu.
Navigateur	Accédez à l'historique, aux marque-pages ou aux recherches.
Appels	Accédez à l'historique des appels entrants, sortants et manqués
Paramètres	Accédez aux paramètres système – sonnerie, etc. – pour lire ou modifier certaines préférences.
Live Folders (1.5)	Accédez aux composants d'exposition de fournisseurs de contenu (détaillé plus loin dans ce chapitre).
Dictionnaire utilisateur (1.5)	L'arrivée du composant d'entrée virtuelle dans Android 1.5 s'accompagne aussi de l'ajout de ce fournisseur de contenu permettant de manipuler le dictionnaire du système.

Afin de garantir une intégration optimale avec le système Android et les autres applications, nous ne pouvons que fortement vous conseiller d'utiliser ces fournisseurs de contenu pour accéder à l'ensemble des informations communes décrites dans le tableau 7-2 ou les modifier.

Le détail de chaque fournisseur de contenu natif sort du périmètre de ce livre. Pour n'en détailler qu'un seul, nous avons choisi le gestionnaire de contacts.

Les concepteurs de la plate-forme Android l'ont bien compris, pouvoir accéder librement aux contacts d'un appareil de téléphonie est une aubaine pour bon nombre d'applications. En accordant les permissions adéquates, un développeur peut effectuer une requête, modifier, supprimer ou encore ajouter un contact.

L'exemple suivant permet de lire l'ensemble des contacts stockés dans l'appareil.

Code 7-3 : Récupérer la liste des contacts depuis le fournisseur de contenu natif

```
public class ListeContacts extends ListActivity {
   Cursor mCurseur;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

        Uri uri = ContactsContract.Contacts.CONTENT_URI;
        String sortOrder = ContactsContract.Contacts.DISPLAY_NAME
                + " COLLATE LOCALIZED ASC";
        // Requête pour toutes les personnes en utilisant la classe
        // Contacts.People.
        mCurseur = getContentResolver().query(uri, null, null, null, sortOrder);
        // Utilisation d'un wrapper pour ne pas avoir à gérer nous-même le
        // curseur en fonction du cycle de vie de l'activité.
        startManagingCursor(mCurseur);

        // Création d'un adaptateur pour associer les données du curseur
        // à la liste de l'activité.
        ListAdapter adaptateur = new SimpleCursorAdapter(this,
                android.R.layout.two_line_list_item, mCurseur,
                new String[] { ContactsContract.Data.DISPLAY_NAME },
                new int[] { android.R.id.text1 });

        // Association de l'adaptateur.
        setListAdapter(adaptateur);
   }
}
```

> **À NE PAS OUBLIER Les permissions d'accès aux fournisseurs de contenu**
>
> La plupart des fournisseurs de contenu ne donnent accès à leurs contenus qu'avec des permissions. Par conséquent, n'oubliez pas d'ajouter la permission adéquate à l'application lors de la lecture de ce fournisseur de contenu. Pour l'exemple précédent, ajoutez la permission `<uses-permission android:name="android.permission.READ_CONTACTS" />` dans le fichier de configuration de l'application, sinon la requête provoquera une erreur.

Ajouter, supprimer ou mettre à jour des données via le fournisseur de contenu

Les fournisseurs ne sont pas seulement des points d'exposition pour permettre aux applications d'effectuer des requêtes sur des bases de données d'autres applications. Ils permettent aussi de manipuler les données exposées en termes d'ajout, de modification et de suppression. Toutes ces actions s'effectuent toujours à l'aide de la classe `ContentResolver`.

Ajouter des données

Pour insérer des données, la classe `ContentResolver` propose deux méthodes : `insert` et `bulkInsert`. La première permet d'insérer une seule donnée et la seconde d'insérer un jeu de données en un seul appel.

La méthode `insert` prend en paramètres un URI du type de donnée que vous souhaitez ajouter et un objet `ContentValues` contenant les données.

Code 7-4 : Insérer des données dans un fournisseur de contenu

```
// Crée un objet contenant les valeurs de la donnée à ajouter.
ContentValues nouvellesValeurs = new ContentValues();
nouvellesValeurs.put(COL_PRENOM, "Damien");
nouvellesValeurs.put(COL_NOM, "Guignard");
// Insertion de la valeur dans le fournisseur de contenu cible.
Uri uriDeMaNouvelleValeur = getContentResolver().insert(MyProvider.CONTENT_URI,
nouvellesValeurs);
```

La méthode `bulkInsert` prend en paramètres une URI et un tableau d'objets `ContentValues`.

Code 7-5 : Insérer des données en masse dans un fournisseur de contenu

```
// Création d'un tableau de valeurs.
ContentValues[] tableauValeurs = new ContentValues[5];
// Alimentation des différentes valeurs dans les différentes instances de
// ContentValues.
tableauValeurs[0] = new ContentValues();
tableauValeurs[0].put(COL_PRENOM, "Damien");
tableauValeurs[0].put(COL_NOM, "Guignard");

tableauValeurs[1] = new ContentValues();
tableauValeurs[1].put(COL_PRENOM, "Julien");
tableauValeurs[1].put(COL_NOM, "Chable");
int nbElementAjoute =
getContentResolver().bulkInsert(MonFournisseur.CONTENT_URI, tableauValeurs);
```

Mettre à jour les données

Vous avez trouvé les éléments qui vous intéressent depuis le fournisseur de contenu et vous souhaitez les mettre à jour ? Pour cela, la classe `ContentResolver` met à votre disposition la méthode `update`. Celle-ci prend en paramètres une valeur `ContentValues` spécifiant les valeurs de chaque colonne de l'entrée ainsi qu'un paramètre de filtre pour spécifier quel ensemble de données est concerné par cette mise à jour.

Tous les éléments répondant à la condition du filtre sont mis à jour et la valeur que retourne la méthode `update` est le nombre d'éléments ayant été mis à jour avec succès.

Le code suivant présente un squelette pour mettre à jour un ensemble d'éléments.

Code 7-6 : Mettre à jour des données d'un fournisseur de contenu

```
// Les valeurs à insérer.
ContentValues nouvellesValeurs = new ContentValues();
nouvellesValeurs.put(COL_PRENOM, "Julien");
nouvellesValeurs.put(COL_NOM, "Chable");

// Le filtre spécifiant les données à mettre à jour.
String filtre = COL_NOM + " = Guignard";

// Mise à jour des éléments du fournisseur de contenu.
int nbLigneMisAJour = getContentResolver().update(MonFournisseur.CONTENT_URI,
nouvellesValeurs, filtre, null);
```

Supprimer des données

Tout comme pour le curseur de base de données, vous avez la possibilité de supprimer des éléments directement depuis le fournisseur de contenu avec la méthode `delete`.

Vous pouvez supprimer une seule valeur, celle ciblée par l'URI, ou plusieurs valeurs en spécifiant un paramètre de filtre.

Code 7-7 : Supprimer des données d'un fournisseur de contenu

```
// Création de l'URI du fournisseur de contenu.
Uri uri = Uri.parse("content://com.eyrolles.android.fournisseurDemo");

// Création de l'URI ciblant l'élément possédant un _ID=10.
Uri personneID10 = ContentUris.withAppendedId(uri, 10);
// Suppression de l'élément possédant un _ID=10.
getContentResolver().delete(personneID10, null, null);

// Suppression de tous les éléments du fournisseur de contenu dont le prénom
// est 'Julien'.
String filtre = COL_PRENOM + " = Julien";
getContentResolver().delete(uri, filtre, null);
```

Créer un fournisseur de contenu

La création d'un fournisseur de contenu s'effectue en dérivant une classe de `ContentProvider` et en redéfinissant les méthodes du tableau 7-3 en fonction de vos besoins.

Tableau 7–3 Méthodes d ún fournisseur de contenu

Nom de la méthode	Description
`onCreate()`	Obligatoire. Appelée lors de la création du fournisseur de contenu. Renvoie un booléen pour spécifier que le fournisseur de contenu a été chargé avec succès.
`query()`	Requête sur le fournisseur de contenu. Retourne un curseur permettant au programme local l'ayant appelé de naviguer au sein des résultats retournés.
`delete()`	Suppression d'une ou de plusieurs lignes de données. L'entier retourné représente le nombre de lignes supprimées.
`insert()`	Appelée lorsqu'une insertion de données doit être réalisée sur le fournisseur de contenu. L'URI renvoyée correspond à la ligne nouvellement insérée.
`update()`	Mise à jour d'une URI. Toutes les lignes correspondant à la sélection seront mises à jour avec les valeurs fournies dans l'objet `ContentValues` des paramètres.
`getType()`	Retourne le type de contenu MIME à partir de l'URI spécifiée en paramètre.

La méthode `query` retourne un objet `Cursor` vous permettant de manipuler et de naviguer dans les données de la même façon qu'avec un curseur de base de données SQLite (souvenez-vous, `Cursor` n'est qu'une interface). Par conséquent, vous pouvez utiliser n'importe quel curseur disponible dans le SDK Android : `SQLiteCursor` (base de données SQLite), `MatrixCursor` (pour des données non stockées dans une base de données), `MergeCursor`, etc.

Nous détaillerons les autres méthodes lors de leur implémentation tout au long d'un exemple exposant les planètes de notre système solaire. Pour commencer, créez une classe, dérivez celle-ci de `ContentProvider` et implémentez la méthode `onCreate`.

Code 7-8 : Implémentation d'un fournisseur de contenu

```
import android.content.ContentProvider;
import android.content.ContentValues;
import android.database.Cursor;
import android.net.Uri;

public class MonFournisseur extends ContentProvider {

   @Override
   public boolean onCreate() {
      return true;
   }
}
```

De façon à identifier sans ambiguïté et pouvoir manipuler un fournisseur de contenu, vous devez spécifier un URI unique, en général basé sur l'espace de noms de la classe d'implémentation du fournisseur. De façon à être accessible par tous les développeurs

et applications, l'URI doit être exposé sous la variable publique statique nommée `CONTENT_URI`.

Dans le cadre de notre exemple, voici la valeur de la variable `CONTENT_URI` sera :

```
content://com.eyrolles.android.provider.planetarium/planetes
```

Avec cet URI, vous demanderez au fournisseur de contenu que nous allons élaborer de renvoyer toutes les planètes de notre système solaire. Si vous complétez la requête en la préfixant avec un numéro, vous demandez la 4e planète de notre système solaire (le soleil correspondant à l'index 0) :

```
content://com.eyrolles.android.provider.planetarium/planetes/4
```

Le SDK d'Android possède une classe `UriMatcher` permettant de faciliter la gestion des URI. En utilisant et en configurant celle-ci, vous n'aurez pas à scinder les URI vous-même pour essayer d'en extraire le contenu, la classe `UriMatcher` le fera pour vous.

Vous trouverez une mise à jour de notre fournisseur de contenu qui prend en compte la gestion des URI ci-dessous :

Code 7-9 : Implémentation d'un fournisseur de contenu

```
public class MonFournisseur extends ContentProvider {

   private static final String authority =
           "com.eyrolles.android.provider.planetarium";
   private static final String uriPlanetes = "content://" + authority
         + "/planetes";
   public static final Uri CONTENT_URI = Uri.parse(uriPlanetes);

   private PlanetesDBAdaptateur dbAdaptateur;

   @Override
   public boolean onCreate() {
      dbAdaptateur = new PlanetesDBAdaptateur(getContext());
      dbAdaptateur.open();
      return true;
   }

   private static final int TOUTES_PLANETES = 0;
   private static final int PLANETE_UNIQUE = 1;

   private static final UriMatcher uriMatcher;
```

```
    static {
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        uriMatcher.addURI(authority, "planetes", TOUTES_PLANETES);
        uriMatcher.addURI(authority, "planetes/#", PLANETE_UNIQUE);
    }
}
```

Lors de l'ajout d'un URI à la classe `UriMatcher`, vous pouvez spécifier qu'une partie de l'URI soit variable en utilisant le caractère #. De cette façon, vous pouvez construire un ensemble d'URI avec la possibilité pour chacun de spécifier un identifiant unique. C'est cet identifiant qui vous sera renvoyé lorsque vous comparerez l'URI spécifié au fournisseur de données avec la méthode `match` de l'`UriMatcher`. Si un URI correspond à un modèle d'URI préalablement enregistré dans cet objet, la méthode vous renverra cet identifiant afin d'identifier la nature de l'URI.

Pour fournir un accès à la source de données, implémentez ensuite toutes les méthodes de la classe abstraite `ContentProvider`. À nouveau, l'utilisation des méthodes de l'adaptateur de la base de données simplifie beaucoup le travail de l'implémentation de ce fournisseur de contenu.

Code 7-10 : Implémentation d'un fournisseur de contenu : insertion, mise à jour et suppression

```
public int delete(Uri uri, String selection, String[] selectionArgs) {
    int retVal = 0;
    switch (uriMatcher.match(uri)) {
    case PLANETE_UNIQUE:
        int id = 0;
        try {
            id = Integer.parseInt(uri.getPathSegments().get(1));
            retVal = dbAdaptateur.removePlanete(id);
        } catch (Exception ex) {
            throw new IllegalArgumentException("URI non supportée : " + uri);
        }
        break;

    case TOUTES_PLANETES:
        retVal = dbAdaptateur.removePlanete(selection, selectionArgs);
        break;

    default:
        throw new IllegalArgumentException("URI non supportée : " + uri);
    }

    getContext().getContentResolver().notifyChange(uri, null);
    return retVal;
}
```

```
public Uri insert(Uri uri, ContentValues valeurs) {
  long id = dbAdaptateur.insertPlanete(valeurs);
   if (id > 0) {
      Uri retUri = ContentUris.withAppendedId(CONTENT_URI, id);
      getContext().getContentResolver().notifyChange(retUri, null);
      return retUri;
   }
    return null;
}

public int update(Uri uri, ContentValues valeurs, String selection,
      String[] selectionArgs) {
   int retVal = 0;
   switch (uriMatcher.match(uri)) {
   case PLANETE_UNIQUE:
     String id = uri.getPathSegments().get(1);
      retVal = dbAdaptateur
            .updatePlanete(valeurs, PlanetesDBAdaptateur.COLONNE_ID
                  + " = " + id, selectionArgs);
      break;

   case TOUTES_PLANETES:
     retVal = dbAdaptateur.updatePlanete(valeurs, selection,
            selectionArgs);
      break;

   default:
      throw new IllegalArgumentException("URI non supportée : " + uri);
   }

   getContext().getContentResolver().notifyChange(uri, null);
   return retVal;
}
```

Lors d'une requête sur un fournisseur de contenu, la méthode renvoie un objet `Cursor`. C'est pourquoi il est indispensable d'exposer le nom et l'ordre des colonnes pour aider les développeurs à manipuler les résultats retournés :

Code 7-11 : Implémentation d'un fournisseur de contenu : requête

```
public static final int COLONNE_ID = 0;
public static final int COLONNE_NOM = 1;
public static final int COLONNE_RAYON = 2;

...

public Cursor query(Uri uri, String[] projection, String selection,
      String[] selectionArgs, String sortOrder) {
```

```
    switch (uriMatcher.match(uri)) {
    case PLANETE_UNIQUE:
       int id = 0;
       // Récupère l'index de la planète désirée.
       try {
          id = Integer.parseInt(uri.getPathSegments().get(1));
       } catch (Exception ex) {
          return null;
       }
       // Récupère le curseur contenant la planète.
      Cursor retCurseurPlanete = dbAdaptateur.getPlaneteCurseur(id);
       // Enregistre le ContentResolver de façon à ce qu'il soit notifié si
       // le curseur change.
       retCurseurPlanete.setNotificationUri(getContext()
             .getContentResolver(), uri);
       return retCurseurPlanete;

       case TOUTES_PLANETES:
       // Récupère le curseur contenant l'ensemble des planètes.
       Cursor retCurseurPlanetes = dbAdaptateur.getAllPlanetesCurseur();
       // Enregistre le ContentResolver de façon à ce qu'il soit notifié si
       // le curseur change.
       retCurseurPlanetes.setNotificationUri(getContext()
          .getContentResolver(), uri);
       return retCurseurPlanetes;

    return null;
}
```

Pour qu'Android puisse identifier la nature des données retournées, la méthode `getType` doit être redéfinie pour renvoyer les types de contenus MIME :

Code 7-12 : Implémentation d'un fournisseur de contenu : type du fournisseur

```
public String getType(Uri uri) {
    switch (uriMatcher.match(uri)) {
    case TOUTES_PLANETES:
       return "vnd.android.cursor.collection/planetarium";
    case PLANETE_UNIQUE:
       return "vnd.android.cursor.item/planetarium";
    default:
       throw new IllegalArgumentException("URI non supportée : " + uri);
    }
}
```

Un fournisseur de contenu est un composant d'une application Android au même titre qu'une activité ou un service : il convient donc de spécifier son existence dans le fichier de configuration de l'application. Voici un squelette de déclaration :

```
<provider android:name=".provider.FournisseurPlanete"
      android:authorities="com.eyrolles.android.provider.planetarium"/>
```

L'utilisation de l'attribut `android:authorities` permet de spécifier à quel URI le fournisseur de contenu répond. Cette valeur correspond au `CONTENT_URI` du fournisseur de contenu. Cela permet à la plate-forme Android de savoir quel fournisseur de contenu exécuter même si l'application n'est pas lancée.

Le fournisseur peut ainsi être appelé par une application comme n'importe quel autre fournisseur.

```
ContentResolver contentResolver = getContentResolver();
Cursor c = contentResolver.query(FournisseurPlanete.CONTENT_URI, null,
      null, null, null);
```

Utiliser un gabarit pour exposer vos fournisseurs de contenu

Si la création d'un fournisseur de contenu est une opération simple pour un développeur Java, il n'en sera pas toujours de même pour ceux qui désireront utiliser votre fournisseur de contenu. Pour faciliter la vie aux autres développeurs de votre équipe, il est donc important d'exposer des méthodes d'aide à la manipulation de votre fournisseur de contenu.

Cette pratique est identique au modèle de conception utilisé dans la partie dédiée aux bases de données et consiste à créer une classe d'abstraction spécifique au fournisseur de contenu exposé. Cette classe possède des méthodes spécifiques aux actions permises par le fournisseur de contenu, cachant la logique intrinsèque, et les colonnes permettant de manipuler le curseur retourné par les méthodes.

Ce modèle est respecté par les fournisseurs de contenu natifs d'Android, nous l'avions vu dans l'exemple dédié au magasin de fichiers multimédias :

```
android.provider.MediaStore.Images.Media.insertImage(getContentResolver(),
bitmap, "ma_maison_de_campagne", "La photo de ma maison à la campagne");
```

Les dossiers dynamiques (Live Folders)

Les dossiers dynamiques ou *Live Folders* ont été introduits avec Android 1.5 afin de permettre aux utilisateurs de visualiser les informations directement sur leur Bureau sans avoir à exécuter les applications. Il ne s'agit de rien d'autre que d'une vue en temps réel des données exposées au travers d'un fournisseur de contenu s'affichant sur le Bureau de l'utilisateur. Si les données changent, vous verrez le dossier dynamique se rafraîchir en temps réel, d'où son nom.

> **VERSION Les Live Folders sont une fonctionnalité d'Android 1.5**
>
> Si vous développez des applications avec la plate-forme Android 1.1, vous n'aurez pas accès à ce type de composant.

Vous pouvez utiliser ce composant pour afficher des contacts, des flux RSS, votre musique, des raccourcis, etc. Les possibilités sont assez nombreuses et ce genre de composant saura donner une autre vie et visibilité à vos applications.

> **DU CÔTÉ DE L'UTILISATEUR Créer un dossier dynamique**
>
> Android fournit quelques exemples de dossiers dynamiques pour afficher vos contacts. Pour créer un dossier dynamique, laissez votre doigt appuyé 2 secondes sur l'écran du Bureau, sélectionnez *Folders* et choisissez ensuite celui que vous souhaitez afficher.

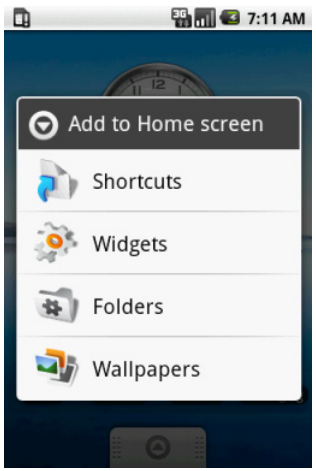


Figure 7–1 Sélectionnez Folders afin d'ouvrir la fenêtre de sélection.

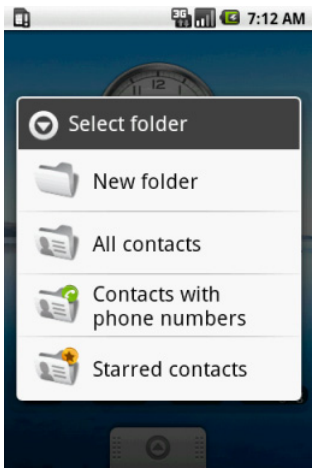


Figure 7–2 Sélectionnez le Live Folder que vous désirez ajouter.

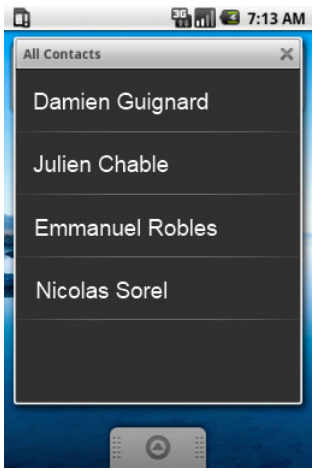


Figure 7–3
Le Live Folder « Tous les contacts » d'Android

Pour un développeur, l'avantage de ce composant est qu'il est simple à implémenter dans les applications Android. Il suffit en effet de (1) implémenter un fournisseur de contenu ou compléter celui que vous avez déjà développé, et (2) créer l'activité du *live folder*.

L'activité en question n'a rien de particulier, elle suit un ensemble de règles de nommage et répond à l'action `android.intent.action.CREATE_LIVE_FOLDER`. Après avoir créé votre activité, la première action que vous aurez à réaliser pour que l'activité soit identifiée en tant que dossier dynamique, est de la déclarer dans le fichier de configuration de l'application accompagnée d'un filtre d'Intents.

Code 7-13 : Ajout d'un Live Folder dans le fichier de configuration de l'application

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.eyrolles"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
      <activity
          android:name=".livefolder.PlanetesLiveFolder"
          android:label="Planetarium">
          <intent-filter>
              <action android:name="android.intent.action.CREATE_LIVE_FOLDER" />
              <category android:name="android.intent.category.DEFAULT" />
          </intent-filter>
      </activity>
    </application>
    <uses-sdk android:minSdkVersion="3" />
</manifest>
```

Une fois l'activité du dossier dynamique ajoutée, vous pouvez commencer à écrire le code de votre activité en prêtant une attention particulière à la méthode `create`.

Attention, un dossier dynamique est une activité qui retourne un résultat, identifiant le fournisseur de contenu. Il s'agit d'un `Intent` spécifiant l'URI de ce fournisseur de contenu et un ensemble de données supplémentaires (nom, icône, etc.) définies par les règles de nommage des Live Folders.

Ainsi, dans la méthode `create` de l'activité, appelez la méthode `setResult` pour renvoyer un résultat de type `RESULT_OK` et un objet `Intent` possédant comme valeur l'URI du fournisseur de contenu et l'ensemble des données supplémentaires suivantes :

- `EXTRA_LIVE_FOLDER_NAME` est le nom du Live Folder, celui qui est s'affiche à l'utilisateur ;
- `EXTRA_LIVE_FOLDER_ICON` spécifie l'icône qui est affichée sur le Bureau de l'utilisateur et sur laquelle il clique pour ouvrir le Live Folder ;
- `EXTRA_LIVE_FOLDER_DISPLAY_MODE` spécifie le mode d'affichage du Live Folder. Par exemple, une valeur de `LiveFolders.DISPLAY_MODE_LIST` permet d'afficher les données sous la forme d'une liste.

Le code suivant est un squelette fonctionnel du Live Folder de notre planétarium permettant de créer une icône et une liste sur le Bureau de l'utilisateur.

Code 7-14 : Implémentation d'un dossier dynamique (*Live Folder*)

```
public class PlanetesLiveFolder extends Activity {

   public static final Uri CONTENT_URI = Uri.parse(
  "content://com.eyrolles.android.provider.planetarium/planetes/live_folders");

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        final Intent intent = getIntent();
        final String action = intent.getAction();

        if (LiveFolders.ACTION_CREATE_LIVE_FOLDER.equals(action)) {
            final Intent intent = new Intent();
            intent.setData(CONTENT_URI);
            intent.putExtra(LiveFolders.EXTRA_LIVE_FOLDER_NAME, "Planètes");
            intent.putExtra(LiveFolders.EXTRA_LIVE_FOLDER_ICON,
                Intent.ShortcutIconResource.fromContext(this, R.drawable.icon));
            intent.putExtra(LiveFolders.EXTRA_LIVE_FOLDER_DISPLAY_MODE,
                            LiveFolders.DISPLAY_MODE_LIST);

            setResult(RESULT_OK, intent);
        } else {
```

```
            setResult(RESULT_CANCELED);
        }

        finish();
    }
}
```

Le code 7-14 peut être exécuté et le Live Folder ajouté au Bureau de l'utilisateur. Cependant, aucun fournisseur de données ne renverra de données puisque l'URI de l'objet `Intent` du Live Folder n'est jusqu'à maintenant géré par aucun fournisseur de contenu. Cela se traduira par l'affichage, pour le moment, d'un Live Folder vide.

Vous devez donc spécifier dans le fournisseur de contenu qu'il devra gérer l'Intent du dossier dynamique, et renvoyer les données qui seront affichées à l'intérieur de celui-ci. Pour ce faire, ajoutez dans le code du fournisseur de contenu une constante pour identifier tout Intent provenant d'un Live Folder, puis ajoutez un `HashMap` permettant de spécifier l'association entre les colonnes renvoyées par le curseur de la requête de sélection et les champs du dossier.

Code 7-15 : Implémentation d'un *Live Folder* – suite : modification du fournisseur de contenu

```
private static final int TOUTES_PLANETES = 0;
private static final int PLANETE_UNIQUE = 1;
private static final int LIVE_FOLDER = 2; // Live folder

...

private static final HashMap<String, String>
LIVE_FOLDER_PROJECTION_MAP;

static {
   LIVE_FOLDER_PROJECTION_MAP = new HashMap<String, String>();
   LIVE_FOLDER_PROJECTION_MAP.put(LiveFolders._ID,
         PlanetesDBAdaptateur.COLONNE_NOM + " AS " + LiveFolders._ID);
   LIVE_FOLDER_PROJECTION_MAP.put(LiveFolders.NAME,
         PlanetesDBAdaptateur.COLONNE_NOM + " AS " + LiveFolders.NAME);
}
```

Complétez l'association des URI de l'`UriMatcher` de façon à ce que l'Intent du Live Folder soit prise en compte par le fournisseur de contenu.

Code 7-16 : Implémentation d'un *Live Folder* (suite) : modification du fournisseur de contenu

```
static {
   ...
   uriMatcher.addURI(authority, "planetes/#", PLANETE_UNIQUE);
   uriMatcher.addURI(authority, "planetes/live_folders", LIVE_FOLDER);
}
```

Puis dans la méthode `getType` du fournisseur de contenu :

Code 7-17 : Implémentation d'un *Live Folder* (suite) : modification du fournisseur de contenu

```
public String getType(Uri uri) {
   switch (uriMatcher.match(uri)) {
      case TOUTES_PLANETES:
      case LIVE_FOLDER:
         return "vnd.android.cursor.collection/planetarium";
      ...
   }
}
```

Ajoutez un cas dans la sélection de l'URI dans la méthode `query` afin d'extraire les données de la table des planètes :

Code 7-18 : Implémentation d'un *Live Folder* (suite) : modification du fournisseur de contenu

```
public Cursor query(Uri uri, String[] projection, String selection,
         String[] selectionArgs, String sortOrder) {
   switch (uriMatcher.match(uri)) {
      case PLANETE_UNIQUE:
      ...

      case TOUTES_PLANETES:
      ...

      case LIVE_FOLDER:
         SQLiteQueryBuilder sqlQb = new SQLiteQueryBuilder();
         sqlQb.setTables("table_planetes");
         sqlQb.setProjectionMap(LIVE_FOLDER_PROJECTION_MAP);
         Cursor c = sqlQb.query(dbAdaptateur.getBaseDonnees(), projection,
               selection, selectionArgs, null, null, null);
         c.setNotificationUri(getContext().getContentResolver(), uri);
         return c;
   }
   return null;
}
```

L'exécution de cette application devrait vous donner la possibilité d'ajouter le dossier dynamique du planétarium et de visualiser toutes les planètes contenues dans la base de données au travers du fournisseur de contenu.

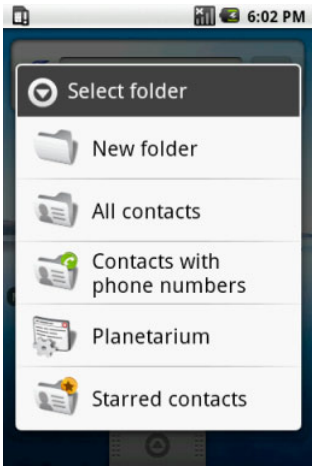


Figure 7–4 Installation du Live Folder du planétarium

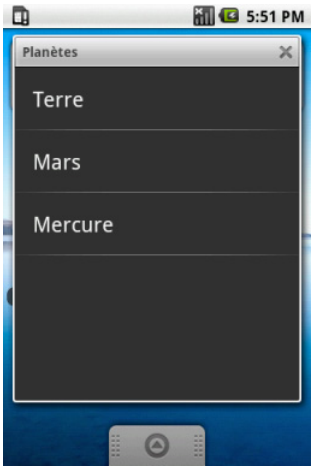


Figure 7–5 Le Live Folder du planétarium ouvert sur le Bureau de l'utilisateur

En résumé

Ce chapitre a été l'occasion d'aborder différentes façons d'exposer les données de vos applications en dehors de celles-ci, pour communiquer tant avec des composants de votre application qu'avec des applications tierces.

Les fournisseurs de contenu permettent aux applications d'effectuer une requête, de modifier ou de supprimer les données exposées par votre application. Un fournisseur de contenu fournit une abstraction de l'accès aux données d'une application ; il permet l'accès et la modification des données de façon unifiée quelle que soit la source de données utilisées par l'application. Bien sûr, vous restez maître de l'accès à ces données en autorisant ou non la lecture, la modification ou la suppression pour chaque requête. Dans la pratique, il est d'ailleurs très courant de n'avoir que des fournisseurs de contenu qui exposent leurs données en lecture.

Les dossiers dynamiques, ou *Live Folders*, arrivés avec Android 1.5, permettent de consommer les données d'un fournisseur de contenu pour les afficher sur le Bureau de l'utilisateur sans nécessiter l'exécution de l'application elle-même. Ce type de composant est très utile si votre application est orientée vers les données, telles que les lecteurs de flux d'informations, de météo, etc.

Les nombreux mécanismes d'accès aux données offrent une grande latitude dans l'utilisation d'un moyen ou un autre pour créer vos applications. Notez que l'accès à des données et à la persistance via le réseau est aussi une excellente option, qui sera traitée au chapitre 6.

8

Multimédia

Les téléphones mobiles ressemblent à des couteaux suisses aux capacités multiples. Les critères de choix ne reposent plus sur les simples fonctions de téléphonie mais sur les fonctions multimédias (formats audio pris en charge, qualité de l'appareil photo, etc.).

Dans ce chapitre, nous allons nous concentrer sur la capture et la lecture de données multimédias pour transformer votre téléphone en boîte à outils multimédia.

La plate-forme Android propose des fonctionnalités d'encodage et de décodage pour différents types de médias. Il est d'ailleurs assez facile d'intégrer des sons, de la vidéo ou des images dans vos applications. Nous allons voir comment le faire en recourant aux mécanismes déjà étudiés.

Plates-formes et formats pris en charge

CULTURE Android et OpenCORE

Android est basée sur OpenCORE, plate-forme multimédia Open Source créée par PacketVideo - membre de l'Open Handset Alliance. Pour plus de détails sur cette plate-forme et notamment sur son architecture, reportez-vous au site dédié.

- http://www.packetvideo.com/products/android/index.html

Le tableau 8-1 donne la liste des formats multimédias que le système Android peut lire. La colonne « Encodeur » indique si la plate-forme peut enregistrer des fichiers au format indiqué.

> **EN SAVOIR PLUS** **Formats supplémentaires**
>
> Chaque appareil peut proposer une partie seulement de ces formats ou des formats supplémentaires non listés ici. C'est par exemple le cas du G1 T-Mobile qui permet de lire le format audio WMA et le format vidéo WMV. On peut également citer le Magic, distribué par SFR, pour lequel les formats disponibles sont AAC, AAC+, AMR-NB, MP3, WMA, WAVE, LC-AAC, MIDI et Ogg Vorbis pour l'audio, et MPEG-4 ainsi que 3GPP pour la vidéo.

Tableau 8–1 Formats multimédias disponibles sur la plate-forme Android

Catégorie	Format	Encodeur	Types de fichiers et extensions
Audio	LC/LTP-AAC	Non	3GPP (`.3gp`) et MPEG-4 (`.mp4`, `.m4a`)
	HE-AAC v1 (AAC+)	Non	
	HE-AAC v2	Non	
	AMR-NB	Oui	3GPP (`.3gp`)
	AMR-WB	Non	3GPP (`.3gp`)
	MP3	Non	MP3 (`.mp3`)
	MIDI	Non	Type 0 et 1 (`.mid`, `.xmf`, `.mxmf`). De même que RTTTL/RTX (`.rtttl`, `.rtx`), OTA (`.ota`), et iMelody (`.imy`)
	Ogg Vorbis	Non	Ogg (`.ogg`)
	PCM/WAVE	Non	WAVE (`.wav`)
Vidéo	H.263	Oui	3GPP (`.3gp`)
	H.264/ MPEG-4 AVC	Non	3GPP (`.3gp`) et MPEG-4 (`.mp4`)
	MPEG-4 SP	Non	3GPP (`.3gp`)
Images	JPEG	Oui	JPEG (`.jpg`)
	GIF	Non	GIF (`.gif`)
	PNG	Non	PNG (`.png`)
	BMP	Non	BMP (`.bmp`)

Pour obtenir la liste complète et à jour de ces formats, consultez le site de référence Android proposé par Google. Vous y apprendrez également de nombreux détails techniques sur l'utilisation de ces formats comme par exemple le fait que le format MP3 est lisible en mono et stéréo, de 8 à 320 Kb/s, en taux d'échantillonnage (*bit rate*) constant ou variable.

> **NOTE Utilisation de Wikipedia**
>
> La description des quelques formats de cette liste s'inspire de l'encyclopédie Wikipédia dont la qualité des articles en fait une référence. N'hésitez pas à vous rapporter à la plus connue des encyclopédies communautaires sur Internet si vous souhaitez plus d'informations !
>
> ▸ http://fr.wikipedia.org

Rappel sur quelques formats audio

Les noms des formats audio sont pour la plupart connus grâce à l'utilisation très répandue des baladeurs numériques. Voici quelques détails supplémentaires qu'il nous a semblé important de rappeler.

Tableau 8–2 Rappel sur quelques formats audio

MP3 (*MPEG Audio Layer 3*)	Algorithme de compression audio le plus connu. Il est capable de réduire la quantité de données nécessaires pour restituer des sons avec une perte de qualité sonore significative mais acceptable pour l'oreille humaine. Pour donner un ordre d'idée, le fichier compressé aura une taille d'environ un douzième du fichier original.
AAC (*Advanced Audio Coding*)	Algorithme de compression audio avec perte de données également. Son algorithme est plus performant que celui du MP3, ce qui permet d'obtenir des fichiers moins volumineux avec une meilleure qualité, tout en nécessitant moins de ressources pour l'encodage ou le décodage. À noter que le format AAC a été choisi comme principal format de fichier par la société Apple dans les iPod ou dans son logiciel iTunes. Android exploite différentes versions du format. Vous pourrez les découvrir en fonction de vos besoins et nous ne les détaillerons pas davantage ici.
AMR-NB (*Adaptive Multi Rate Narrow-Band*)	Format de compression audio de faible qualité souvent utilisé dans les technologies destinées aux mobiles. Il existe une variante dite « bande élargie », l'AMR-WB (*AMR Wide-Band*).
MIDI (*Musical Instrument Digital Interface*)	Protocole de communication et de commande permettant l'échange de données entre instruments de musique électronique.
WMA (*Windows Media Audio*)	Format de compression audio développé par Microsoft. Ses performances sont comparables à celles du MP3 et offre de nombreux avantages tels que la compression avec ou sans perte, la gestion des droits numériques, etc.
Ogg Vorbis	Ogg est le nom du principal projet de la fondation Xiph.org dont le but est de proposer à la communauté des formats et codecs (procédés de compression et de décompression d'un signal numérique) multimédias ouverts et libres de tout brevet.

Rappel sur quelques formats vidéo

Comme pour le format audio, nous rappelons les principaux formats utilisés.

Tableau 8–3 Rappel sur les principaux formats vidéo

H.263	Format développé à l'origine pour la transmission de la vidéo sur des lignes à très bas débit pour des applications de visiophonie via le réseau téléphonique commuté. Il a ensuite été intégré dans les protocoles de visioconférence sur IP.
MPEG-4 SP, MPEG-4 Part 2 ou MPEG-4 ASP (*Advanced Simple Profile*)	Partie 2 de la norme MPEG-4 pour la compression vidéo. Elle est utilisée, entre autres, par les codecs DivX 3 à 6 (propriétaires) et Xvid (sous licence GNU GPL). La partie 10 de la norme MPEG-4 représente une amélioration de ce codage, appelée H.264 ou MPEG-4 AVC (voir ci-dessous).
H.264, aussi appelé MPEG-4 AVC (*Advanced Video Coding*)	Norme de codage vidéo. La norme UIT-T H.264 et la norme ISO/CEI MPEG-4 Part 10 sont techniquement identiques. Ce format comprend de nombreuses techniques nouvelles qui lui permettent de compresser beaucoup plus efficacement les vidéos que les normes précédentes et fournit plus de flexibilité aux applications dans un grand nombre d'environnements réseau. Il est utilisé par les codecs DivX 7 et x264 (sous licence GNU GPL).

Les formats d'images

Il s'agit de formats assez classiques, manipulables dans la plupart des logiciels (libres ou non).

Tableau 8–4 Rappel sur les principaux formats d'image

JPEG (*Joint Photographic Experts Group*)	Le format JPEG, au départ le nom d'un groupe de travail, est un format de compression très efficace mais avec perte de qualité. Plus l'image est compressée, plus sa qualité diminue. Il faut donc trouver un compromis permettant un chargement rapide tout en gardant une qualité acceptable via un facteur de compression. Le format JPEG est particulièrement adapté et recommandé pour les images de type photographie, car il permet d'utiliser un grand nombre de couleurs et donne un bon rendu pour les images nuancées et les dégradés. En revanche, il ne supporte pas la transparence et ne permet pas de créer des animations.
GIF (*Graphics Interchange Format*)	Format qui utilise une compression sans perte de qualité. Les images au format GIF peuvent contenir 256 couleurs – en fait 256 couleurs par calque, le GIF89a supportant les calques multiples, ce qui rend ce format peu adapté pour les photographies, les images nuancées et les dégradés. Il donne en revanche d'excellents résultats pour les images comportant très peu de couleurs, les logos, les formes géométriques, les boutons, etc. Le format GIF supporte la transparence et permet de créer des animations : les images GIF animées, largement utilisées sur Internet. Le format GIF a longtemps été pénalisé puisque soumis à un brevet. C'est en partie cette contrainte qui a poussé la communauté à développer de nouveaux formats comme le PNG. Le GIF est maintenant un format libre mais beaucoup moins utilisé.

Tableau 8–4 Rappel sur les principaux formats d'image (suite)

PNG (*Portable Network Graphics*)	Format d'image numérique ouvert qui a été créé pour remplacer – et surpasser – le format GIF, à l'époque propriétaire et dont la compression était soumise à un brevet. Comme le GIF, le PNG supporte une palette indexée jusqu'à 256 couleurs mais également les niveaux de gris jusqu'à 16 bits et les couleurs réelles jusqu'à 42 bits, ainsi que la transparence. Les fichiers PNG sont généralement plus légers que les fichiers GIF. En revanche, le format PNG est moins performant que le format JPEG pour la compression des photographies dans la mesure où il s'agit d'un format non destructeur. Par ailleurs, il ne permet pas de créer des animations.
BMP (*Bitmap*, littéralement « carte de bits »)	Format d'image matricielle ouvert développé par Microsoft et IBM. Chaque point représente exactement – position et couleur – un pixel à l'écran ou à l'impression. Le format BMP est maintenant peu utilisé car il n'utilise généralement pas de compression et les fichiers de ce type sont donc très volumineux.

Lecture audio

Android permet de lire différents types de sources : des fichiers audio ou vidéo stockés dans les ressources de l'application, depuis des fichiers du système de fichiers, ou à partir d'un flux de données provenant d'une connexion réseau.

> LIMITATION **Capacité de lecture audio**
>
> On ne peut lire les données audio que via le haut-parleur, les écouteurs branchés ou l'éventuel kit Bluetooth configuré. Les sons ne peuvent pas être lus dans le flux de la conversation.

Lecture d'un fichier embarqué par l'application

C'est probablement l'utilisation la plus courante pour alerter l'utilisateur ou pour jouer une musique dans un jeu, par exemple.

Voici les différentes étapes de son utilisation (code 8-1) :

1. Placer le son (ou la vidéo) que l'on veut jouer dans le répertoire `res`. Prenons l'exemple d'un son qui devra être déposé dans le répertoire `res/raw` du projet afin que le complément Eclipse le trouve et en fasse une ressource accessible depuis la classe `R`.
2. Créer une instance de la classe `MediaPlayer` grâce à la méthode statique `MediaPlayer.create`.
3. Appeler la méthode `start` sur l'instance qui suit.

Code 8-1 : Création et lecture d'un fichier multimédia

```
MediaPlayer mediaPlayer = MediaPlayer.create(context, R.raw.fichier_exemple);
mp.start();
```

Pour arrêter la lecture, il faut appeler la méthode `stop`. Il est également possible de rejouer le son (ou la vidéo) en appelant successivement les méthodes `reset`, puis `prepare` sur l'instance de la classe `MediaPlayer` avant d'appeler à nouveau la méthode `start`. À noter que la méthode `create` appelle `prepare` la première fois, c'est pour cela que nous ne l'avons pas utilisée dans l'exemple.

Pour mettre le son (ou la vidéo) en pause, il suffit d'invoquer la méthode `pause`, la lecture reprenant après l'appel de la méthode `start`.

Le schéma complet des états et des transitions entre états possibles de la classe `MediaPlayer` est disponible dans la documentation de la classe fournie avec le SDK ou sur Internet. La figure 8-1 propose une représentation simplifiée.

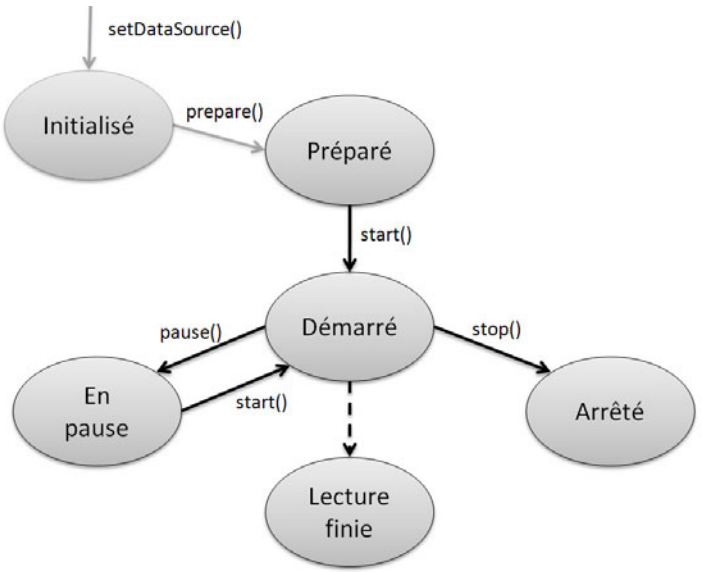


Figure 8–1
Diagramme d'état d'un MediaPlayer simplifié

Lecture à partir d'un fichier ou d'une URL

La méthode est identique qu'il s'agisse d'un fichier ou d'une URL :

1. Créer une instance de la classe `MediaPlayer`.
2. Appeler la méthode `setDataSource` sur cette instance avec en paramètre une chaîne de caractères correspondant au chemin du fichier qui est à jouer (vers le système de fichiers ou une URL).

3 Utiliser la méthode `prepare`, puis `start` sur l'instance ainsi configurée :

Code 8-2 : Lecture d'un fichier multimédia depuis une URL

```
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setDataSource(PATH_TO_FILE);
mediaPlayer.prepare();
mediaPlayer.start();
```

Les différents états s'enchaînent de la même façon que décrite précédemment. Veillez à bien traiter les éventuelles exceptions résultant de l'utilisation de la méthode `setDataSource` en cas de paramètre invalide ou autre problème d'entrée/sortie (E/S).

Réalisation d'un lecteur MP3

Nous pouvons à présent nous lancer dans notre première application multimédia qui permettra de lire un fichier au format MP3 intégré dans les ressources de l'application. Pour cela, un seul bouton est nécessaire sur l'interface graphique.

Code 8-3 : Fichier main.xml pour le lecteur MP3

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
>
<Button android:id="@+id/btn_lecture"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Lecture du fichier"
/>
</LinearLayout>
```

Code 8-4 : Activité LecteurMP3.java

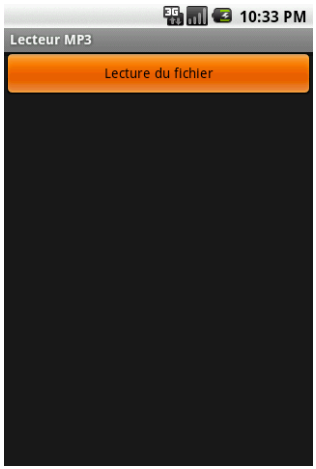
```
package com.eyrolles.android.multimedia;

import android.app.Activity;
import android.media.MediaPlayer;
import android.media.MediaPlayer.OnCompletionListener;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
```

```
public class LecteurMP3 extends Activity {

    private MediaPlayer mediaPlayer;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

       Button btnLecture = (Button)this.findViewById(R.id.btn_lecture);
       btnLecture.setOnClickListener(

        new OnClickListener(){

            public void onClick(View viewParam) {
                mediaPlayer =
                    MediaPlayer.create(LecteurMP3.this, R.raw.son_a_lire); 1

                    // Planifier une action à la fin de la lecture du fichier.
                mediaPlayer.setOnCompletionListener(new OnCompletionListener(){ 2
                  publicvoid onCompletion(MediaPlayer mediaPlayerParam) {
                          // Reste à coder l'action à réaliser.
                    }
                  });
                mediaPlayer.start(); 3
            }
        });
    }

    @Override
    protected void onDestroy() {
       super.onDestroy();
      // Libération des ressources si nécessaire
       if (mediaPlayer != null) {
           mediaPlayer.release(); 4
           mediaPlayer = null;
       }
    }

}
```

On crée donc notre instance de la classe `MediaPlayer` en passant en paramètre la ressource audio à lire 1. On peut éventuellement programmer une action à réaliser à la fin de la lecture du fichier 2. Enfin, on déclenche la lecture 3.

Une bonne pratique consiste à toujours penser à libérer les ressources 4 même si, dans notre exemple, tout aurait fonctionné sans cet ajout.

Figure 8–2
Aperçu du lecteur audio



Enregistrement audio

Généralités sur l'enregistrement audio

L'enregistrement de médias est pris en charge par la classe `MediaRecorder`, que ce soit pour un enregistrement audio ou vidéo. La séquence pour enregistrer des sons est la suivante :

1. Créer une nouvelle instance de la classe `android.media.MediaRecorder`.
2. Déterminer la source audio (ici le micro).

```
mediaRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
```

3. Fixer le format de sortie pour l'enregistrement. Les formats de sortie sont accessibles en tant que membres statiques de la classe `android.media.MediaRecorder.OutputFormat`.

```
mediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.DEFAULT);
```

4. Régler le type d'encodage audio. Les différentes valeurs sont sélectionnables via la classe `android.media.MediaRecorder.AudioEncoder` et ses constantes de classe.

```
mediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.DEFAULT);
```

5 Assigner le fichier de sortie. Attention, cette opération doit être effectuée après l'appel de la méthode `setOutputFormat` et avant l'appel de la méthode `prepare`.

```
mediaRecorder.setOutputFile("monFichier.mp3");
```

Pour commencer l'enregistrement, il faut utiliser les méthodes `prepare` et `start`. Utilisez les méthodes `stop` et `release` pour arrêter l'enregistrement.

De façon analogue au `MediaPlayer`, le diagramme d'état complet du `MediaRecorder` est disponible dans la documentation de la classe. La figure 8-3, propose une version grandement simplifiée pour visualiser plus facilement les différentes étapes.

Le dernier point important consiste à ajouter les permissions requises pour l'enregistrement audio dans le fichier `AndroidManifest.xml` :

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

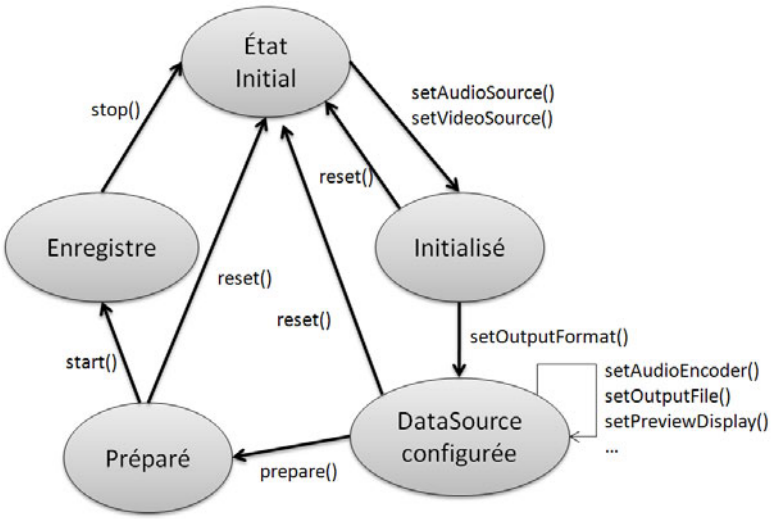


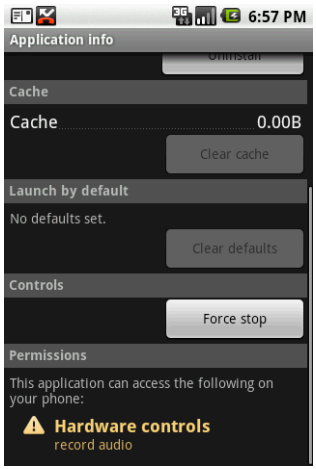
Figure 8–3
Diagramme d'état d'un MediaRecorder simplifié

En cas de doute, pour vérifier les permissions sollicitées par une application, vous pouvez naviguer dans le menu *Settings>Applications>Manage applications* et sélectionner votre application.

À partir de cet exemple, nous allons construire le code qui permettra à votre activité d'enregistrer le son capté par le micro de l'appareil.

Notez que c'est aussi là l'occasion de découvrir l'émulation d'une carte mémoire sur laquelle nous allons stocker le nouveau fichier enregistré.

Figure 8–4
Affichage des informations sur l'application dans l'émulateur



Simulation d'une carte mémoire

SDK inférieur à la version 1.5

Pour simuler une carte SD, nous allons utiliser le logiciel proposé par le SDK : `mksdcard`. Pour commencer, ouvrez une interface DOS et saisissez la ligne de commande suivante :

```
C:\android-sdk\tools\mksdcard -l SD256M 256M C:\android-sdk\card
```

Cette commande crée un fichier nommé `card` qui fonctionnera comme une carte mémoire de 256 Mo grâce à l'émulateur. Il est à noter que les dernières versions du kit de développement requièrent au moins 9 Mo pour faire fonctionner le simulateur correctement.

Il convient ensuite d'ajouter un paramètre de lancement pour l'émulateur afin que la carte mémoire soit prise en compte. Deux solutions sont alors possibles : ajouter le paramètre à la ligne de commande, ou configurer Eclipse de façon appropriée. Pour cela, sélectionnez le menu *Run>Run Configurations...* Le champ *Additional Emulator Command Line Options* de l'onglet *Target* correspondant à votre projet vous permet de renseigner l'option suivante.

```
-sdcard C:\android-sdk\card
```

À partir du SDK 1.5

Le kit de développement 1.5 a introduit la notion d'AVD, que nous avons déjà abordée dans le premier chapitre. Lors de la création du profil matériel, il est possible de demander la création ou l'utilisation d'une image existante.

> **ATTENTION Choisir la plate-forme cible**
>
> Veillez à choisir correctement la plate-forme cible grâce à l'option `-t`. Notez que pour lister les plates-formes et ainsi récupérer leurs identifiants, vous pouvez utiliser la commande `android.bat list targets`.

Pour créer une image de 256 Mo pour la plate-forme 1.5, la commande est la suivante.

```
android create avd -n mon_profil_avd -t 2 -c 256M.
```

Les derniers plugins ADT (et surtout la toute dernière interface graphique du SDK 2.0 intégrée par le plugin) proposent de créer cette carte mémoire dans des formulaires. Rappelez-vous de la création de votre premier AVD au chapitre 1.

Figure 8–5
Création d'une carte mémoire liée à un profil matériel grâce à l'interface graphique du SDK 2.0

Vérifier visuellement et par la programmation la présence de la carte SD

Pour vérifier la présence de la carte, cliquez sur l'icône *Settings>SD card & phone storage* sur l'émulateur. Toutes les informations relatives à la mémoire du téléphone et surtout à celle que vous avez créée sur la carte SD s'afficheront alors.

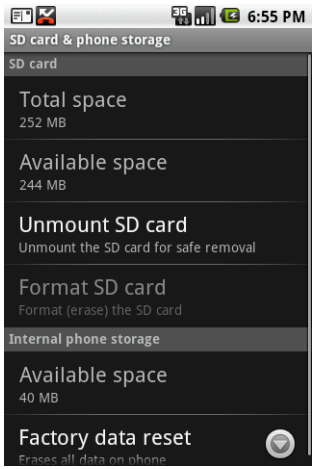


Figure 8–6
Informations sur le stockage de l'émulateur

Côté programmation, les quelques lignes de code suivantes vous permettront de vérifier la présence d'une carte externe.

```
String state = android.os.Environment.getExternalStorageState();
if (!state.equals(android.os.Environment.MEDIA_MOUNTED)) {
    // Réaliser une action ou ajouter une trace...
}
```

On pourra même vérifier si on a le droit d'écrire sur ce média externe :

```
File storageDir = Environment.getExternalStorageDirectory();
if (!storageDir.canWrite()) {
  // Réaliser une action ou ajouter une trace…
}
```

ÉVOLUTIONS DU SDK Nouvelles permissions avec Donut (SDK 1.6)

Donut introduit une nouvelle permission pour accéder à un média externe comme notre carte SD. Vous devrez donc ajouter la ligne suivante dans la section `manifest` du fichier de configuration de l'application :

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

Construction de l'enregistreur

Voici les différentes parties de notre enregistreur. Attention à ne pas oublier la permission permettant d'enregistrer du son dans la section `manifest` du fichier de configuration de l'application.

Code 8-5 : Fichier AndroidManifest.xml pour l'enregistreur

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

Nous opterons ici pour un extrait de code qui permet de lancer l'enregistrement.

Code 8-6 : Code pour déclencher l'enregistrement

```
MediaRecorder mediaRecorder = new MediaRecorder();

// Vérification de la présence de la carte de stockage.
String state = android.os.Environment.getExternalStorageState();
if (!state.equals(android.os.Environment.MEDIA_MOUNTED)) { (1)
    Log.e(LOG_TAG_ENREGISTREUR,
```

```
            "La carte mémoire n'est pas présente");
        return;
    }

    // Vérifie que l'on peut écrire sur la carte.
    File repertoireStockage = Environment.getExternalStorageDirectory();
    if (!repertoireStockage.canWrite()) { 2
        Log.e(LOG_TAG_ENREGISTREUR,
            "Impossible d'écrire sur le périphérique de stockage.");
        return; // Si on est dans un méthode, sinon utiliser un if/else
    }

    // Création du fichier de destination.
    try {
       fichierEnregistre = File.createTempFile("EnregistrementAudio",
                                               ".mp4", repertoireStockage); 3
    } catch (IOException e) {
        Log.e(LOG_TAG_ENREGISTREUR, "Problème E/S avant l'enregistrement");
        return; // Si on est dans un méthode, sinon utiliser un if/else
    }

    mediaRecorder = new MediaRecorder();
    mediaRecorder.setAudioSource(MediaRecorder.AudioSource.MIC); 4

    mediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
    mediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);

    mediaRecorder.setOutputFile(fichierEnregistre.getAbsolutePath());

    mediaRecorder.prepare();
    mediaRecorder.start();
    ;
```

Dans cet exemple, nous avons intégré la vérification de la présence d'une carte pour le stockage externe 1 mais également la possibilité d'écrire dessus 2. La création du fichier cible s'effectue en 3. Ensuite, l'enregistreur reprend la séquence de commandes 4 que nous avons étudiée. Attention enfin à ne pas oublier de traiter l'arrêt de l'enregistrement grâce aux méthodes `stop` et `release` de l'instance de `MediaRecorder`.

Si vous souhaitez lire le fichier avec les applications intégrées à Android, il faut l'indexer pour que l'application de lecture de musique le trouve dans la base multimédia. Pour cela, un outil est proposé avec le SDK, accessible depuis l'émulateur en cliquant l'icône *Dev Tools>Media Scanner*. Il suffit ensuite de lancer le lecteur en cliquant sur l'icône *Music*, votre fichier est alors disponible !

> **REMARQUE** **Indexation de l'enregistrement ?**
>
> Vous remarquerez que nous n'avons pas indiqué de titre ni d'auteur et que, plus embêtant, nous devons lancer une recherche sur le téléphone pour que notre enregistrement (audio mais aussi vidéo et il en serait de même pour une image) ne soit disponible pour les applications du téléphone. Ces deux problématiques seront traitées dans la dernière section de ce chapitre « Demandez encore plus à Android ».

Prendre des photos

Les appareils photo - qui peuvent également prendre des vidéos selon les modèles - sont maintenant présents sur la plupart des téléphones. Bien qu'ils soient nettement moins performants que les appareils classiques pour la plupart des utilisations, ils offrent néanmoins de très nombreuses fonctionnalités et représentent une source quasi inépuisable d'applications.

Voici quelques applications - existantes ou non - possibles avec un appareil photo :

- prise d'une photo et recherche du lieu, du sujet ou de l'objet photographié (application Goggles de Google) ;
- relevé de code-barre pour obtenir le meilleur prix d'un produit (application ShopSavvy) ;
- prise d'une photo de l'utilisateur pour un site communautaire ;
- prise d'un cliché, géolocalisation de l'image (et même avec une orientation si la boussole est disponible) et envoi sur un site d'albums photos.

La classe Camera

Que ce soit pour prendre un cliché ou pour régler des paramètres, nous utiliserons principalement la classe `Camera` dans nos manipulations.

La ligne de code suivante permet de gérer l'obtention de l'instance de la classe `Camera`.

```
Camera camera = Camera.open();
```

Pour libérer cette ressource matérielle, la commande suivante doit être exécutée :

```
camera.release();
```

Entre ces deux commandes, prenez votre photo !

Et n'oubliez pas, pour utiliser l'appareil photo sous Android, il faut ajouter la permission au fichier de configuration de l'application (section `manifest`).

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

Les réglages sont quant à eux accessibles via l'objet `Camera.Parameters` que l'on récupère grâce à la méthode `getParameters` de l'objet `Camera`.

Tous les paramètres sont consultables et réglables grâce à leurs accesseurs (méthodes `get` et `set` associées). Pour consulter le format photo, par exemple, on utilisera la méthode `getPictureFormat` et pour paramétrer ce format en JPEG, on invoquera la méthode `setPictureFormat(PixelFormat.JPEG)`.

Utiliser la prévisualisation

La prévisualisation du champ filmé par la caméra peut être affichée sur une `Surface`. À noter que sur un émulateur, le rendu affichera un damier sur lequel un carré bouge de façon aléatoire.

Cette surface d'affichage est décrite dans un fichier de mise en page XML de façon assez habituelle en spécifiant sa taille.

Code 8-7 : Fichier main.xml pour la prévisualisation de la caméra

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <SurfaceView
        android:id="@+id/surface_view"
        android:layout_width="fill_parent"
        android:layout_height="320px"
    />
</LinearLayout>
```

La surface que nous allons détenir en la manipulant sera contrôlable grâce à une instance d'un objet de type `SurfaceHolder`. L'interface `SurfaceHolder` implémentée par cet objet permet :

- de contrôler la taille de la surface ;
- de changer son format ;
- d'éditer les pixels de la surface ;
- de surveiller les changements inhérents à la surface.

Un client peut surveiller les événements liés à la surface en implémentant l'interface du *callback* `SurfaceHolder.Callback`. Une fois enregistré, ce client sera notifié des événements de création, suppression et modification de la surface.

> **PRÉCISION Les callbacks**
>
> Un *callback*, terme qui pourrait se traduire en français par « rappel », est en pratique matérialisé par un objet implémentant les méthodes d'une interface. Cet objet est passé en paramètre à une classe qui le garde précieusement. Cette dernière invoquera les méthodes de l'interface que l'objet implémente pour signaler un fait donné lors d'un événement précis ou à la fin d'un traitement, par exemple.
> Vous pouvez à ce sujet vous reporter au principe des écouteurs d'événements dans la bibliothèque graphique Swing de la plate-forme Java standard ou étudier de plus près le design pattern observateur.

Mettons de suite ces concepts en pratique et réalisons une prévisualisation que nous compléterons ensuite par la prise d'un cliché.

Code 8-8 : Fichier de l'activité VideoPreview pour la prévisualisation de la caméra

```
package com.eyrolles.android.multimedia;

import java.io.IOException;

import android.app.Activity;
import android.hardware.Camera;
import android.os.Bundle;
import android.util.Log;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class VideoPreview extends Activity
    implements SurfaceHolder.Callback { (1)
    private static final String LOG_TAG_VIDEO_PREVIEW = "VideoPreview";

    private SurfaceView surfaceView;
    private SurfaceHolder surfaceHolder;
    private Camera camera;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        surfaceView = (SurfaceView)findViewById(R.id.surface_view); (2)
        surfaceHolder = surfaceView.getHolder(); (3)
        surfaceHolder.addCallback(this); (4)
        surfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }
```

```
    /**
     * Méthode appelée après un changement de taille ou de format de la surface.
     * @param holder L'instance implémentant SurfaceHolder
     * dont la surface a changée.
     * @param format Le nouveau PixelFormat de la surface.
     * @param width La nouvelle largeur de la surface.
     * @param height La nouvelle hauteur de la surface.
     */
    public void surfaceChanged(SurfaceHolder holder, int format, int width,
            int height) { 5
        if (camera != null) {
            camera.stopPreview();
            Camera.Parameters p = this.camera.getParameters();
            p.setPreviewSize(width, height);
            this.camera.setParameters(p);
            try {
                this.camera.setPreviewDisplay(holder);
                this.camera.startPreview();
            } catch (IOException e) {
                Log.e(LOG_TAG_VIDEO_PREVIEW,
                      "Erreur E/S lors du setPreviewDisplay sur l'objet Camera",
                        e);
            }
        }
    }

    /**
     * Méthode appelée immédiatement après la création de la surface.
     */
    public void surfaceCreated(SurfaceHolder holder) {
        camera = Camera.open(); 6
    }

    /**
     * Méthode appelée immédiatement avant la destruction de la surface.
     */
    public void surfaceDestroyed(SurfaceHolder holder) {
        if (camera != null) {
            camera.stopPreview(); 7
            camera.release();
        }
    }
}
```

On implémente l'interface `SurfaceHolder.Callback` 1 et on s'enregistre pour être notifié des événements relatifs à la surface 4. C'est d'ailleurs l'objet de type `SurfaceHolder` obtenu auprès de la surface 3 déclarée dans le fichier de mise en page 2 qui nous notifiera des différentes étapes de création, suppression et modification de la surface.

À la création de la surface (6), on récupère l'objet permettant de manipuler la caméra.

Quand la surface est modifiée (5), on diffuse sur la surface les images captées par l'objectif. La documentation précise bien que cette méthode est appelée au moins une fois après la création de la surface, donc pas de souci.

Enfin, quand la surface est détruite (7), on libère les ressources liées à la caméra.

Capture d'une image

Pour prendre une image, on utilise la méthode `takePicture` sur une instance de l'objet `Camera`. Cette méthode prend en paramètres différents callbacks.

Code 8-9 : Prendre une photo avec l'appareil intégré

```
private void takePicture() {
    camera.takePicture(shutterCallback, rawCallback, jpegCallback);
}
```

Prenons l'exemple de la méthode `onShutter` de l'objet `ShutterCallback`. Elle sera appelée au moment où l'obturateur se refermera et que la photo sera prise. Le code contenu dans cette méthode sera alors exécuté. On pourrait, par exemple, jouer un son pour signaler la prise de la photo.

Voici les différents callbacks qui peuvent être utilisés :

- `Camera.AutoFocusCallback` : donne des informations sur le succès ou l'échec d'une opération d'autofocus ;
- `Camera.ErrorCallBack` : rappel sur erreur du service de prise de vue ;
- `Camera.PictureCallback` : prise d'une photographie ;
- `Camera.PreviewCallback` : notification des images disponibles en prévisualisation ;
- `Camera.ShutterCallback` : rappel quand l'obturateur est fermé, mais les données ne sont pas encore enregistrées.

L'objet `ShutterCallback` (1) est un paramètre indispensable à chaque appel de la méthode permettant de prendre une photo. Vous pouvez ensuite choisir, grâce au `PictureCallback`, une capture au format brut (2), au format JPEG (3) ou les deux.

> UTILISER L'ÉMULATEUR **Émulation de la prise de clichés**
>
> Sur l'émulateur, comme pour le cas particulier de la prévisualisation vidéo et de son enregistrement, une image type sera enregistrée correspondant à un téléphone portable qui « parle le langage Android »… Vous comprendrez la signification cette phrase dès que vous verrez la photographie !

Reprenons à présent le code précédent qui permet d'afficher une prévisualisation de la caméra et ajoutons un bouton qui, actionné, déclenchera la prise d'une image au format JPEG. Cette image sera enregistrée sur l'espace de stockage externe (notre carte émulée).

Code 8-10 : Fichier main.xml pour la mise en page de la prise de photo

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <SurfaceView
        android:id="@+id/surface_view"
        android:layout_width="fill_parent"
        android:layout_height="320px"
    />
    <Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:id="@+id/btn_prendrePhoto"
    android:text="Clic clac !"
    />
</LinearLayout>
```

Code 8-11 : Fichier pour l'activité CapturePhoto

```
package com.eyrolles.android.multimedia;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.OutputStream;

import android.app.Activity;
import android.content.ContentValues;
import android.hardware.Camera;
import android.hardware.Camera.PictureCallback;
import android.hardware.Camera.ShutterCallback;
import android.net.Uri;
import android.os.Bundle;
import android.provider.MediaStore.Images;
import android.provider.MediaStore.Images.Media;
import android.util.Log;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.widget.Button;
```

```
public class CapturePhoto extends Activity implements SurfaceHolder.Callback,
                                        PictureCallback, ShutterCallback { (1)

    private SurfaceView surfaceView;
    private SurfaceHolder surfaceHolder;
    private Camera camera;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        surfaceView = (SurfaceView)findViewById(R.id.surface_view);
        surfaceHolder = surfaceView.getHolder();
        surfaceHolder.addCallback(this);
        surfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);


        Button prendrePhoto = (Button) findViewById(R.id.btn_prendrePhoto);
        prendrePhoto.setOnClickListener(new View.OnClickListener() {
            public void onClick(View viewParam) {
                camera.takePicture(CapturePhoto.this, null, CapturePhoto.this); (2)
            }
        });
    }

    public void surfaceChanged(SurfaceHolder holder, int format, int width,
                               int height) {
        if (camera != null) {
            camera.stopPreview();
            Camera.Parameters p = this.camera.getParameters();
            p.setPreviewSize(width, height);
            this.camera.setParameters(p);
            try {
                this.camera.setPreviewDisplay(holder);
                this.camera.startPreview();
            } catch (IOException e) {
                Log.e(getClass().getSimpleName(),
                      "Erreur E/S lors du setPreviewDisplay sur l'objet Camera", e);
            }
        }
    }

    public void surfaceCreated(SurfaceHolder holder) {
        camera = Camera.open();
    }

    public void surfaceDestroyed(SurfaceHolder holder) {
        if (camera != null) {
            camera.stopPreview();
            camera.release();
        }
    }
```

```
    public void onPictureTaken(byte[] data, Camera arg1) { 3
        ContentValues values = new ContentValues();
        values.put(Media.TITLE, "Mon image");
        values.put(Media.DESCRIPTION, "Image prise par le téléphone");
        Uri uri = getContentResolver().insert(Images.Media.EXTERNAL_CONTENT_URI,
                                              values); 4
        OutputStream os;
        try {
            os = getContentResolver().openOutputStream(uri);
            os.write(data);
            os.flush();
            os.close();
        } catch (FileNotFoundException e) {
            Log.e(getClass().getSimpleName(), "Fichier non trouvé à l'écriture de
                    l'image", e);
        } catch (IOException e) {
            Log.e(getClass().getSimpleName(), "Erreur E/S à l'enregistrement de
                    l'image", e);
        }
        camera.startPreview();
    }

    public void onShutter() { 5
        Log.d(getClass().getSimpleName(), "Clic clac !");
    }
}
```

Par rapport au programme précédent, on ajoute un bouton qui, une fois activé, déclenchera la prise de vue 2. Pour pouvoir être notifié de cette prise de vue, on donne à la méthode qui capture la photographie une instance d'objet de type `ShutterCallback` et une instance d'objet de type `PictureCallback`, c'est-à-dire l'instance courante puisque la classe `CapturePhoto` implémente ces deux interfaces 1.

La méthode `onShutter` est appelée tout de suite après la prise de vue 5, mais nous ne l'exploitons pas dans cet exemple. La méthode `onPictureTaken` 3 nous permet quant à elle d'enregistrer l'image dont les données sont au format JPEG.

Observez la syntaxe `Images.Media.EXTERNAL_CONTENT_URI` 4. Ce morceau de code permet de récupérer l'emplacement où sont stockées les images enregistrées sur un média externe et d'y placer ses propres images. Le stockage vidéo et audio fonctionne aussi de cette façon.

Si vous voulez pouvoir consulter l'image, vous devrez l'indexer manuellement (comme pour le fichier audio) afin que la galerie soit notifiée de sa présence. L'indexation automatique et l'ajout de métadonnées pour le classement de l'image sont abordés dans la section suivante.

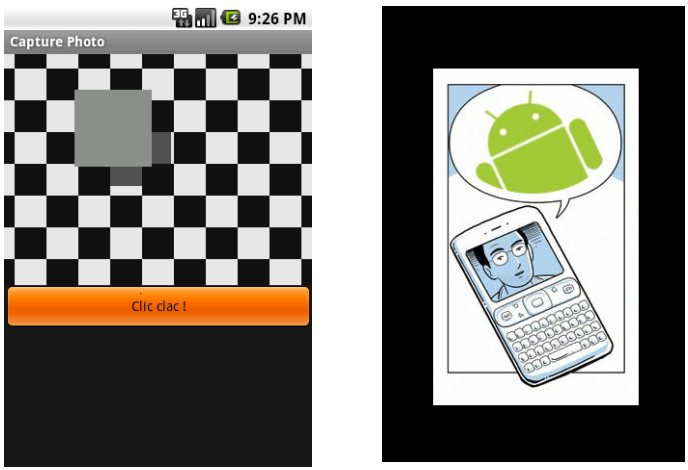


Figure 8–7
Rendu de l'application de photographie sur l'émulateur

Utilisation des fonctionnalités vidéo

Les fonctionnalités de capture vidéo sont pour le moment un peu moins utilisées sur les appareils Android, la part belle est réservée à la consommation des médias vidéo. Voici quelques idées (qui peuvent parfois trouver une implémentation sur le marché Android) :

- prendre une vidéo et la publier sur un site communautaire ou sur un site de partage tel que YouTube (avec l'application éponyme) ;
- récupérer des vidéos sur un site de partage (Dailymotion, YouTube, etc.) pour pouvoir les regarder en mode déconnecté, par exemple lors d'un voyage, ou pour conserver les meilleures. En poussant plus loin, on arriverait à un système de vidéos à la demande (VOD) ;
- vidéo de présentation téléchargeable pour des sites de recherche d'emploi.

Réalisation d'un lecteur vidéo

Pour pouvoir lire un petit film, il faut savoir de quel fichier il s'agit et comment y accéder. Nous avons déjà utilisé les ressources d'un projet grâce à l'exemple du lecteur audio. Nous allons plutôt lire une vidéo qui se trouve sur le réseau.

Notez que si vous voulez lire un fichier vidéo à partir de la carte mémoire, vous pouvez le déposer sur la carte simulée grâce à la commande suivante :

```
adb push c:\films\exemple.mp4 /sdcard
```

Code 8-12 : Fichier main.xml pour le layout de l'application de lecture vidéo

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<VideoView android:id="@+id/videoView" (1)
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    />
</LinearLayout>
```

Pour réaliser l'application, nous avons besoin d'un composant de type `VideoView` (1) pour y afficher la vidéo. Concernant le taille de ce composant, avec nos réglages tout sera automatique, mais vous pouvez également donner sa taille en pixels. Le tout, avec des valeurs fixes, est de ne pas se tromper en étant confronté à un appareil dont l'écran ne supporte pas les tailles déterminées.

Code 8-13 : Fichier LecteurVideo.java pour l'activité du lecteur vidéo

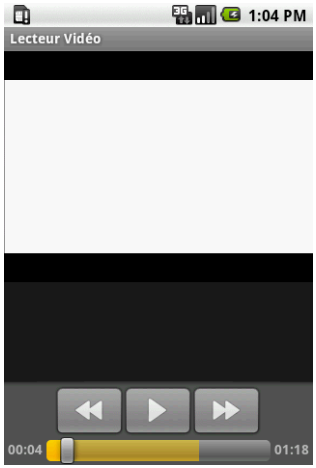
```
package com.eyrolles.android.multimedia;

import android.app.Activity;
import android.net.Uri;
import android.os.Bundle;
import android.widget.MediaController;
import android.widget.VideoView;

public class LecteurVideo extends Activity {

    private VideoView videoView;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.main); (1)

        videoView = (VideoView) findViewById(R.id.videoView); (2)
        videoView.setMediaController(new MediaController(this));
        videoView.setVideoURI(
            Uri.parse("http://video.exemple.com/video.mp4")); (3)
        videoView.start();
    }
}
```

On affecte la mise en page conçue plus haut (1) pour pouvoir récupérer le composant `VideoView` (2) et lui communiquer l'adresse de la vidéo à lire (3).

L'utilisation directe du composant de type `VideoView` n'est pas la seule solution. Comme nous vous l'avons expliqué, vous pouvez utiliser une `MediaPlayer` que vous associerez à un `MediaController`. Ce `MediaController` peut être affecté au composant `VideoView` et le tour est joué.

Pour suivre finement les étapes de la lecture, vous pouvez également utiliser des écouteurs dont `OnBufferingUpdateListener` qui vous permettra d'être averti de l'avancement de la mise en cache de votre vidéo.

Figure 8–8
Rendu du lecteur vidéo



Enregistrement vidéo

> **LIMITATION** **Version 1.5 du SDK pour l'enregistrement vidéo**
>
> Les API pour l'enregistrement vidéo (au format 3GP) ne sont disponibles que depuis la version 1.5 du SDK Android. Par ailleurs, il n'y a pas de réglage possible pour visualiser le rendu d'une source externe sur l'émulateur.

L'enregistrement vidéo est assez semblable à la capture audio bien qu'un soupçon plus compliqué. Voici les différentes étapes permettant d'enregistrer une vidéo :

1. Créer une nouvelle instance de la classe `android.media.MediaRecorder`.
2. Prévoir un affichage de la prévisualisation vidéo sur une surface adaptée. Pour cela, il faut utiliser la méthode `setPreviewDisplay` de la classe `MediaRecorder`.
3. Fixer la source vidéo grâce à la méthode `setVideoSource` de la classe `MediaRecorder`, généralement grâce au paramètre `MediaRecorder.VideoSource.CAMERA`.

4. Fixer la source audio grâce à la méthode `setAudioSource` de la classe `MediaRecorder`. Là encore, le paramètre `MediaRecorder.AudioSource.MIC` semble tout désigné.
5. Régler le format (méthode `setOutputFormat` de la classe `MediaRecorder`), la taille (méthode `setVideoSize` de la classe `MediaRecorder`) et la fréquence des images (méthode `setVideoFrameRate` de la classe `MediaRecorder`).
6. Fixer l'encodage audio grâce à la méthode `setAudioEncoder` de la classe `MediaRecorder`, puis régler l'encodage vidéo via la méthode `setVideoEncoder` de cette même classe.
7. Invoquer les méthodes `prepare` et `start` pour lancer l'enregistrement, puis `stop` et `release` pour l'arrêter.

En guise d'exemple, réalisons un programme minimaliste qui permettra d'enregistrer une vidéo. La classe `Preview` affichera directement ce qui est dans le champ de vision de l'appareil. Elle sera utilisée dans l'activité `CaptureVideo` comme affichage principal.

Code 8-14 : Activité pour la capture vidéo

```
package com.eyrolles.android.multimedia;

import android.app.Activity;
import android.content.pm.ActivityInfo;
import android.media.MediaRecorder;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;

public class CaptureVideo extends Activity {

    private static final String FICHIER_SORTIE = "/sdcard/video_test.3gp";
    private final static String LOG_TAG = "CaptureVideo";

    private MediaRecorder mediaRecorder;
    private Preview preview;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        mediaRecorder = new MediaRecorder(); (1)
        mediaRecorder.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
        mediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
        mediaRecorder.setVideoEncoder(MediaRecorder.VideoEncoder.MPEG_4_SP);
        mediaRecorder.setOutputFile(FICHIER_SORTIE);
```

```
        preview = new Preview(this, mediaRecorder); (2)
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);

        setContentView(preview);
    }

    /**
     * Création du menu avec deux boutons
     * pour lancer et couper l'enregistrement.
     */
    public boolean onCreateOptionsMenu(Menu menu) { (3)
        menu.add(0, 0, 0, "Enregistrer");
        menu.add(0, 1, 0, "Arrêter");
        return super.onCreateOptionsMenu(menu);
    }

    /**
     * Action sur une sélection d'un bouton du menu.
     */
    public boolean onOptionsItemSelected(MenuItem item) { (4)
        if (item.getItemId() == 0) {
            try {
                mediaRecorder.start();
            } catch (Exception e) {
                Log.e(LOG_TAG, e.getMessage, e);
                mediaRecorder.release();
            }
        } else {
            mediaRecorder.stop();
            mediaRecorder.release();
            mediaRecorder = null;
        }
        return super.onOptionsItemSelected(item);
    }
}
```

Cette activité permet de configurer une instance de la classe `MediaRecorder` (1) comme nous l'avons appris. Cet objet est donné en paramètre de construction de la classe `Preview` qui est détaillée ensuite (2).

L'activité dispose d'un menu sommaire (3) permettant de démarrer et d'arrêter l'enregistrement (4).

Code 8-15 : Classe Preview

```
package com.eyrolles.android.multimedia;

import android.content.Context;
import android.media.MediaRecorder;
```

```
import android.view.Surface;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class Preview extends SurfaceView implements SurfaceHolder.Callback {

    private SurfaceHolder surfaceHolder;
    private MediaRecorder mediaRecorder;

    public Preview(Context context, MediaRecorder recorder) {
        super(context);
        mediaRecorder = recorder;
        surfaceHolder = getHolder(); (1)
        surfaceHolder.addCallback(this);
        surfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }

    public Surface getSurface() {
        return surfaceHolder.getSurface();
    }

    public void surfaceCreated(SurfaceHolder holder) { (2)
        mediaRecorder.setPreviewDisplay(surfaceHolder.getSurface()); (3)
        try {
            mediaRecorder.prepare();
        } catch (Exception e) {
            mediaRecorder.release();
            mediaRecorder = null;
        }
    }

    public void surfaceDestroyed(SurfaceHolder holder) { (4)
        if (mediaRecorder != null) {
            mediaRecorder.release();
            mediaRecorder = null;
        }
    }

    public void surfaceChanged(SurfaceHolder holder, int format,
        int w, int h) {
        ;
    }
}
```

La première étape consiste à installer le callback `SurfaceHolder.Callback` pour être prévenu quand la surface va être créée ou détruite (1).

En implémentant les méthodes `surfaceCreated` (2) et `surfaceDestroyed` (4), on est notifié de la création et de la destruction de la surface quand elle est soit visible, soit cachée.

On fixe la surface sur laquelle on veut afficher la prévisualisation (3) et on fait passer le diagramme d'état de l'instance de la classe `MediaRecorder` dans l'état `prepared`. Dans le code précédent, ces étapes ont été effectuées et il ne restait qu'à invoquer la méthode `start` pour démarrer l'enregistrement.

Figure 8–9
Résultat de l'enregistreur vidéo

De manière générale, la capture vidéo ne fonctionne pas bien avec tous les émulateurs dont nous disposons au moment de l'écriture de ce livre. Un téléphone de développement pour le test est alors un atout considérable pour manipuler les API multimédias d'Android.

Dernier conseil, si vous ne trouvez pas la vidéo en allant dans la galerie de l'émulateur ou de votre téléphone de développement, pensez à indexer de nouveau le contenu multimédia de votre téléphone comme nous l'avons fait pour l'enregistreur de sons.

En cas de doute, vous pouvez également utiliser l'incontournable complément Eclipse dont la vue *File Explorer* (*Window>Show View>Other...>Android*) permet de naviguer dans l'arborescence de l'émulateur.

Demandez encore plus à Android : indexation de contenu et reconnaissance des visages

Indexation des médias et mise à disposition des applications

Et si nous voulions que le média enregistré soit automatiquement disponible pour le lecteur de musique concernant les sons, la galerie d'images concernant les photographies, etc. ?

Ajout des médias à la bibliothèque

Pour cela, nous allons employer la classe `MediaScannerConnection` qui permet aux applications de passer les médias fraîchement créés au service `media scanner`. Ce service va lire les métadonnées du fichier et va ajouter ce dernier au fournisseur de contenu multimédia en tenant compte des métadonnées lues.

Le `MediaScannerConnectionClient` propose une interface pour le service d'indexation qui retourne l'URI du fichier scanné au client de la classe `MediaScannerConnection`.

Reprenons l'enregistreur audio précédent et automatisons la détection des fichiers sonores enregistrés. Le code nécessaire étant assez réduit, nous allons en profiter pour bien architecturer notre développement et créer du code facilement réutilisable.

Créons tout d'abord l'interface `ScannerListener` qui sera utilisée plus tard dans notre développement et qui permet à n'importe quelle application de proposer un écouteur à notre code afin d'être prévenue de la fin du traitement du fichier soumis.

```
package com.eyrolles.android.multimedia;

public interface ScannerListener {
    public void newFileScanned(String pathFile); (1)
}
```

Nous avons donc une seule méthode (1) qui prend en paramètre le chemin du fichier traité. La classe qui implémentera cette interface et qui sera notifiée en tant qu'écouteur recevra donc ce chemin et pourra l'exploiter. Passons à présent à la création de la classe `ScannerClient` qui sera en charge de la gestion des connexions au service d'indexation.

Code 8-16 : Fichier ScannerClient.java

```
package com.eyrolles.android.multimedia;

import android.content.Context;
import android.media.MediaScannerConnection;
import android.media.MediaScannerConnection.MediaScannerConnectionClient;
import android.net.Uri;

public class ScannerClient implements MediaScannerConnectionClient{
        private String mediaToScan;
        private String mimeTypeToScan;

        private MediaScannerConnection mediaScanner;
        private ScannerListener scannerListener;
```

```
        public ScannerClient(Context context){
            mediaScanner = new MediaScannerConnection(context,this);
            mediaToScan = "";
        }

        public void setMediaToScan(String media, String mimeType){
            mediaToScan = media;
            mimeTypeToScan = mimeType;
            mediaScanner.connect();
        }

        public void setOnScannerListener(ScannerListener scannerListener){
            this.scannerListener = scannerListener;
        }

        public void onMediaScannerConnected() {

            // Si on ne donne pas de type MIME, l'extension du fichier sera
            // utilisée pour déterminer son type.
            mediaScanner.scanFile(mediaToScan, mimeTypeToScan);
        }

        public void onScanCompleted(String path, Uri uri) {

            if (mediaScanner.isConnected())
                mediaScanner.disconnect();

            if (scannerListener != null)
                scannerListener.newFileScanned(path);
        }
}
```

Dans l'activité principale, à la fin de l'enregistrement du son, ajoutez les lignes suivantes :

```
ScannerClient scannerClient = new
ScannerClient(getApplicationContext());
scannerClient.setMediaToScan(fichierEnregistre.getAbsolutePath(),
                                "audio/amr");
```

Ce code n'exploite pas les possibilités d'écouteur. La classe qui utilise ce fragment de code ne sera pas notifiée de la fin du processus. Pour ce faire, implémentons l'interface `ScannerListener` dans la classe courante. Ceci implique de donner corps à la méthode `newFileScanned` présente dans cette interface pour en respecter le contrat :

```
public void newFileScanned(String pathFile) {
    // Ajouter des métadonnées au fichier, afficher un message, etc.
}
```

Demandons ensuite au `ScannerClient` de nous notifier quand le fichier sera indexé :

```
ScannerClient scannerClient = new
ScannerClient(getApplicationContext());
scannerClient.setOnScannerListener(this);
scannerClient.setMediaToScan(fichierEnregistre.getAbsolutePath(),
                                "audio/amr");
```

Renseigner les informations des médias indexés

Si nous prenons l'exemple des fichiers audio récemment créés, par défaut, aucune information sur les fichiers n'est renseignée. Si vous utilisez le lecteur audio, vous vous apercevrez que tous les fichiers sont classés dans la catégorie *Artiste inconnu*. Rien n'empêche le bon fonctionnement de votre application ou du système, mais à force d'enregistrer du contenu sans le classer, vous ne vous y retrouvez plus et vous n'exploiterez pas les possibilités proposées par la plate-forme.

Pour stocker toutes les informations concernant les fichiers audio, il faut utiliser la classe `ContentValues`. Cette dernière va contenir tous les attributs que nous allons ajouter et qui sont destinés à qualifier nos médias au moment de leur indexation.

De manière générale, c'est le `MediaStore` qui contient les métadonnées de tous les médias enregistrés aussi bien en mémoire interne qu'en mémoire externe. À partir de cette base :

- la classe `MediaStore.Audio` est le conteneur pour tout le contenu audio ;
- la classe `MediaStore.Images` contient les métadonnées pour toutes les images disponibles, par exemple la latitude, la longitude et l'orientation qui permettent de géolocaliser un cliché ;
- la classe `MediaStore.Video` est le conteneur pour tout le contenu vidéo ;
- la classe `MediaStore.MediaColumns` contient quant à elle les champs communs à la plupart des classes citées tels que le titre, le type MIME, la taille, etc.

Voici les différentes étapes permettant de préparer les données que nous voulons associer au fichier :

1 Création du réceptacle pour nos informations :

```
ContentValues contentValues = new ContentValues();
```

2 Ajout des propriétés à ce réceptacle. Pour un fichier audio, on aurait :

```
values.put(MediaStore.MediaColumns.TITLE, "Le titre à afficher");
values.put(Audio.AudioColumns.ALBUM, "Nom de l'album");
values.put(MediaStore.MediaColumns.MIME_TYPE, "audio/amr");
insertedValues.put(MediaStore.MediaColumns.DATA, PATH_TO_FILE);
```

3 Association au média grâce au `ContentResolver` :

```
ContentResolver resolver = context.getContentResolver();
resolver.update(uri, insertedValues, null, null);
```

Modifions la classe `ScannerClient` utilisée précédemment afin d'ajouter des propriétés à un fichier qui sera inséré dans la base multimédia du téléphone.

Code 8-17 : Classe ScannerClient.java

```
package com.eyrolles.android.multimedia;

import android.content.ContentResolver;
import android.content.ContentValues;
import android.content.Context;
import android.media.MediaScannerConnection;
import
android.media.MediaScannerConnection.MediaScannerConnectionClient;
import android.net.Uri;
import android.provider.MediaStore;

public class ScannerClient implements MediaScannerConnectionClient{
    private String mediaToScan;
    private String mimeTypeToScan;

    private MediaScannerConnection mediaScanner;
    private ScannerListener scannerListener;
    private ContentValues mediaProperties;

    private Context context;

    public ScannerClient(Context context){
        this.context = context;
        mediaScanner = new MediaScannerConnection(context,this);
        mediaToScan = "";
        mimeTypeToScan = null;
        mediaProperties = null;
    }

    public void setMediaToScan(String media, String mimeType){
        mediaToScan = media;
        mimeTypeToScan = mimeType;
        mediaScanner.connect();
    }

    public void setMediaProperties(ContentValues properties) {
        mediaProperties = properties;
    }
```

```
    public void setOnScannerListener(ScannerListener scannerListener){
        this.scannerListener = scannerListener;
    }

    public void onMediaScannerConnected() {
        mediaScanner.scanFile(mediaToScan, mimeTypeToScan);
    }

    public void onScanCompleted(String path, Uri uri) {

        if (mediaScanner.isConnected())
            mediaScanner.disconnect();

        if (scannerListener != null)
            scannerListener.newFileScanned(path);

        if (mediaProperties != null) {
            ContentValues insertedValues = new ContentValues(mediaProperties);
            insertedValues.put(MediaStore.MediaColumns.DATA, path);
            ContentResolver resolver = context.getContentResolver();
            resolver.update(uri, insertedValues, null, null);
        }
    }
}
```

Dans votre programme principal, ajoutez les lignes de code suivantes :

```
ContentValues values = new ContentValues();
values.put(MediaStore.MediaColumns.TITLE, "UnTitre");
values.put(Audio.AudioColumns.ALBUM, "Bruits de fond");
values.put(Audio.Media.MIME_TYPE, "audio/amr");
(...)
ScannerClient scannerClient = new
ScannerClient(getApplicationContext());
scannerClient.setMediaProperties(values);
scannerClient.setMediaToScan(fichierEnregistre.getAbsolutePath(),
                                    "audio/amr");
```

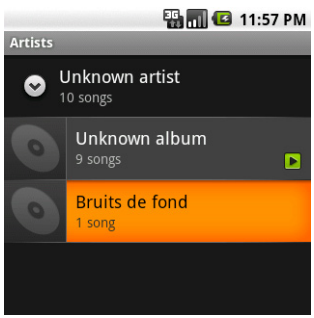


Figure 8–10
Résultat de l'ajout de métadonnées dans le lecteur audio d'Android

Détection des visages sur des images

Oublions quelques minutes les problématiques audio et vidéo et penchons-nous sur une classe à part : `FaceDetector`.

Comme son nom l'indique, elle va nous donner les moyens de détecter le visage d'une ou plusieurs personnes sur une image en se basant sur les yeux. Les différentes étapes de son utilisation sont les suivantes :

1. Création d'une instance de la classe `FaceDetector` qui prendra pour son constructeur les dimensions de l'image ainsi que le nombre de visages à détecter.
2. Transformation de l'image à traiter en bitmap grâce à la méthode statique `BitmapFactory.decodeResource`.
3. Utilisation de la méthode `findFaces` de la classe `FaceDetector`. En passant l'image que nous voulons analyser ainsi qu'un tableau de visages correctement dimensionné, nous récupérerons - espérons-le - notre tableau rempli.

L'objectif de ce programme sera de détecter un certain nombre de visages dans une photographie et d'encercler les yeux détectés.

Commençons par créer une activité qui affichera une vue personnalisée sur laquelle nous dessinerons.

Code 8-18 : Activité DetectionVisages

```
package com.eyrolles.android.multimedia;

import android.app.Activity;
import android.os.Bundle;

public class DetectionVisages extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        VueVisages vueVisages = new VueVisages(this);
        setContentView(vueVisages);
    }
}
```

Créons ensuite la vue personnalisée qui hérite de `View` et qui redéfinira sa méthode `onDraw`.

Code 8-19 : class VueVisages.java

```
package com.eyrolles.android.multimedia;

import android.content.Context;
import android.graphics.Bitmap;
```

```
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.PointF;
import android.graphics.Rect;
import android.media.FaceDetector;
import android.view.View;

public class VueVisages extends View {

    private static final int NB_VISAGES_DETECTES = 2;

    private int longueurImage;
    private int hauteurImage;

    private Bitmap imageSource;

    private PointF pointsMilieuDesYeux[] = new PointF[NB_VISAGES_DETECTES];

    public VueVisages(Context context) {
        super(context);

        imageSource = BitmapFactory.decodeResource(getResources(),
            R.drawable.image); (1)

        longueurImage = imageSource.getWidth();
        hauteurImage = imageSource.getHeight();

        // Récupération des visages
        FaceDetector.Face visagesTrouves[] =
            new FaceDetector.Face[NB_VISAGES_DETECTES];
        FaceDetector faceDetector =
            new FaceDetector(longueurImage, hauteurImage, NB_VISAGES_DETECTES); (2)

        faceDetector.findFaces(imageSource, visagesTrouves); (3)

        // Récupération du milieu des yeux
        FaceDetector.Face visageCourant = null;
        for (int i = 0; i < visagesTrouves.length; i++) { (4)
            visageCourant = visagesTrouves[i];
            if (visageCourant == null)
                break;

            PointF milieuYeux = new PointF();
            visageCourant.getMidPoint(milieuYeux);
            pointsMilieuDesYeux[i] = milieuYeux;
        }
    }
```

```
    protected void onDraw(Canvas canvas) {
        Paint paintCerclesYeux = new Paint(Paint.ANTI_ALIAS_FLAG);
        paintCerclesYeux.setStyle(Paint.Style.FILL);
        paintCerclesYeux.setColor(Color.RED);

        // Affichage de l'image
        canvas.drawBitmap(imageSource, null,
            new Rect(0, 0, longueurImage, hauteurImage), null);

        // Marque les visages trouvés
        for (int i = 0; i < pointsMilieuDesYeux.length; i++) { ❺
            if (pointsMilieuDesYeux[i] != null) {
                // Dessin d'un cercle autour des yeux
                canvas.drawCircle(pointsMilieuDesYeux[i].x,
                pointsMilieuDesYeux[i].y, 60,
                paintCerclesYeux);
        }
      }
    }
}
```

Une fois nos variables initialisées, le programme transforme l'image chargée à partir du projet (l'image est donc à placer dans le répertoire `res/drawable`) en bitmap ❶. Un tableau de visages est récupéré ❷❸ et si aucun visage n'est détecté, chacun de ses éléments sera `null`. Ce qui nous importe pour tracer le dessin sur l'image est le point entre les yeux qui est récupéré dans la boucle ❹ et sert de base au dessin de la boucle ❺ qui transforme légèrement les données en tenant compte du redimensionnement de l'image à la taille de l'écran.

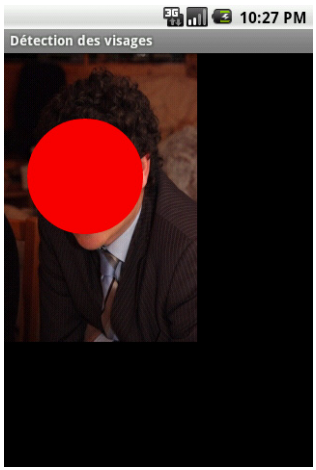


Figure 8–11
Rendu de l'application de détection de visages

En résumé

Les appareils mobiles sont devenus des outils multimédias puissants. La plate-forme Android n'a pas sous-estimé cet aspect en rendant accessibles toutes les fonctionnalités de ces appareils grâce à des fonctions d'assez haut niveau.

Dans ce chapitre, nous avons appris à lire et enregistrer des sons, des images ainsi que des vidéos et à intégrer ces enregistrements au système.

Citons quelques programmes qui tirent parti de toutes ces possibilités et qui sont devenus des incontournables du marché Android :

- Google Sky : dessine et nomme les étoiles que l'objectif de votre appareil photo cible ;
- SFR|Bouygues|Orange TV : tous les opérateurs ont sorti leur application pour regarder la télévision sur votre téléphone Android ;
- Google Listen : permet d'écouter vos podcasts préférés ;
- Shazam : analyse la musique autour de vous et permet de l'identifier (trouver son titre, son interprète, etc.).

9

Statut réseau, connexions et services web

Les applications seront (inter)connectées ou ne seront plus. L'envoi et la récupération de données sont des fonctions indispensables : qu'elle présente de l'information, dialogue sur les réseaux sociaux, mette à jour des bases de données distantes ou affiche de la publicité, une application doit communiquer.

La plupart des applications sont communicantes. Difficile de s'y tromper, car de toutes les permissions qui sont réclamées, la demande de connexion à Internet est l'une des plus fréquentes.

Voici quelques exemples de domaines où, sans réseau, il n'y a pas d'application :

- client pour poster des commentaires sur les réseaux sociaux (Facebook ou Twitter, pour ne citer que les plus connus) ;
- multimédia : télévision (opérateurs mobiles), musique (Deezer ou Spotify, par exemple) ;
- et bien évidemment toutes les applications qui concernent la consultation d'informations : information consommation (applications opérateurs), application dédiée à un journal (Le Monde), trafic ferroviaire (SNCF), trafic routier ou tout simplement navigateur Internet.

Pour réaliser ce type d'applications, voici notre programme pour ce chapitre :

- la consultation du statut réseau du téléphone ;
- la création de requêtes pour accéder à des informations distante ;
- la réalisation de clients pour des services web ;
- l'utilisation des sockets.

Disponibilité du réseau

Afin de concevoir intelligemment les applications qui nécessitent une connectivité au réseau, il est indispensable de vérifier au préalable la disponibilité de cette fameuse connexion.

Pour cela, Android met à notre disposition la classe `ConnectivityManager` qui va nous permettre de déterminer la disponibilité de la connexion réseau et d'obtenir des notifications en cas de changement.

Code 9-1 : Extrait de code qui teste la disponibilité de la connexion réseau

```
protected void onStart() {
    super.onStart();

    ConnectivityManager connectivityManager =
        (ConnectivityManager) getSystemService(CONNECTIVITY_SERVICE); (1)
    NetworkInfo networkInfo =
connectivityManager.getActiveNetworkInfo(); (2)

    // Le type de connexion, réseau ou Wi-Fi.
    int networkType = networkInfo.getType();

    // Vérification de l'état de la connexion.
    State networkState = networkInfo.getState(); (3)
    if (networkState.compareTo(State.CONNECTED) == 0) {
        // Nous sommes connectés.
    }
}
```

En passant la constante `CONNECTIVITY_SERVICE` à la méthode `getSystemService`, nous obtenons un manager (1). À l'aide de ce dernier, nous récupérons une instance de la classe `NetworkInfo` (2). La méthode `getState` appliquée à l'objet ainsi récupéré (3) nous permet de tester l'état de la connexion (états possibles : `CONNECTING`, `CONNECTED`, `DISCONNECTING`, `DISCONNECTED`, `SUPSPENDED` et `UNKNOWN`).

Pour que cet extrait de code fonctionne, n'oubliez pas d'ajouter la permission `android.permission.ACCESS_NETWORK_STATE` dans la section `manifest` du fichier de configuraion de l'application comme suit.

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

Interroger un serveur web grâce au protocole HTTP

Nous allons tout d'abord étudier le protocole HTTP, puis nous le mettrons en œuvre en effectuant quelques requêtes – HTTP – à partir d'un client Android vers un serveur web.

Rappels sur le protocole HTTP

> **CULTURE HTTP, un protocole omniprésent**
>
> Vous utilisez le protocole HTTP (*HyperText Transfer Protocol*) tous les jours dès que vous manipulez un navigateur Internet . Le format des URL vous a d'ailleurs forcément mis la puce à l'oreille : elles commencent par `http://`. Le protocole HTTP est utilisé dans sa version 1.0 depuis le milieu des années 1990. Il a ensuite évolué jusqu'à la version 1.1 qui est aujourd'hui la version la plus utilisée.

À une requête HTTP effectuée par un client, est associée une réponse d'un serveur.

La requête client est constituée :

- d'une ligne de requête (qui peut être de plusieurs types : `GET`, `POST`, `HEAD`, etc.) ;
- d'en-têtes (lignes facultatives qui permettent de communiquer des informations supplémentaires comme la version du client, très utile pour pouvoir adapter les contenus aux mobiles) ;
- du corps de la requête composé de lignes optionnelles utilisées par certains types de requêtes.

Nous nous contenterons d'utiliser des requêtes de type `GET` et `POST`.

Requêtes HTTP de type GET

Les requêtes de type `GET` servent à demander une ressource située à une URL donnée.

Nous allons tout de suite étudier la composition d'une telle requête.

> **VOCABULAIRE URL**
>
> Une URL (*Uniform Resource Locator),* littéralement « localisateur uniforme de ressource », est une chaîne de caractères utilisée pour adresser les ressources du World Wide Web : document HTML, image, son, forum Usenet, boîte aux lettres électronique, etc. Elle est appelée de façon informelle « adresse web » (source : Wikipédia).

Code 9-2 : Exemple de requête HTTP de type GET

```
// Pour l'URL http://www.lemonde.fr, la requête suivante est envoyée à un
// serveur par le navigateur.
GET / HTTP/1.1 (1)
Host: www.lemonde.fr (2)
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; fr; (3) rv:1.9.1.3)
Gecko/20090824 Firefox/3.5.3 (.NET CLR 3.5.30729)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: fr,fr-fr;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie: xtvrn=$43260$
```

La requête HTTP sert donc à obtenir une ressource (1)(2). La ligne suivante (3) identifie le type de client, ici un navigateur Firefox. Cette information, bien qu'optionnelle, est très importante. Elle permet, par exemple, de détecter un navigateur mobile et d'adapter le contenu de la page à la volée (ou de choisir une page différente en cache).

Ce type de requête permet également indirectement de transmettre des informations en passant des paramètres dans l'URL de la ressource sollicitée. On obtient alors des URL formées avec des paires clé/valeur, par exemple :

```
http://www.example.fr/index.html?param1=valeur1&param2=valeur2
```

Les contraintes de cette méthode concernent essentiellement la taille limitée des paramètres (en fait de l'URL dans son ensemble) et l'encodage impératif des caractères transmis (espaces, esperluettes, etc.). Mais c'est une méthode rapide et facile à employer d'autant qu'il est très aisé de récupérer les valeurs côté serveur dans n'importe quel langage de script (JSP, PHP, etc.). Dernier avantage, une telle URL peut être enregistrée dans les favoris des navigateurs.

À toute requête, une réponse est associée. Cette dernière est composée :

- d'une ligne de statut composée de la version du protocole utilisée, d'un code et de la signification de ce dernier ;

- d'en-têtes constitués de lignes optionnelles contenant des paires en-tête/valeur ;
- du corps de la réponse contenant le document demandé.

Code 9-3 : Réponse du site lemonde.fr à la requête précédente

```
HTTP/1.x 200 OK (1)
Server: Apache
Content-Type: text/html
X-Server: britpop
Vary: Accept-Encoding
Content-Encoding: gzip
Cache-Control: max-age=34
Date: Wed, 11 Nov 2009 11:34:15 GMT
Transfer-Encoding: chunked
Connection: keep-alive, Transfer-Encoding
```

Dans cet échange, tout s'est bien passé. Le code `200 OK` (1) nous le confirme.

De façon générale :

- les codes en `20x` concernent la réussite d'une opération ;
- les codes en `30x` indiquent que la ressource n'est plus à l'emplacement désigné (redirection) ;
- les codes en `40x` indiquent une erreur du client. L'erreur `404` est la plus connue et signifie que le serveur n'a pas trouvé la ressource à l'adresse indiquée ;
- les codes en `50x` sont la conséquence d'une erreur interne au serveur.

Requêtes HTTP de type POST

Les requêtes de type `POST` permettent d'envoyer des données au composant logiciel situé à l'URL utilisée. Dans l'exemple du code 9-4, la requête mise en évidence concerne un formulaire simple constitué d'une zone de texte et d'un bouton d'envoi.

Code 9-4 : Requête de type POST résultante de la soumission du formulaire

```
POST /formulaires/post/formpost.asp HTTP/1.1
Host: www.exemple.fr
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; fr; rv:1.9.1.3)
Gecko/20090824 Firefox/3.5.3 (.NET CLR 3.5.30729)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: fr,fr-fr;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.exemple.fr/formulaires/post/formpost.asp
```

```
Cookie: ASPSESSIONIDQCQRRTDB=HPEPPNEBBFGOOCHALDKDIJEH
Content-Type: application/x-www-form-urlencoded
Content-Length: 24
textbox=Test&button1=Submit (1)
```

Observez la dernière ligne (1). Elle comporte le nom de la zone de saisie suivi de la valeur texte contenue au moment de la soumission du formulaire. Le reste de la ligne, séparé par une esperluette (&) comme pour les paramètres passés par un `GET`, contient les informations relatives au bouton d'envoi du formulaire.

Code 9-5 : Réponse à la requête précédente

```
HTTP/1.x 200 OK
Date: Wed, 11 Nov 2009 12:38:43 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
Content-Length: 385
Content-Type: text/html
Cache-Control: private
```

On obtient une réponse standard, semblable à celle de la requête `GET` précédente, contenant le très important code de réponse en première ligne.

Spécifications du protocole

Vous retrouverez la cinématique complète des échanges ainsi que leur forme et contenu exacts dans les RFC associées.

> **NORME Les RFC 1945 et 2616**
>
> Les RFC (*Request For Comments*) sont apparues en 1969 au moment même où les bases de l'Internet se définissaient. Elles représentent une série de notes retraçant toutes les normes existantes ou proposées pour l'Internet. Les RFC sont rédigées à l'initiative d'experts techniques, puis sont revues par la communauté Internet dans son ensemble.
> Pour ceux d'entre vous qui ne connaissent pas trop le protocole HTTP, il est conseillé de jeter un coup d'œil aux RFC suivantes : la RFC 1945, qui traite de la version 1.0 du protocole HTTP, et la RFC 2616 consacrée à la version 1.1 du protocole.
> ▸ http://www.ietf.org/rfc.html

Utilisation de la classe HttpClient

Le kit de développement Android contient le client `HttpClient` d'Apache - dans une version adaptée à Android - qui propose un certain nombre de fonctionnalités pour utiliser le protocole HTTP.

> **CULTURE Le projet HTTP Components d'Apache**
>
> Pour plus d'informations sur la bibliothèque `HttpClient`, consultez le site Apache concernant le projet *HTTP Components*. Nous utiliserons d'ailleurs une partie de ce projet pour résoudre la question du `POST` multiparties.
> ▸ http://hc.apache.org/

Quel que soit le type de requête, l'utilisation de ce client suit le schéma suivant :

1. Création d'une instance de la classe `HttpClient` (de `DefaultHttpClient` plus exactement).
2. Création d'une instance d'un objet requête qui spécialisera notre client.
3. Réglage des propriétés de cette requête.
4. Exécution de la requête grâce à l'instance de type `HttpClient` obtenue auparavant.
5. Analyse et traitement de la réponse.

Requête de type GET

Appliquons donc ce schéma à une requête HTTP de type `GET` qui récupérera une page située à l'adresse passée en paramètre.

Code 9-6 : Récupération d'une page grâce au protocole HTTP et à la classe HttpClient

```
/**
* Méthode qui retourne la page désignée par l'adresse.
* @param adresse Adresse sous forme d'URL.
* @return Contenu de la page sous forme d'une chaîne de caractères.
*/
public String getPage(String adresse) {

    StringBuffer stringBuffer = new StringBuffer("");
    BufferedReader bufferedReader = null;
    try {
        HttpClient httpClient = new DefaultHttpClient(); (1)
         HttpGet httpGet = new HttpGet(); (2)

        URI uri = new URI(adresse);
        httpGet.setURI(uri); (3)

        HttpResponse httpResponse = httpClient.execute(httpGet); (4)
        InputStream inputStream = httpResponse.getEntity().getContent();
        bufferedReader = new BufferedReader(new InputStreamReader(inputStream)); (5)

        String ligneLue = bufferedReader.readLine();
        while (ligneLue != null) { (6)
            stringBuffer.append(ligneLue);
            stringBuffer.append("\n");
```

```
            ligneLue = bufferedReader.readLine();
        }
    } catch (Exception e) {
        Log.e(LOG_TAG, e.getMessage());
    } finally { 7
        if (bufferedReader != null) {
            try {
                bufferedReader.close(); 8
            } catch (IOException e) {
                Log.e(LOG_TAG, e.getMessage());
            }
        }
    }

    return stringBuffer.toString();
}
```

On crée une instance de la classe `DefaultHttpClient` 1 ainsi qu'une instance de la classe `HttpGet` 2 qui correspond à une requête HTTP de type `GET`.

On règle l'URI 3 et il suffit ensuite d'exécuter la requête avec le client créé 4. La lecture de la réponse 5 6 est assez standard, tout comme la fermeture des différents flux 7 8.

> **RAPPEL Passage de paramètres : caractères spéciaux et taille limite**
>
> Pour une requête de type `GET`, si l'on veut passer des paramètres au serveur, on procède en complétant l'URL comme suit :
> `http://www.exemple.com?param1=23&param2=754.`
> La requête précédente est donc composée de deux paramètres, `param1` et `param2`, qui seront passés à la page qui se cache derrière l'URL `http://www.exemple.com`.
> À noter qu'il faut veiller à bien encoder tous les caractères spéciaux (espace, par exemple) et que la taille des données passées via cette méthode est limitée par les serveurs web, notamment pour des raisons de sécurité (attaques par débordement de tampon).

Requête de type POST

Il suffit d'appliquer chaque étape déjà vue pour une requête `GET` à une requête HTTP de type `POST`.

Code 9-7 : Envoi de paramètres en POST grâce à la classe HttpClient

```
StringBuffer stringBuffer = new StringBuffer("");
BufferedReader bufferedReader = null;
try {
    HttpClient httpClient = new DefaultHttpClient(); 1
    HttpPost httpPost = new HttpPost(URL);
```

```
    List<NameValuePair> parametres = new ArrayList<NameValuePair>(); (2)
    parametres.add(new BasicNameValuePair("parametre1", "valeurDuParametre1"));

    UrlEncodedFormEntity formEntity = new UrlEncodedFormEntity(parametres);
    httpPost.setEntity(formEntity); (3)

    HttpResponse httpResponse = httpClient.execute(httpPost); ( (4)
    bufferedReader = new BufferedReader(
        new InputStreamReader(httpResponse.getEntity().getContent()));

  String ligneLue = bufferedReader.readLine();
  while (ligneLue != null) {
    stringBuffer.append(ligneLue);
    stringBuffer.append("\n");
    ligneLue = bufferedReader.readLine();
  }
} catch (Exception e) {
  Log.e(LOG_TAG, e.getMessage());
} finally {
  if (bufferedReader != null) {
    try {
      bufferedReader.close();
    } catch (IOException e) {
      Log.e(LOG_TAG, e.getMessage());
    }
  }
}

Log.i(LOG_TAG, stringBuffer.toString());
```

On crée un client de type `DefaultHttpClient` (1) qui servira à envoyer la requête préconstruite (4). Cette dernière est formée d'une liste de paramètres sous forme de paire nom/valeur (2) associée à un formulaire encodé dans la requête (3).

Requête de type POST multipart

Grâce à une requête HTTP de type `POST`, on peut communiquer des paires clé/valeur, mais on peut également transmettre des paramètres plus complexes comme des fichiers : il s'agit de la requête de type `POST` multiparties (*multipart*).

> NORME **POST multipart – RFC 1521**
>
> Pour plus d'informations, consultez la RFC 1521.

Ce type de requête `POST` n'est pas directement disponible sous Android avec l'API `HttpClient`, mais c'est encore une fois grâce à des bibliothèques Apache que nous allons sortir de l'impasse :

- Commons IO : http://commons.apache.org/io/
- HttpMime : http://hc.apache.org/httpcomponents-client/httpmime/index.html
- Mime4j : http://james.apache.org/mime4j/

Téléchargez ces bibliothèques et liez-les à votre projet pour que les classes que nous allons utiliser par la suite soient reconnues par votre éditeur, et surtout pour qu'il n'y ait pas d'erreur de compilation.

TECHNIQUE Ajout de bibliothèques à un projet

Pour ajouter des bibliothèques à un projet afin qu'elles soient considérées comme des dépendances, la démarche est la même que pour un projet classique : effectuez un clic droit sur la bibliothèque que vous souhaitez ajouter, puis sélectionnez *Build Path>Add to Build Path*.
Vous pouvez également accéder à ces propriétés grâce à un clic droit sur le projet dans l'explorateur de paquetages, puis en sélectionnant *Java Build Path* et enfin *Librairies*.

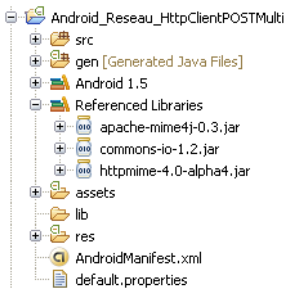


Figure 9–1 Rendu une fois les bibliothèques ajoutées

Une fois les réglages effectués, les archives apparaissent sous la rubrique *Referenced Librairies* de la vue explorateur de paquetages d'Eclipse.

Code 9-8 : Envoi d'une image et d'autres données via un POST multiparties

```
HttpClient httpClient = new DefaultHttpClient();
HttpPost postRequest = new HttpPost("http://www.exemple.com/upload_multi.php"); 1

// Préparation des parties
File fichier = new File("imagePrise.jpg"); ) 2
StringBody infosFichier = new StringBody("Photo du 22 mars 2010");
```

```
// Ajout de ces parties à la requête
MultipartEntity multipartContent = new MultipartEntity();
multipartContent.addPart("infos", infosFichier);
multipartContent.addPart("fichier", new FileBody(fichier)); 3

postRequest.setEntity(multipartContent); 4

HttpResponse response = httpClient.execute(postRequest);

// Fermeture
response.getEntity().getContent().close();
```

Vous devez tout d'abord créer une instance de la classe `HttpPost` 1 dont nous utiliserons la méthode `setEntity` 4. À cette méthode, nous donnerons en paramètre une instance de `MultipartEntity` qui est en fait le corps de la requête multiparties.

La requête est remplie grâce aux différentes parties en invoquant la méthode `addPart` 3. À noter que l'envoi d'un fichier est rendu très simple grâce aux flux 2 et à la signature de la méthode `addPart` prenant ce flux en paramètre 4.

Problématique du multitâches

Les connexions peuvent être réalisées à de nombreuses occasions et dans de multiples parties de l'application :

- vérification de mises à jour au démarrage ;
- envoi de statistiques d'utilisation ou déconnexion à la fin de l'application ;
- consultation des meilleurs scores pour des jeux en accédant au menu adéquat ;
- synchronisation régulière d'information potentiellement un peu partout ;
- chargement de pages ;
- etc.

Mais il y a un impératif à garder en tête : l'application et son lien direct avec l'utilisateur qu'est l'interface graphique doivent toujours répondre ! Si l'on n'utilise pas les capacités multitâches d'Android, on peut se retrouver à attendre qu'une connexion lente s'établisse avec un site distant ou même voir un traitement échouer en cas de coupure (le fameux effet tunnel).

D'ailleurs, si vous développez sur des appareils mobiles, la connexion donnée est par définition intermittente. Il faut absolument en tenir compte.

Sur ce sujet, les principes de base sont les mêmes depuis longtemps :

- réduire les connexions réseau au minimum ;

- proposer un mode dégradé de l'application afin qu'elle puisse fonctionner sans réseau et/ou prévienne l'utilisateur du comportement à adopter (quitter l'application, attendre qu'elle se reconnecte automatiquement, etc.).

Une fois ces bonnes pratiques en tête, n'oubliez pas d'étudier la problématique de programmation en tâche de fond qui est abordée au chapitre 11.

Les services web

Un service web est un élément de logique applicative informatique distribué (le plus souvent disponible sur Internet ou sur un intranet) permettant l'échange de données entre applications et systèmes hétérogènes.

Autour de cette définition, beaucoup de termes circulent : UDDI, WSDL, SOAP, XML-RPC, REST, services web, etc. Faisons le tri et mettons d'un côté SOAP et XML-RPC qui permettent de transporter l'information XML au travers d'une encapsulation XML. D'un autre côté, plaçons REST qui est plus une philosophie du Web et qui, pour résumer, propose de faire beaucoup plus simple que les services traditionnels.

CULTURE Bien architecturer une application REST

N'hésitez pas à lire l'ebook d'Olivier Gutknecht sur la façon de concevoir une application REST
📖 O. Gutknecht, *Bien architecturer une application REST*, Eyrolles 2009 (ebook)

RPC (*Remote Procedure Call*) est un protocole d'appel de procédure distante, autrement dit de fonctions disponibles sur d'autres systèmes qu'on appelle en utilisant des protocoles comme SOAP et XML-RPC.

ATTENTION À propos de XML-RPC

XML-RPC est à la fois le nom d'une technologie d'échange de données XML via un réseau, mais également une dénomination englobant tous ces protocoles d'échange XML. Par exemple, SOAP est un protocole de type XML-RPC (requête de documents XML par appel de procédures à distance).
Ces deux technologies, sont issues du même moule, les deux protocoles ont ensuite évolué séparément.

Ces deux protocoles peuvent donc être utilisés pour invoquer des méthodes à distance. Nous ne traitons que le cas de SOAP dans cette section.

Deux questions se posent alors. Où se trouvent ces services, c'est-à-dire comment trouver un service donné ? Comment utiliser les méthodes du service une fois localisé (quels sont les paramètres à utiliser) ?

Pour répondre à ces deux questions, nous avons :

- UDDI (*Universal Description, Recovery and Integration*) qui est une spécification technique permettant de publier et de rechercher des services ;
- WSDL (*Web Services Description Language*) qui permet de décrire le format des messages requis pour communiquer avec le service déployé.

Derrière tous ces sigles se cache l'idée qu'un programme peut rechercher dans un annuaire un service donné suivant des critères (température dans une ville, calcul complexe, etc.). Après qu'il a localisé le serveur, le programme contacte le service et obtient sa description pour savoir quels paramètres lui envoyer et sous quelle forme sera le résultat. Ensuite, il ne reste qu'à invoquer ce service (en utilisant SOAP) grâce aux informations obtenues.

Ces possibilités sont en fait peu utilisées, mais on a tout de même conservé la description (WSDL) du service qui permet notamment de générer de façon automatique des classes (*stubs*) invoquant le service.

Dans ce livre, nous détaillerons deux approches de service différentes : les services plus traditionnels utilisant SOAP et dont la mécanique a été décrite dans cette partie, et les services REST.

Services basés sur le protocole SOAP

SOAP (*Simple Object Access Protocol*) est un protocole bâti sur XML qui permet la transmission de messages. Il est notamment utilisé dans le cadre d'architectures de type SOA (*Service Oriented Architecture*) pour les services web WS-*.

Le transfert s'effectue le plus souvent à l'aide du protocole HTTP, mais un autre protocole (FTP, SMTP, etc.) peut également être utilisé.

Un message SOAP est composé de trois parties :

- une enveloppe contenant des informations sur le message afin de permettre son acheminement et son traitement ;
- un en-tête, inséré dans l'enveloppe et contenant, par exemple, des directives ou des informations contextuelles liées au traitement du message ;
- un modèle de données au format XML, pour les informations à transmettre, lui aussi intégré à l'enveloppe.

Code 9-9 : Exemple de demande de réservation (source http://www.w3.org/)

```
POST /reservation HTTP/1.1
Host: http://travelcompany.example.org
Content-Type: text/xml; charset=utf-8
Content-Length: length
```

```
SOAPAction: "http://travelcompany.example.org/reservation"

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
<env:Header> (1)
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
          env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
   <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/employees"
          env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <n:name>John Doe</n:name>
  </n:passenger>
</env:Header>
<env:Body> (2)
  <p:itinerary
    xmlns:p="http://travelcompany.example.org/reservation/travel">
   <p:departure> (3)
     <p:departing>New York</p:departing>
     <p:arriving>Los Angeles</p:arriving>
     <p:departureDate>2001-12-14</p:departureDate>
     <p:departureTime>late afternoon</p:departureTime>
     <p:seatPreference>aisle</p:seatPreference>
   </p:departure>
   <p:return> (4)
     <p:departing>Los Angeles</p:departing>
     <p:arriving>New York</p:arriving>
     <p:departureDate>2001-12-20</p:departureDate>
     <p:departureTime>mid-morning</p:departureTime>
     <p:seatPreference/>
   </p:return>
  </p:itinerary>
  <q:lodging
   xmlns:q="http://travelcompany.example.org/reservation/hotels">
   <q:preference>none</q:preference>
  </q:lodging>
</env:Body>
</env:Envelope>
```

Dans cet exemple, la séparation entre chaque partie est nette. Mises à part les lignes concernant le transport, c'est-à-dire l'envoi grâce au protocole HTTP, on voit bien que l'enveloppe englobe tout le document pour contenir les en-têtes (1) et le corps de la requête SOAP (2).

On distingue également bien la partie métier, c'est-à-dire la demande de réservation avec une structure de données découpée en deux sous-parties : les souhaits pour l'aller (3) et les souhaits pour le retour (4).

La réponse du système de réservation est elle aussi tout à fait lisible sans traitement machine, c'est d'ailleurs ce qui fait la force d'un fichier XML bien structuré.

Code 9-10 : Réponse à la demande de réservation de voyage

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
<env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
   <m:dateAndTime>2001-11-29T13:35:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/employees"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <n:name>John Doe</n:name>
  </n:passenger>
</env:Header>
<env:Body>
  <p:itineraryClarification
    xmlns:p="http://travelcompany.example.org/reservation/travel">
   <p:departure>
     <p:departing>
       <p:airportChoices> (1)
            JFK LGA EWR
       </p:airportChoices>
     </p:departing>
  </p:departure>
   <p:return>
     <p:arriving>
       <p:airportChoices> (2)
           JFK LGA EWR
       </p:airportChoices>
     </p:arriving>
  </p:return>
  </p:itineraryClarification>
</env:Body>
</env:Envelope>
```

Si nous mettons à nouveau la partie protocole HTTP de côté, on retrouve les mêmes grands découpages avec l'enveloppe, les en-têtes et le corps du message, c'est-à-dire la réponse métier du système de réservation. En l'occurrence, le service demande à l'utilisateur de faire un choix d'aéroport pour ses déplacements, tant au départ (1) qu'à l'arrivée (2).

POUR ALLER PLUS LOIN Le protocole SOAP

Cet ouvrage n'a pas pour objectif de détailler SOAP. Dans cette partie du livre, nous cherchons à créer un client Android et à mettre en place un service rapidement afin de pouvoir développer. Nous faisons ainsi quelques rappels pour vous replonger dans les services et SOAP en particulier. Si ce domaine ne vous est pas familier, nous vous conseillons de consulter quelques ouvrages sur SOAP, les services web et surtout XML qui est à la base de tout dans ce domaine. Vous pouvez également lire les spécifications du W3C et dont la traduction française est disponible à l'adresse suivante :

- http://www.w3.org/2002/07/soap-translation

SOAP et Android

SOAP présente les avantages suivants :

- encapsulable par HTTP. Grâce au protocole HTTP, il facilite la communication et évite les problèmes liés aux pare-feux et autres serveurs mandataires (proxys) ;
- basé sur XML, il est indépendant des plates-formes cliente et serveur ;
- adaptable à différents protocoles de transport ;
- simple et extensible.

Si on ne s'en tient qu'à cette liste, SOAP est fortement concurrencé par les autres formes de services plus souples comme REST ou tout simplement par tous les mécanismes requête/réponse sur HTTP.

En ajoutant le point noir de SOAP, à savoir qu'il s'agit d'un protocole très verbeux et parfois assez gourmand en ressources, beaucoup le déconseillent sur une plate-forme telle qu'Android, qui proposera un environnement contraint sur des appareils mobiles. Néanmoins il faut se rappeler les points suivants.

- Les services SOAP sont déjà largement déployés. Il n'est donc pas question de les ignorer et il faut donc bien un client SOAP sur Android. Ce qui n'est pas tout à fait vrai car on pourrait se contenter d'un proxy contacté en REST depuis notre client et qui traduirait en SOAP les requêtes, pouvant éventuellement les mettre en cache.
- Le second argument est plus déterminant. SOAP ne permet pas uniquement un simple échange d'informations, c'est également un ensemble de standards pour les transactions, le cryptage, l'orchestration (possibilité d'enchaîner plusieurs services) et bien d'autres. Toutes ces possibilités sont regroupées derrière un terme éloquent : WS-*. Celui-ci vient du grand nombre de spécifications et standards associés qui commencent par WS- comme WS-Security, WS-Transaction, etc.

- Enfin, même si sortis de la boîte Android ils ne proposent pas de fonctionnalités pour manipuler SOAP, les projets libres nous proposent des implémentations qui conjuguées aux API intégrées pour la communication (l'API `HttpClient` étudiée précédemment) nous permettent de résoudre les points de blocage éventuels.

Pour toutes ces raisons, nous avons fait le choix de présenter l'utilisation des services SOAP sur la plate-forme Android.

Mise en place d'un service SOAP avec Tomcat et Axis

Le but de cet ouvrage n'est pas de réaliser ce service. Il existe plusieurs façons de réaliser ce service SOAP en fonction du langage (PHP, Java, etc.). D'ailleurs, il existe en général des bibliothèques ou des frameworks qui vous simplifieront la vie.

Nous allons utiliser une solution aussi simple que rapide en couplant Tomcat et Axis, deux briques logicielles que nous décrirons au moment de leur utilisation.

Pour ceux qui sont plus aguerris et/ou qui ne voudraient pas réaliser toutes ces étapes, le site XMethods (http://www.xmethods.net) propose une liste de services utilisables gratuitement.

Installation de Tomcat

Tomcat est un serveur web capable en autres de traiter du code Java pour créer une réponse dynamique à une requête.

> **EN SAVOIR PLUS** **Tomcat de la fondation Apache**
>
> Tomcat, issu du projet Jakarta, fait partie de l'Apache Software Foundation, une organisation dédiée au développement de logiciels open source. Il est d'ailleurs publié sous la licence Apache Software. C'est un moteur de servlets qui a été conçu en suivant le guide de référence officiel de l'implémentation des technologies *Java Servlet* et *Java Server Pages* (JSP). Les JSP sont des pages contenant du code Java imbriqué dans du HTML. Cette approche est similaire à celle de l'intégration PHP/HTML. Au moment de l'écriture de ce livre, la dernière version de Tomcat est la 6.0.20, disponible sous forme d'archive ou d'exécutable.
> ▸ http://tomcat.apache.org/

Une fois l'installation effectuée, l'URL http://127.0.0.1:8080 doit vous mener à la page d'accueil une fois le serveur démarré (script `startup` dans le répertoire `bin` de l'installation).

En cas de problème, consultez le site http://tomcat.apache.org/. Les deux problèmes qu'on rencontre le plus souvent sont : les exécutables Java ne sont pas référencés dans le path ou la variable d'environnement `JAVA_HOME` est absente ou incorrecte.

Installation d'Axis2

Il faut encore installer Axis, en version 2.

> **En savoir plus** **Axis2**
>
> Axis est un package logiciel complet prenant en charge SOAP. Il permet ainsi de créer des clients, des serveurs ou des passerelles utilisant le protocole SOAP. Axis2 (version réarchitecturée du projet Axis) est distribué sous licence Apache. Au moment de la rédaction de cet ouvrage, la version disponible (depuis octobre 2009) est la 1.5.1 en Java.
> ▸ http://ws.apache.org/axis2/

Téléchargez la distribution en archive WAR (*Web Archive*) et placez-la dans le sous-répertoire `webapp` de votre installation Tomcat qui va automatiquement détecter et déployer l'application. La page d'accueil d'Axis est alors disponible à l'adresse suivante : http://127.0.0.1:8080/axis2.

Nous allons utiliser le service `getVersion` qui permet d'obtenir la version d'Axis installée. Les plus curieux peuvent obtenir le WSDL à l'URL suivante : http://127.0.0.1:8080/axis2/services/Version?wsdl. Il est alors possible d'utiliser des outils qui généreront automatiquement un client (NetBeans, WTK pour Java ME, etc.) ou produiront une interface compatible pour manipuler le service déployé (XMLSpy, etc.).

Notez qu'il existe une multitude de sites proposant une aide pour créer vos propres services, allant d'une simple calculatrice à des services beaucoup plus complexes.

Création du client Android

Adaptation de la bibliothèque kSOAP 2

Pour nous affranchir de la manipulation pure du protocole SOAP, nous utiliserons la bibliothèque kSOAP 2.

> **En savoir plus** **kSOAP 2**
>
> kSOAP est une bibliothèque sous licence BSD construite pour les environnements contraints en ressources comme les applications Java Mobile (J2ME, CDC, CLDC, etc.). kSOAP 2 est une version complètement réarchitecturée par rapport à la version 1. Si nécessaire, gardez un œil sur le projet *ksoap2-android* qui propose également une version modifiée de kSOAP pour Android.
> ▸ http://sourceforge.net/projects/ksoap2

Nous choisissons la bibliothèque complète pour J2SE (pas la version Java Mobile J2ME) et au moment de l'écriture de ce livre il s'agit de la bibliothèque `ksoap2-j2se-full-2.1.2.jar`. Ajoutez-la au projet.

Dans la conception de la bibliothèque, toutes les classes concernant la manipulation des objets SOAP sont bien séparées du moyen de transmission employé sur chaque plate-forme Java : J2S2, J2ME, Android, etc. Ainsi, les classes construisant les requêtes et analysant les réponses ne sont pas à modifier, a priori, puisqu'elles sont faites en utilisant des propriétés classiques de Java.

Cependant, les moyens de transmission sont souvent dépendants de la plate-forme et résultent d'un choix global comme celui qui a été fait pour Android de se baser sur l'API `HttpClient`.

En regardant de plus près, c'est-à-dire le paquetage `org.ksoap2.transport`, on ne trouve pas au premier coup d'œil de classe qui soit de façon évidente compatible avec Android.

Mais en analysant les sources, on s'aperçoit rapidement qu'il n'y a pas de raison pour que la classe `HttpTransportSE` ne puisse pas transmettre correctement, et elle le fait ! Cependant, au lieu de l'utiliser brutalement, il vaut mieux respecter la façon de faire de la bibliothèque et créer nos propres classes. Bien qu'elles n'ajoutent pas de comportement, nous aurons un emploi plus flexible.

Code 9-11 : Code source de la classe AndroidHttpTransport

```
package org.ksoap2.transport;

import java.io.IOException;

public class AndroidHttpTransport extends HttpTransportSE {
    /**
     * @see HttpTransportSE#HttpTransportSE(String)
     */
    public AndroidHttpTransport(String url) {
        super(url);
    }

    /**
     * @see org.ksoap2.transport.HttpTransportSE#getServiceConnection()
     */
    @Override
    protected ServiceConnection getServiceConnection()
        throws IOException {
        return new AndroidServiceConnection(super.url);
    }
}
```

Nous étendons donc la classe `HttpTransportSE` sans changer son comportement. Nous faisons de même avec la seconde classe, `ServiceConnectionSE`, que nous utilisons dans la méthode `getServiceConnection`.

Code 9-12 : Code source de la classe AndroidServiceConnection

```
package org.ksoap2.transport;

import java.io.IOException;

public class AndroidServiceConnection extends ServiceConnectionSE {
    /**
     * @see ServiceConnectionSE#ServiceConnectionSE(String)
     */
    public AndroidServiceConnection(String url)
        throws IOException {
        super(url);
    }
}
```

Et c'est tout !

Utilisation de la bibliothèque adaptée

L'adaptation et l'utilisation de la bibliothèque légèrement modifiée sont très simples. Réalisons un code qui consomme le service mis en place avec Tomcat et Axis.

Code 9-13 : Extrait du code client consommant le service version d'Axis

```
private static final String SOAP_ACTION = "urn:getVersion";
private static final String METHOD_NAME = "getVersion";
private static final String NAMESPACE = "http://axisversion.sample";
private static final String URL =
    "http://192.168.1.1:8080/axis2/services/Version";

private void executeAppelSOAP() {

    SoapObject requete = new SoapObject(NAMESPACE, METHOD_NAME); (1)
    SoapSerializationEnvelope enveloppe = new
        SoapSerializationEnvelope(SoapEnvelope.VER11);
    enveloppe.setOutputSoapObject(requete);

    AndroidHttpTransport androidHttpTransport =
        new AndroidHttpTransport (URL); (2)
    try {
        androidHttpTransport.call(SOAP_ACTION, enveloppe); (3)
        Object resultat = enveloppe.getResponse(); (4)
        // Affichage dans la console de l'émulateur.
        // Exploitez le résultat !
        Log.d("ClientSOAP", "--> " + resultat.toString());
    } catch(Exception e) {
        e.printStackTrace();
    }
}
```

Nous utilisons donc la bibliothèque kSOAP que l'on vient d'adapter à Android.

Il faut tout d'abord créer, avec les classes de base de kSOAP, une requête SOAP sous la forme d'un `SoapObject` (1). Les paramètres du constructeur utilisés sont :

- l'espace de noms (*namespace*) qui est contenu dans les premières lignes du WSDL ;
- le nom de la méthode à consulter.

Ces deux paramètres permettent au service d'aiguiller la requête vers le bon code métier. Ensuite, on crée une enveloppe contenant la requête.

Il ne reste plus qu'à envoyer notre objet SOAP en instanciant notre classe `AndroidHttpTransport` (2) avec en paramètre l'URL du serveur. Si vous avez opté pour l'installation Tomcat/Axis, ce sera en local sur votre ordinateur. Si vous avez choisi d'utiliser un service sur Internet, l'URL à contacter sera précisée.

L'enveloppe SOAP est envoyée grâce à la classe que nous avons implémentée (3).

On extrait ensuite la réponse de l'enveloppe retournée par le service (4). Elle est pour l'exemple rapidement affichée par la suite, à vous de la transformer ou de l'interpréter selon votre besoin (voir figure 9-2).

De façon générale, en cas de doute sur les valeurs des différents éléments à transmettre, consultez le fichier WSDL qui contient toutes les informations nécessaires.

	pid	tag	Message
D	288	dalvikvm	HeapWorker thread shutting down
D	288	dalvikvm	HeapWorker thread has shut down
D	288	jdwp	JDWP shutting down net...
I	288	dalvikvm	Debugger has detached; object registry had 1 entries
D	288	dalvikvm	VM cleaning up
I	53	ActivityManager	Start proc com.eyrolles.android.reseau for activity com.eyr
E	288	AndroidRuntime	ERROR: thread attach failed
D	288	dalvikvm	LinearAlloc 0x0 used 636716 of 5242880 (12%)
D	295	ddm-heap	Got feature list request
D	295	ClientSOAP	--> Hi - the Axis2 version is 1.5
I	53	ActivityManager	Displayed activity com.eyrolles.android.reseau/.ClientSOAP:
D	215	dalvikvm	GC freed 124 objects / 5448 bytes in 56ms
D	93	dalvikvm	GC freed 3142 objects / 181808 bytes in 61ms

Figure 9–2 Aperçu du résultat de notre client SOAP

Les services REST

L'architecture REST part du principe qu'Internet est composé de ressources accessibles à partir d'une URL. En effectuant une requête sur cette URL, une représentation de la ressource demandée sera renvoyée, cette représentation étant traitée par l'application cliente qui se place dans un état donné. Si l'application cliente lance un nouvel appel sur cette ressource, une nouvelle représentation est envoyée. L'application cliente change d'état (*state transfer*) pour chaque représentation de ressource.

> **CULTURE REST, un style d'architecture**
>
> REST (*Representational State Transfer*) est un style d'architecture décrit par Roy Thomas Fielding, l'un des principaux auteurs de la spécification HTTP, dans sa thèse intitulée *Architectural Styles and the Design of Network-based Software Architectures*.

Les réponses sont souvent en XML (ce qui est conseillé), tout comme pour SOAP, mais ce n'est pas une obligation (on peut, par exemple, utiliser des réponses JSON).

Choix et présentation d'un service REST

De plus en plus de sites utilisent des services REST. Un exemple classique est le site Delicious (http://delicious.com/) et son service de gestion de signets. Pour notre part, nous allons utiliser les Yahoo! Search Web Services pour obtenir des réponses à une recherche. Ces services bénéficient d'une sortie des résultats en XML ou en JSON.

Voyons tout de suite un résumé de l'utilisation de cette API pour démarrer rapidement. Le nom d'hôte concerné par nos requêtes de recherche est `http://search.yahooapis.com` et le chemin définissant la méthode utilisée est `/WebSearchService/V1/webSearch`.

Il est également indispensable de passer les paramètres correspondant à notre recherche ainsi que le format de retour escompté. Le paramètre `appid` permet de donner un compte, la valeur `YahooDemo` associée - proposée par Yahoo! pour tests - nous suffira. Le paramètre `query` correspond à la recherche effectuée, `results` au nombre de résultats escomptés et enfin `output` règle le format de sortie.

> **RESSOURCES Yahoo! Search Web Services**
>
> Pour en savoir plus sur la manipulation de cette API, consultez le site de Yahoo! à l'adresse suivante :
> - http://developer.yahoo.com/search

Le client Android JSON associé

JSON (*JavaScript Object Notation*) est un format de données qui permet de représenter de l'information structurée sous forme de texte. Sa représentation est plus légère que le format XML.

Pour obtenir les résultats d'une recherche concernant Android au format JSON, par exemple, nous demandons des informations avec l'URL suivante :

```
http://search.yahooapis.com/WebSearchService/V1/
webSearch?appid=YahooDemo&query=android&results=2&output=json
```

Cette URL demande seulement deux résultats pour faciliter la lecture. Voici un extrait de la réponse proposée par le service.

```
{"ResultSet":
        {"type":"web",
        "totalResultsAvailable":280000000,
        "totalResultsReturned":2,
        "firstResultPosition":1,
        "moreSearch":"\/WebSearchService\/V1\/
webSearch?query=android&amp;appid=YahooDemo&amp;region=us","Result":[{"
Title":"Google Android","Summary":"An Open Source project to develop an
open and free mobile platform. Site includes a blog, SDK, and user
groups.",(...)
```

Nous allons créer un client (sans nous préoccuper de l'interface graphique associée) qui récupérera le nombre de résultats disponibles et le nombre de résultats retournés afin de simplifier le code employé. Une fois la démarche assimilée, il vous sera facile de l'étendre.

Code 9-14 : Client JSON pour une recherche sur Android

```
package com.eyrolles.android.reseau;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
import org.json.JSONException;
import org.json.JSONObject;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class ClientJSON extends Activity {

  private static final String LOG_TAG = "ClientJSON";

  private static final String NOM_HOTE_YAHOO_SEARCH = "http://search.yahooapis.com";
  private static final String PATH_METHODE = "/WebSearchService/V1/webSearch?";
   private static final String PARAMETRES_REQUETE_JSON =
    "appid=YahooDemo&query=android&results=2&output=json"; (1)

  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

```
    // Avec une interface graphique, mettre dans un thread
    resultatsYahooJSON(NOM_HOTE_YAHOO_SEARCH +
      PATH_METHODE + PARAMETRES_REQUETE_JSON);
  }

  public void resultatsYahooJSON(String url) {

    HttpClient httpClient = new DefaultHttpClient();

    try {
      HttpGet httpGet = new HttpGet(url);
      HttpResponse httpResponse = httpClient.execute(httpGet);
      HttpEntity httpEntity = httpResponse.getEntity();

      if ( httpEntity != null) {

        InputStream inputStream = httpEntity.getContent();

          // Lecture du retour au format JSON
        BufferedReader bufferedReader = new BufferedReader(
          new InputStreamReader(inputStream));
        StringBuilder stringBuilder = new StringBuilder();

        String ligneLue = bufferedReader.readLine();
        while (ligneLue != null) {
          stringBuilder.append(ligneLue + "\n");
          ligneLue = bufferedReader.readLine();
        }
        bufferedReader.close();

        // Analyse du retour
        JSONObject jsonObject = new JSONObject(stringBuilder.toString()); (2)
        JSONObject jsonResultSet = jsonObject.getJSONObject("ResultSet"); (3)
        int nombreDeResultatsTotal =
                      jsonResultSet.getInt("totalResultsAvailable"); (4)
        Log.i(LOG_TAG, "Resultats au total " + nombreDeResultatsTotal);
        int nombreDeResultatsRetournes =
                      jsonResultSet.getInt("totalResultsReturned");
        Log.i(LOG_TAG, "Resultats retournes " + nombreDeResultatsRetournes);
       }
      } catch (IOException e) {
        Log.e(LOG_TAG, e.getMessage());
      } catch (JSONException e) {
        Log.e(LOG_TAG, e.getMessage());
      }
    }
}
```

Les paramètres sont fixés ❶ mais pourraient être dynamiquement puisés dans les champs texte d'une interface graphique. Nous avons vu les détails de la réalisation d'une requête HTTP, son résultat est converti en chaîne de caractères, puis passé en paramètre de constructeur à un objet `JSONObject` ❷. Dès lors, nous utilisons les méthodes d'instance pour récupérer un `ResultSet` ❸, puis les informations que nous cherchons ❹.

Le client Android XML associé

Réalisons la même fonctionnalité, à savoir l'obtention des différents nombres de résultats, mais cette fois avec une sortie en XML.

Le paramètre `output` est alors modifié et l'URL devient donc : `http://search.yahooapis.com/WebSearchService/V1/webSearch?appid=YahooDemo&query=android&results=2&output=xml`

Voici un extrait du résultat obtenu :

```
<?xml version="1.0" encoding="UTF-8"?>
<ResultSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:yahoo:srch" xsi:schemaLocation="urn:yahoo:srch http://
api.search.yahoo.com/WebSearchService/V1/WebSearchResponse.xsd"
    type="web"
    totalResultsAvailable="280000000"
    totalResultsReturned="2"
    firstResultPosition="1"
    moreSearch="/WebSearchService/V1/
webSearch?query=android&amp;appid=YahooDemo&amp;region=us"> (…)
```

PERFORMANCE Parseurs XML

Si la performance est un élément déterminant dans votre application, Android propose également un analyseur syntaxique de type SAX. Les classes à considérer sont alors `SAXParserFactory` et `SAXParser`.

Pour rappel, SAX (*Simple API for XML*) travaille les documents XML de manière événementielle. DOM (*Document Object Model*), quant à lui, ne va pas seulement parcourir le document XML, mais il va en plus en fabriquer une représentation en mémoire. Ce type de parcours et de construction est donc beaucoup plus consommateur en temps CPU mais surtout en mémoire. En revanche, il est beaucoup plus agréable à manipuler.

Pour finir, un analyseur syntaxique DOM n'exclut pas l'utilisation de SAX, une implémentation DOM pouvant utiliser SAX pour analyser le fichier XML avant de construire sa représentation en mémoire. C'est pour cette raison que vous vous retrouverez parfois à devoir intercepter une `SAXException` en manipulant un parseur DOM…

Les plus curieux pourront s'intéresser à un troisième type de parseur dit « de type Pull », avec notamment les classes `XmlPullParserFactory` et `XmlPullParser`.

Notre document est petit, les ressources des téléphones récents sont de plus en plus impressionnantes, utilisons un analyseur syntaxique (parseur) de type DOM pour extraire les informations qui nous intéressent du fichier XML renvoyé. Cet analyseur permettra de lire le fichier XML et d'en extraire les balises, leurs attributs ainsi que leur contenu. Il existe différents types de parseurs utilisant des stratégies différentes pour réaliser ces opérations. DOM est une variante qui a pris pour parti d'analyser le document et d'en construire une représentation sous forme d'arbre en mémoire.

Code 9-15 : Client XML pour une recherche sur Android

```
private void connexionParsingXML(String urlParam) {
    URL url;
    try {
        url = new URL(urlParam);
        InputStream inputStream =
            url.openConnection().getInputStream(); (1)
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder(); (2)
        Document dom = builder.parse(inputStream); (3)
        inputStream.close();

        Element racine = dom.getDocumentElement(); (4)
        String totalResultsAvailable =
            racine.getAttribute("totalResultsAvailable"); (5)
        Log.i("Client XML", totalResultsAvailable);
        String totalResultsReturned =
            racine.getAttribute("totalResultsReturned");
        Log.i ("Client XML", totalResultsReturned);

    } catch (MalformedURLException e) {
        Log.e("Client XML", e.getMessage());
    } catch (IOException e) {
        Log.e("Client XML", e.getMessage());
    } catch (SAXException e) {
        Log.e("Client XML", e.getMessage());
    } catch (ParserConfigurationException e) {
        Log.e("Client XML", e.getMessage());
    }
}
```

Il est tout à fait possible de réemployer la même séquence que dans l'exemple JSON précédent pour récupérer une `InputStream` grâce à la classe `HttpClient`. Voici une autre manière de procéder (1).

Classique avec les analyseurs syntaxiques DOM, on récupère une instance de `DocumentBuilder` grâce à la méthode `DocumentBuilderFactory` (2). Cette instance de `DocumentBuilder` accepte en paramètre de sa méthode `parse` une instance de la

classe `InputStream` et charge le document en mémoire pour constituer une instance de `Document` ❸.

Ensuite, une fois la racine récupérée (la balise XML `ResultSet` dans notre exemple) ❹, les attributs sont disponibles ❺ et nous n'avons pas besoin de plus dans l'exemple, mais on pourrait parcourir l'ensemble de la hiérarchie du document.

Les sockets

L'utilisation des sockets, ou connecteurs réseau, est plus confidentielle sur une application mobile puisque de plus bas niveau, mais elle fait partie du large spectre des possibilités. Voici quelques pistes pour les employer.

> **RAPPEL Les sockets**
>
> Un socket est un flux que l'on peut lire ou sur lequel on peut écrire des données brutes. Le concept de socket permet à plusieurs processus de communiquer sur la même machine (en local) ou via un réseau. Pour cela, un socket est identifié par une adresse IP spécifique et un numéro de port.
> Il existe deux grands types d'utilisations et donc de communications :
> - en mode connecté, grâce au protocole TCP, pour établir une connexion durable ;
> - en mode non connecté, grâce au protocole UDP. Un mode non connecté implique qu'il n'y a pas de confirmation de réception des informations et pas de contrôle d'erreur. Ce mode est beaucoup plus performant pour des applications vidéo, par exemple, mais moins fiable.

Le serveur

Tout d'abord, construisons un serveur pour faire face au client que nous comptons développer. Et si nous voulions des réponses fiables à toutes sortes de questions que l'on peut se poser au quotidien ? Le code 9-16 suivant présente le code source d'un serveur qui, à une question posée, associera une réponse. Cette implémentation est indépendante d'Android et est donc réalisable en Java J2SE classique.

Code 9-16 : Serveur qui donne la bonne aventure

```
package fr.eyrolles.android.reseau.serveurs;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.ServerSocket;
```

```
import java.net.Socket;
import java.util.Random;

public class ServeurBonneAventure implements Runnable { (1)

    /** Port à utiliser, de préférence supérieur à 1024 */
    private static final int NUMERO_PORT_SERVEUR = 7777;
    private static final String[] tableauReponses = { (2)
        "Oui", "Non", "Un jour on y verra plus clair"
    };

    /**
     * Méthode run() du thread serveur qui renvoie une réponse
     * à une question.
     */
    public void run() { (3)
        try {
            ServerSocket socketServeur =
                new ServerSocket(NUMERO_PORT_SERVEUR); (4)

            while (true) {
                // Attente d'une connexion cliente
                Socket socketClient = socketServeur.accept(); (5)

                BufferedReader bufferedReader =
                    new BufferedReader(
                    new InputStreamReader(socketClient.getInputStream())); (6)

                String question = bufferedReader.readLine();
                BufferedWriter bufferedWriter =
                    new BufferedWriter(
                    new OutputStreamWriter(socketClient.getOutputStream())); (7)

                String reponse = getReponse(question);
                // Envoi de la réponse
                bufferedWriter.write(reponse + "\n");
                bufferedWriter.flush();
                socketClient.close(); (8)
            }
        } catch (IOException e) {
            System.out.println("Erreur IO : " + e.getMessage());
        }
    }

    /**
     * Retourne une réponse au hasard.
     * @param question Non utilisé en fait. D'où une réponse fiable.
     * @return Retourne une chaîne de caractères contenant la réponse
     * du tableau <code>tableauReponses</code>.
     */
```

```
import java.net.Socket;
import java.util.Random;

public class ServeurBonneAventure implements Runnable { (1)

   /** Port à utiliser, de préférence supérieur à 1024 */
    private static final int NUMERO_PORT_SERVEUR = 7777;
    private static final String[] tableauReponses = { (2)
        "Oui", "Non", "Un jour on y verra plus clair"
    };

    /**
     * Méthode run() du thread serveur qui renvoie une réponse
     * à une question.
     */
    public void run() { (3)
        try {
            ServerSocket socketServeur =
                new ServerSocket(NUMERO_PORT_SERVEUR); (4)

            while (true) {
                // Attente d'une connexion cliente
                Socket socketClient = socketServeur.accept(); (5)

                BufferedReader bufferedReader =
                    new BufferedReader(
                    new InputStreamReader(socketClient.getInputStream())); (6)

                String question = bufferedReader.readLine();
                BufferedWriter bufferedWriter =
                    new BufferedWriter(
                    new OutputStreamWriter(socketClient.getOutputStream())); (7)

                String reponse = getReponse(question);
                // Envoi de la réponse
                bufferedWriter.write(reponse + "\n");
                bufferedWriter.flush();
                socketClient.close(); (8)
            }
        } catch (IOException e) {
            System.out.println("Erreur IO : " + e.getMessage());
        }
    }

    /**
     * Retourne une réponse au hasard.
     * @param question Non utilisé en fait. D'où une réponse fiable.
     * @return Retourne une chaîne de caractères contenant la réponse
     * du tableau <code>tableauReponses</code>.
     */
```

```
    private String getReponse(String question) {
        int nombreReponses = tableauReponses.length;
        Random random = new Random(System.currentTimeMillis());
        return tableauReponses[random.nextInt(nombreReponses)];
    }

    /**
     * Méthode principale pour le démarrage de l'application.
     * @param args Arguments de la ligne de commande.
     */
    public static void main(String[] args) { ❾
        ServeurBonneAventure serveurBonneAventure =
            new ServeurBonneAventure();
        Thread thread = new Thread(serveurBonneAventure);
        thread.start(); ❿
    }
}
```

La classe `ServeurBonneAventure` implémente l'interface `Runnable` ❶. Par ce biais, elle s'engage à donner corps à la méthode `run`. Cette dernière contient le code du serveur et c'est elle qui sera invoquée par le thread, au démarrage de l'application ❾, suite à l'appel de la méthode `start` ❿. On initialise le tableau contenant les réponses possibles ❷. On crée la fonction principale qui renverra la réponse serveur ❸.

La méthode principale du thread crée un `ServeurSocket` ❹ qui, avec la méthode `accept` ❺, se met en attente d'une connexion. L'objectif de ce programme est modeste et son code volontairement allégé. Une possibilité d'amélioration consisterait à déléguer les réponses à un thread pour obtenir un serveur capable de multitâches ou tout du moins de connexions clientes simultanées multiples.

Une fois la connexion du client effectuée, la question est lue ❻, puis une réponse est tirée au sort et écrite avant que la connexion ne soit close et que le serveur n'attende un nouveau client ❺. On crée le flux qui contiendra les données de réponse ❼. Une fois les données écrites, on ferme la connexion ❽.

J2SE n'est pas le sujet du livre, mais plusieurs choses sont à améliorer si vous souhaitez exploiter cet exemple autrement qu'à titre de démonstration :

- il faut déléguer le traitement de la réponse à un thread séparé. Ici le traitement est assez court et vous n'aurez pas beaucoup de connexions, mais le serveur n'est plus disponible pour accepter les requêtes pendant que vous récupérez la réponse aléatoire ;
- la boucle principale est infinie et sans condition de sortie. Prévoyez un mécanisme pour pouvoir arrêter proprement le thread principal.

Le client Android

En face de ce serveur, voici le client dont nous détaillons uniquement la méthode `poserQuestion`.

Code 9-17 : Extrait de code pour le client du serveur de voyance

```
private String poserQuestion(String serveur, int port,
                                        String question) throws IOException {

    Socket socketClient = new Socket(serveur, port); (1)

    BufferedWriter bufferedWriter = new BufferedWriter(
        new OutputStreamWriter(socketClient.getOutputStream())); (2)
    BufferedReader bufferedReader = new BufferedReader(
        new InputStreamReader(socketClient.getInputStream())); (3)

    bufferedWriter.write(question +"\n");
    bufferedWriter.flush();

    return bufferedReader.readLine();
}
```

Finalement, ce code est assez simple puisqu'il revient à créer un socket (1) avec les bonnes informations d'adresse et de port de connexion, puis à écrire la question (2) avant de lire la réponse (3) via l'utilisation classique des flux Java.

Un exemple d'invocation de cette méthode pourrait donc être obtenue grâce au code suivant :

```
String reponse = poserQuestion("192.168.1.1", 7777, "Une question");
```

> **PIÈGE À ÉVITER Utilisation de l'émulateur**
>
> Il faut utiliser une adresse différente de celle du loopback traditionnel en 127.0.0.1 puisque l'émulateur pense que c'est lui qu'on contacte quand un programme fait appel à cette adresse depuis un programme qu'il exécute. Utilisez la commande `ipconfig` sous Windows ou `ifconfig` sous Linux pour obtenir l'adresse IP externe dont vous aurez besoin.

Côté permissions, puisque nous avons besoin d'une connexion, il suffit d'ajouter `android.permission.INTERNET` à la section `manifest` du fichier de configuration de l'application comme suit :

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

En résumé

Dans ce chapitre, nous avons abordé l'utilisation de sockets dans un client Android, même si cet emploi est de plus en plus rare au profit de techniques consommant des services HTTP ou des services XML plus traditionnels.

En parlant de services HTTP, nous avons employé l'API Apache `HttpClient` intégrée qui est une ressource de programmation réseau incontournable. En mettant en œuvre ce composant, nous avons utilisé plusieurs types de requêtes pour consommer divers types de réponses.

Pour aller plus loin, nous avons également créé des clients pour des services web classiques consommant des messages SOAP, puis nous avons étudié des services REST et leurs messages JSON.

Toutes ces étapes nous ont amené à utiliser la plus grande partie des méthodes de communication entre client mobile et serveur. Vous avez maintenant toutes les cartes en mains pour créer vos propres clients qui accéderont à l'immense quantité d'informations disponible sur Internet.

10

Les graphismes 3D avec OpenGL ES

Aujourd'hui la 3D est devenu un standard pour les jeux et les applications de visualisation. La plate-forme Android supporte pleinement ce type d'affichage et, de plus, facilite la tâche au développeur un tant soit peu familier avec OpenGL ES.

Il y a quelques années encore, il semblait peu concevable de voir des objets en trois dimensions sur un téléphone mobile. Aujourd'hui, avec Android, cela semble être une époque définitivement révolue ! La plate-forme embarque en effet OpenGL ES, la version embarquée d'OpenGL, cette API graphique 3D ouverte et encore largement utilisée aujourd'hui sur les ordinateurs de bureau et notamment les jeux.

Dans ce chapitre, vous découvrirez comment créer des applications tirant partie de la 3D sur la plate-forme Android. De la création d'une simple forme jusqu'à l'application d'une texture, OpenGL ES vous étonnera.

La 3D sur Android avec OpenGL ES et la vue GLSurfaceView

OpenGL avec Android

L'utilisation d'OpenGL ES 1.0 est possible depuis la version 1.1 d'Android. Cependant, devant la complexité à intégrer ce type de rendu, les développeurs de la plate-forme ont réfléchi à la manière de rendre son utilisation plus simple. Là où il fallait

auparavant configurer les différents éléments de rendu, de gestion des entrées et du cycle de vie de l'activité, il ne faut maintenant plus intégrer qu'un simple composant.

CULTURE Historique d'OpenGL ES

Dès son origine, *OpenGL(R) Embedded Systems* a été conçue pour être une API libre, gratuite et multi-plate-forme pour les graphiques 2D/3D destinés aux appareils embarqués (téléphones, consoles, automobiles, etc.). Elle représente un sous-ensemble de la version bureautique d'OpenGL proposant une interface entre le matériel d'accélération graphique et les applications.
La version 1.x d'OpenGL ES offre une accélération pour un ensemble de fonctions figées (par opposition à la version 2.x détaillée plus loin) en se basant sur les spécifications d'OpenGL 1.5. Les fonctionnalités de cette version améliorent le rendu graphique tout en réduisant l'utilisation de la bande passante de la mémoire de façon à économiser la batterie des appareils. Tout comme son pendant bureautique, OpenGL ES supporte également quelques fonctions d'extension propres aux constructeurs qui souhaitent implémenter une fonctionnalité spécifique dans leur matériel.
La version 2.x d'OpenGL ES - pour le moment non supportée par Android et non approfondie dans ce présent livre - est quant à elle basée sur les spécifications 2.0 d'OpenGL et permet la programmation de rendu 3D avec la possibilité d'exploiter la puissance des *shaders* (programme permettant de paramétrer une partie du processus de rendu réalisé par une carte graphique ou un moteur de rendu logiciel) et du langage *OpenGL ES Shading Language*. Cependant, cette version ne supporte pas les fonctions fixes de transformation et le pipeline d'OpenGL ES 1.x.

Figure 10–1
Le logo officiel d'OpenGL ES
(tous droits réservés OpenGL(R) ES)

La version 1.5 d'Android a permis aux développeurs de repenser complètement la surface de rendu 3D pour en proposer une plus simple à intégrer : la vue `android.opengl.GLSurfaceView`. Cette dernière apporte un certain nombre de facilités par rapport à Android 1.1 :

- liaison transparente entre OpenGL ES et le système de vue d'Android ;
- gestion transparente d'OpenGL ES avec le cycle de vie de l'activité ;
- choix simplifié d'un format de tampon de pixel adéquat ;
- création et gestion simplifiée d'un thread séparé pour rendre la surface OpenGL ES de façon fluide ;
- ajout d'outils de débogage pour tracer tous les appels aux API OpenGL ES.

Grâce à ces apports non négligeables, le développeur Android 1.5 pourra réaliser très rapidement une application 3D fiable, que ce soit un jeu ou une application de visualisation scientifique, ou toute autre application.

L'utilisation d'OpenGL ES requiert des connaissances de l'API qui vont bien au-delà du périmètre de ce livre. Si certaines commandes ou concepts vous semblent obscurs, n'hésitez pas à vous reporter à la bibliographie en fin d'ouvrage. En effet, cette partie n'a pas pour vocation de faire de vous un développeur OpenGL, mais simplement de vous permettre de comprendre, d'interagir et d'utiliser OpenGL avec Android.

La vue `GLSurfaceView` représente la surface de l'application sur laquelle vous allez rendre votre scène 3D. Cette vue gère pour vous toutes les interactions avec l'environnement extérieur que représente l'application. En revanche, elle délègue son rendu à un objet de type `GLSurfaceView.Renderer`.

L'objet de rendu GLSurfaceView.Renderer

L'interface `GLSurfaceView.Renderer` comprend trois méthodes à implémenter :

- `onSurfaceCreated` : méthode appelée lorsque la surface est créée ou recréée, correspondant respectivement à l'exécution du thread de rendu et la perte du contexte EGL. Le corps de cette méthode est l'emplacement de prédilection pour placer toutes les créations de ressources (textures, modèles 3D, etc.). Notez que les ressources EGL, telles que les textures, sont automatiquement supprimées lorsque le contexte EGL est perdu ;
- `onSurfaceChanged` : méthode appelée lorsque la surface de rendu change de taille et après que la surface a été créée. Votre zone de vue et votre matrice de projection (si nécessaire) seront idéalement spécifiées dans le corps de cette méthode ;
- `onDrawFrame` : méthode appelée pour rendre l'image courante. L'implémentation minimale de cette méthode consiste à effacer les tampons de couleurs et de profondeur :
  `gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT)`.

Le code suivant présente le squelette minimal d'une classe implémentant l'interface `GLSurfaceView.Renderer`.

Code 10-1 : Squelette minimal de l'application

```
import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;

class ClearRenderer implements GLSurfaceView.Renderer {
    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
        // Ne rend rien.
    }

    public void onSurfaceChanged(GL10 gl, int w, int h) {
        gl.glViewport(0, 0, w, h);
```

```
    }

    public void onDrawFrame(GL10 gl) {
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
    }
}
```

Ce code illustre l'implémentation minimale des trois méthodes de l'objet de rendu. Nous reviendrons sur les commandes OpenGL utilisées dans les prochaines parties.

Intégrer la surface de rendu OpenGL ES dans une activité

Voyons maintenant comment réaliser l'activité qui intégrera la surface de rendu OpenGL. La simplicité apportée par Android 1.5 et la vue `GLSurfaceView` permet d'intégrer en quelques lignes le composant dans une activité.

Code 10-2 : Intégration de GLSurfaceView dans l'application

```
import android.app.Activity;
import android.opengl.GLSurfaceView;
import android.os.Bundle;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;

public class Demo3D extends Activity {

   private GLSurfaceView mGLView;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      mGLView = new GLSurfaceView(this);
      mGLView.setRenderer(new ClearRenderer());
      setContentView(mGLView);
   }

   @Override
   protected void onPause() {
      super.onPause();
      mGLView.onPause();
   }

   @Override
   protected void onResume() {
      super.onResume();
      mGLView.onResume();
```

```
    }

    class ClearRenderer implements GLSurfaceView.Renderer {
        public void onSurfaceCreated(GL10 gl, EGLConfig config) {
            // Ne rend rien.
        }

        public void onSurfaceChanged(GL10 gl, int w, int h) {
            gl.glViewport(0, 0, w, h);
        }

        public void onDrawFrame(GL10 gl) {
            gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
        }
    }
}
```

Cette activité n'affiche pour le moment qu'un écran noir. L'application efface l'écran à chaque image. Le cycle de vie de l'activité est correctement géré : le rendu est mis en pause lorsque l'application l'est aussi, et reprend dès que l'application redevient active.

Pour ajouter des objets de rendu à la surface OpenGL, ajoutez des appels aux API dans la méthode `onDrawFrame` de la classe de rendu.

Notez que pour le moment, cette application ne prend pas en compte les entrées de l'utilisateur, elle se contente de se rafraîchir et d'afficher le rendu.

Les formes 3D avec OpenGL ES

La bibliothèque OpenGL possède une multitude de fonctions permettant d'afficher des primitives ou polygones (triangles, carrés, etc.) sur la surface de rendu.

Une primitive est constituée d'un ensemble de sommets qui, reliés par des lignes, forment une face. Chaque sommet possède des coordonnées, une sur chaque axe du plan X, Y, Z, que l'on peut représenter comme à la figure 10-2.

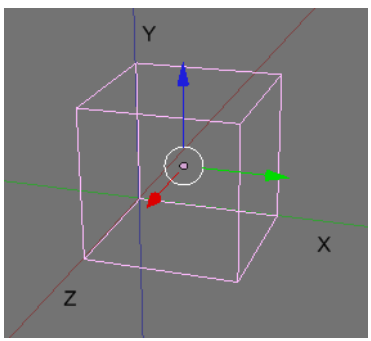


Figure 10–2
Les sommets s'inscrivent dans le plan en trois dimensions selon les axes X, Y et Z.

EN SAVOIR PLUS Documentation sur OpenGL ES

Ce livre n'a pas pour objectif de vous présenter OpenGL ES dans sa totalité, mais seulement de vous introduire aux commandes de base pour réaliser votre première application OpenGL le plus rapidement possible. Pour de plus amples informations, nous vous conseillons de consulter la référence OpenGL ES sur le site Khronos Group :

- http://www.khronos.org/opengles/

Les tableaux de sommets, de faces et de couleurs

Afin d'afficher des formes 3D à l'écran, vous pourriez très bien utiliser différentes primitives, une pour chaque face de la forme par exemple. Cependant, ce procédé se révélerait très consommateur en ressources puisque chaque face ferait l'objet d'un appel à une fonction. Pour pallier ce problème, OpenGL dispose de méthodes permettant de rendre un ensemble de sommets en un minimum d'appels.

Les méthodes d'OpenGL permettent de stocker des faces et des sommets directement dans des tableaux offrant ensuite la possibilité de demander le rendu à l'aide d'une seule et même méthode. Cela permet de gagner un temps important, surtout avec Android qui utilise une interface de communication Java (et donc non système).

Un tableau de 3 sommets stocke les informations de la façon suivante pour définir une face :

x1	y1	z1	x2	y2	z2	x3	y3	z3

Dans le cas d'un tableau de faces, les index des sommets formant une face seront ordonnés de la façon suivante :

face1_index_sommet1	face1_index_sommet2	face1_index_sommet3

Maintenant, effectuons notre premier rendu 3D. Cet exemple réutilise la structure de l'exemple précédent en y ajoutant une classe permettant à un objet de la scène d'être rendu de façon autonome.

Code 10-3 : Application de rendu d'une primitive 3D

```
public class DemoTriangleRender extends Activity {

   private GLSurfaceView mGLView;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      mGLView = new GLSurfaceView(this);
```

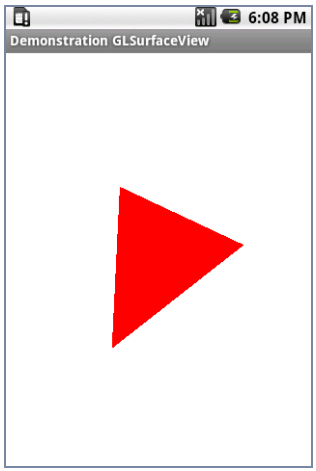
```
        mGLView.setRenderer(new TriangleRenderer(this));
        setContentView(mGLView);
    }

    @Override
    protected void onPause() {
        super.onPause();
        mGLView.onPause();
    }

    @Override
    protected void onResume() {
        super.onResume();
        mGLView.onResume();
    }
}

class TriangleRenderer implements GLSurfaceView.Renderer {

    private Triangle mTriangle;

    private float angle = 0.f;

    public TriangleRenderer(Context context) {
        mTriangle = new Triangle();
    }

    @Override
    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
        // Nous spécifions la couleur de vidage du tampon de couleur, ici blanc.
        gl.glClearColor(1.0f, 1.0f, 1.0f, 1);

        // Nous activons la gestion des tableaux de sommets.
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    }

    @Override
    public void onDrawFrame(GL10 gl) {
        // Vide le tampon de couleur et applique la couleur par défaut.
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT);

        // Remplace la matrice courante transformée par une matrice
        // d'identité. Cette opération permet de supprimer toutes les
        // transformations (rotation, translation, etc.) réalisées sur la
        // la précédente matrice.
        gl.glLoadIdentity();
        // Effectue une rotation de la matrice sur les axes X et Z.
        gl.glRotatef(angle++, 1.0f, 0, 1.0f);
```

```
        // Dessine le triangle.
        mTriangle.draw(gl);
    }

    @Override
    public void onSurfaceChanged(GL10 gl, int w, int h) {
        // Détermine la surface de rendu.
        gl.glViewport(0, 0, w, h);
    }
}

class Triangle {

    private final static int NB_SOMMETS = 3;

    // Utilisation de tampons pour stocker le sommet et les index des faces.
    private FloatBuffer mFTamponSommet;
    private ShortBuffer mSTamponIndex;

    // Les sommets (X, Y, Z).
    float[] sommets = { -0.5f, -0.25f, 0, 0.5f, -0.25f, 0, 0.0f, 0.5f, 0 };

    // Index des sommets.
    short[] index = { 0, 1, 2 };

    public Triangle() {
        // Alloue la mémoire pour le stockage des sommets et des index.
        // L'allocation doit être directe pour que le garbage collector ne
        // puisse pas venir modifier les données dans le tas.
        ByteBuffer vbb = ByteBuffer.allocateDirect(NB_SOMMETS * 3 * 4);
        vbb.order(ByteOrder.nativeOrder());
        mFTamponSommet = vbb.asFloatBuffer();
        mFTamponSommet.put(sommets);

        ByteBuffer ibb = ByteBuffer.allocateDirect(NB_SOMMETS * 2);
        ibb.order(ByteOrder.nativeOrder());
        mSTamponIndex = ibb.asShortBuffer();
        mSTamponIndex.put(index);

        mFTamponSommet.position(0);
        mSTamponIndex.position(0);
    }

    public void draw(GL10 gl) {
        // Spécifie le tampon contenant les sommets à utiliser.
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, mFTamponSommet);
        // Spécifie la couleur des prochaines faces.
        gl.glColor4f(1.f, 0f, 0f, 1.f);
        // Dessine les faces contenues dans le tampon des index à partir
        // des sommets stockés dans le tampon des sommets.
```

```
            gl.glDrawElements(GL10.GL_TRIANGLE_STRIP, NB_SOMMETS,
               GL10.GL_UNSIGNED_SHORT, mSTamponIndex);
        }
    }
```

Ce code permet d'afficher un écran avec un triangle de couleur rouge effectuant une rotation sur deux axes.

Figure 10–3
Exemple d'affichage d'une primitive avec OpenGL ES



En plus de pouvoir spécifier les sommets et les index composant les faces de votre objet, vous pouvez également déterminer les couleurs de chaque sommet. Le code suivant permet d'afficher un cube dont chaque sommet possède une couleur définie dans un tableau de couleurs.

Code 10-4 : Application de rendu d'un objet 3D et gestion de la couleur

```
class CubeRenderer implements GLSurfaceView.Renderer {

    private Cube mCube;

    private float angle = 0.f;

    public CubeRenderer(Context context) {
        mCube = new Cube();
    }

    @Override
    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
        // Nous spécifions la couleur de vidage du tampon de couleur, ici blanc.
        gl.glClearColor(1.0f, 1.0f, 1.0f, 1);
```

```
        // Nous activons la gestion des tableaux de sommets et de couleurs.
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glEnableClientState(GL10.GL_COLOR_ARRAY);

        // Calcul de la couleur des sommets selon Gourraud (dégradé issu de
        // l'interpolation des couleurs entre deux sommets).
        gl.glShadeModel(GL10.GL_SMOOTH);
        // Active le test de profondeur.
        gl.glEnable(GL10.GL_DEPTH_TEST);
    }

    @Override
    public void onDrawFrame(GL10 gl) {
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);

        gl.glLoadIdentity();
        gl.glRotatef(angle++, 1.0f, 0, 1.0f);

        mCube.draw(gl);
    }

    @Override
    public void onSurfaceChanged(GL10 gl, int w, int h) {
        // Détermine la surface de rendu.
        gl.glViewport(0, 0, w, h);
    }
}

class Cube {
    private FloatBuffer mTamponSommets;
    private FloatBuffer mTamponCouleurs;
    private ByteBuffer mTamponIndex;

    float unite = 0.5f;
    float vertices[] = { -unite, -unite, -unite, unite, -unite, -unite, unite,
unite, -unite, -unite, unite, -unite, -unite, -unite, unite, unite, -unite,
unite, unite, unite, unite, -unite, unite, unite};

    // Les couleurs sont données.
    float couleurs[] = { 0, 0, 0, 1.f, 1.f, 0, 0, 1.f, 1.f, 1.f, 0, 1.f, 0, 1.f,
0, 1.f, 0, 0, 1.f, 1.f, 1.f, 0, 1.f, 1.f, 1.f, 1.f, 1.f, 1.f, 0, 1.f, 1.f, 1.f};

    byte indexFaces[] = { 0, 4, 5, 0, 5, 1, 1, 5, 6, 1, 6, 2, 2, 6, 7, 2, 7, 3,
3, 7, 4, 3, 4, 0, 4, 7, 6, 4, 6, 5, 3, 0, 1, 3, 1, 2 };

    public Cube() {
        ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length * 4);
        vbb.order(ByteOrder.nativeOrder());
```

```
        mTamponSommets = vbb.asFloatBuffer();
        mTamponSommets.put(vertices);
        mTamponSommets.position(0);

        ByteBuffer cbb = ByteBuffer.allocateDirect(couleurs.length * 4);
        cbb.order(ByteOrder.nativeOrder());
        mTamponCouleurs = cbb.asFloatBuffer();
        mTamponCouleurs.put(couleurs);
        mTamponCouleurs.position(0);

        mTamponIndex = ByteBuffer.allocateDirect(indexFaces.length);
        mTamponIndex.put(indexFaces);
        mTamponIndex.position(0);
    }

    public void draw(GL10 gl) {
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, mTamponSommets);
        // On spécifie le tampon de couleur à utiliser.
        gl.glColorPointer(4, GL10.GL_FLOAT, 0, mTamponCouleurs);
        gl.glDrawElements(GL10.GL_TRIANGLES, 36, GL10.GL_UNSIGNED_BYTE,
            mTamponIndex);
    }
}
```

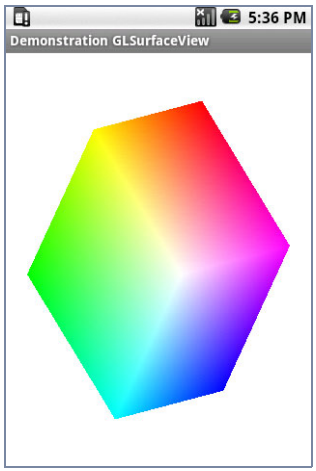


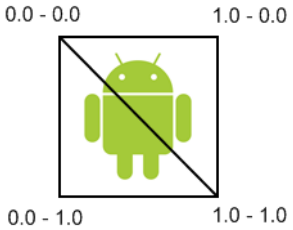
Figure 10–4
Affichage d'un cube coloré avec OpenGL ES

Les textures

Pour donner une apparence réaliste à un objet 3D, la seule utilisation des couleurs ne suffit pas. C'est pourquoi OpenGL ES permet de recourir aux textures qui vous permettront d'appliquer une image sur une face.

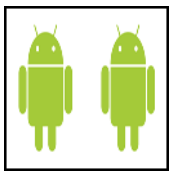
Chaque sommet se voit alors attribuer des coordonnées U et V spécifiant les coordonnées X et Y de la texture. Les coordonnées d'une texture permettent de placer précisément cette dernière sur une face. Le plus souvent, vous spécifierez les coordonnées en utilisant des nombres réels allant de 0 à 1, soit 0.0 à 1.0 comme sur la figure ci-jointe.

Figure 10–5
Coordonnées en nombres à virgule flottante d'une texture 2D



Vous pouvez spécifier des coordonnées inférieures à 0.0 ou supérieures à 1.0. Cela décalera d'autant de fois la texture soit dans un sens, soit dans l'autre. Si, par exemple, vous spécifiez la coordonnée U du coin gauche à 0.0 et celle du coin droit à 2.0, vous indiquez à OpenGL ES d'afficher deux fois la texture sur la largeur.

Figure 10–6
Texture spécifiée avec une coordonnée U de 2.0



ZOOM Les mipmaps

Les mipmaps d'une texture représentent les textures redimensionnées par OpenGL ES à partir de la texture originale et qui serviront à rendre la texture en fonction de la distance entre la face et la caméra. Plus la face est éloignée du point de vue, plus OpenGL ES se servira d'une mipmap petite. Cela permet à OpenGL de dépenser moins de ressources processeur à appliquer la texture et surtout, d'améliorer le rendu graphique de sorte qu'il n'y ait pas ou peu de variations visibles entre deux images pour une face sur laquelle la texture est appliquée.

Pour utiliser la gestion des textures dans votre application, vous devez tout d'abord activer son utilisation avec la commande :

```
gl.glEnable(GL10.GL_TEXTURE_2D);
```

De même que pour les sommets et les couleurs, l'utilisation d'un tableau de coordonnées de textures est incontournable et il vous revient de l'activer :

```
gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
```

Pour que OpenGL ES puisse utiliser une texture et l'appliquer, vous devez d'abord la charger en mémoire.

Charger une texture

Pour charger une texture, vous devez tout d'abord générer des identifiants uniques permettant à OpenGL de savoir avec quelle texture il va travailler.

```
int[] textures = new int[1];
// Génération des id de textures (dans notre cas, un seul).
gl.glGenTextures(1, textures, 0);

// Spécifie à OpenGL ES que nous travaillons avec la première texture.
mTextureID = textures[0];
gl.glBindTexture(GL10.GL_TEXTURE_2D, mTextureID);
```

Une fois cela réalisé, vous utiliserez ces identifiants pour spécifier à OpenGL ES avec quelle texture il doit opérer. Nous verrons un peu plus tard comment gérer cela pour les rendus.

Une fois que vous avez obtenu les identifiants de texture nécessaires à votre application, vous allez pouvoir paramétrer certaines propriétés de rendu des textures sur les faces. Parmi ces propriétés, vous pourrez spécifier la façon dont OpenGL ES utilise les mipmaps, la répétition des textures sur les faces et le mode de mélange des couleurs entre les différents niveaux de textures.

```
// Création des mipmaps (génération de textures de dimensions intermédiaires).
gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MIN_FILTER,
      GL10.GL_NEAREST);
gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MAG_FILTER,
      GL10.GL_LINEAR);

// Paramétrage de l'application des textures sur les faces.
gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_S,
      GL10.GL_CLAMP_TO_EDGE);
gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_T,
      GL10.GL_CLAMP_TO_EDGE);
gl.glTexEnvf(GL10.GL_TEXTURE_ENV, GL10.GL_TEXTURE_ENV_MODE,
      GL10.GL_REPLACE);
```

Vous devez à présent charger l'image que vous souhaitez utiliser et créer la texture. Pour créer une texture et effectuer toutes les opérations en une seule, vous pouvez vous aider de la classe utilitaire `GLUtils` :

```
// Accès à la ressource et chargement de l'image en tant que texture.
InputStream is = mContext.getResources().openRawResource(
      R.drawable.robot);
Bitmap bitmap;
try {
   bitmap = BitmapFactory.decodeStream(is);
} finally {
   try {
      is.close();
   } catch (IOException e) {}
}

GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);
bitmap.recycle();
```

L'appel à la méthode `recycle` de la classe `Bitmap` est ici optionnel mais conseillé, surtout sur des applications destinées aux appareils mobiles. Cette méthode libère la mémoire occupée par les pixels et marque l'image comme morte, c'est-à-dire que vous ne pourrez plus l'utiliser par la suite. Si vous n'appelez pas cette méthode, le ramasse-miettes (*garbage collector*) devra attendre que vous n'ayez plus de référence vers l'objet pour libérer la mémoire.

> **À NOTER Dimension des textures**
>
> Vous aurez sûrement remarqué que nous spécifions souvent la constante `GL_TEXTURE_2D` dans les différentes méthodes. L'API OpenGL standard autorise en effet l'utilisation de textures à 1, 2 ou 3 dimensions. C'est grâce à ce paramètre que nous indiquons à OpenGL ES que nous souhaitons opérer avec des textures 2D. Les textures à 1 et 3 dimensions servent beaucoup dans le domaine des applications médicales alors que les textures 2D sont généralement utilisées pour les jeux et autres applications métiers.

Générez autant d'identifiants de texture que nécessaire. Néanmoins, faites attention à la gestion des ressources mémoire et notamment des textures, qui altèrent les performances, mais aussi à l'indisponibilité des ressources les moins utilisées dans certains cas.

Utiliser et appliquer une texture

OpenGL ES permet l'application de plusieurs niveaux de texture sur une même face. Cette possibilité ouvre la voie à de nombreux effets graphiques (par exemple dans un jeu, un impact de projectile sur un mur de briques).

Pour spécifier à quel niveau de texture vous travaillez, vous utiliserez la commande `glActiveTexture` avec un paramètre représentant le niveau (le niveau zéro représente le premier niveau) tel que `GL_TEXTURE0`, `GL_TEXTURE1`, etc.

```
// Spécifie l'utilisation du premier niveau de texture.
gl.glActiveTexture(GL10.GL_TEXTURE0);
```

Pour spécifier à OpenGL ES quelle texture vous souhaitez utiliser dans le rendu des prochaines primitives, vous devez utiliser la méthode `glBindTexture`. À chaque fois que vous souhaitez changer de texture, vous devez appeler cette méthode en spécifiant l'identifiant de la texture adéquate.

```
gl.glBindTexture(GL10.GL_TEXTURE_2D, idTextureAAppliquer);
```

Spécifier les coordonnées de texture

L'application d'une texture sur une face requiert pour chaque sommet de posséder deux valeurs supplémentaires en plus des coordonnées X, Y et Z : les coordonnées de texture U et V spécifiant le positionnement de la texture.

Ces coordonnées de texture peuvent être spécifiées dans un tableau de la même manière que les coordonnées des sommets en utilisant la méthode `glTexCoordPointer`. Cette dernière prend en arguments le nombre de coordonnées de texture par sommet, le type de données des valeurs et le tableau contenant les coordonnées de texture :

```
gl.glVertexPointer(3, GL10.GL_FLOAT, 0, mTamponSommets);
gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, mTamponTexCoords);
gl.glDrawElements(GL10.GL_TRIANGLES, 36, GL10.GL_UNSIGNED_BYTE, mTamponIndex);
```

Voici le code complet de l'implémentation d'un `Renderer` affichant un cube avec une texture sur ses côtés.

Code 10-5 : Application de rendu d'une texture sur des faces

```
class CubeTextureRenderer implements GLSurfaceView.Renderer {

   private Context mContext;
   private CubeTexture mCubeTexture;
   private int mTextureID;

   private float angle = 0.f;
```

```
public CubeTextureRenderer(Context context) {
   mContext = context;
   mCubeTexture = new CubeTexture();
}

@Override
public void onSurfaceCreated(GL10 gl, EGLConfig config) {
   // Fond de couleur noire.
   gl.glClearColor(0, 0, 0, 1);

   gl.glEnable(GL10.GL_DEPTH_TEST);

   // Active la gestion des textures.
   gl.glEnable(GL10.GL_TEXTURE_2D);

   gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);

   // Active les tableaux de coordonnées de texture.
   gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);

   int[] textures = new int[1];

   /*
    * Création de la texture (cela doit être réalisé pour chaque texture).
    */

   // Génération des id de textures (dans notre cas, un seul).
   gl.glGenTextures(1, textures, 0);

   // Création de la première texture.
   mTextureID = textures[0];
   gl.glBindTexture(GL10.GL_TEXTURE_2D, mTextureID);

   // Création des mipmaps.
   gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MIN_FILTER,
      GL10.GL_NEAREST);
   gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MAG_FILTER,
      GL10.GL_LINEAR);

   // Paramétrage de l'application des textures sur les faces.
   gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_S,
      GL10.GL_CLAMP_TO_EDGE);
   gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_T,
      GL10.GL_CLAMP_TO_EDGE);
   gl.glTexEnvf(GL10.GL_TEXTURE_ENV, GL10.GL_TEXTURE_ENV_MODE,
      GL10.GL_REPLACE);

   // Accès à la ressource et chargement de l'image en tant que texture.
   InputStream is = mContext.getResources().openRawResource(
      R.drawable.robot);
```

```
        Bitmap bitmap;
        try {
            bitmap = BitmapFactory.decodeStream(is);
        } finally {
            try {
                is.close();
            } catch (IOException e) {}
        }

        GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);
        bitmap.recycle();
    }

    @Override
    public void onDrawFrame(GL10 gl) {
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
        gl.glLoadIdentity();

        // Spécifie l'utilisation du premier niveau de texture.
        gl.glActiveTexture(GL10.GL_TEXTURE0);
        // Spécifie à OpenGL l'utilisation de la texture possédant l'identifiant
        // indiqué
        gl.glBindTexture(GL10.GL_TEXTURE_2D, mTextureID);

        gl.glRotatef(angle++, 1, 1, 1);

        mCubeTexture.draw(gl);
    }

    @Override
    public void onSurfaceChanged(GL10 gl, int w, int h) {
        gl.glViewport(0, 0, w, h);
    }
}

class CubeTexture {
    private FloatBuffer mTamponSommets;
    private FloatBuffer mTamponTexCoords;
    private ByteBuffer mTamponIndex;

    float unite = 0.5f;
    // Les coordonnées des sommets (X, Y, Z).
    float vertices[] = { -unite, -unite, -unite, unite, -unite, -unite, unite,
        unite, -unite, -unite, unite, -unite, -unite, -unite, unite, unite,
        -unite, unite, unite, unite, unite, -unite, unite, unite };

    // Les coordonnées de texture (U, V).
    float texCoords[] = { 0, 0, 1.0f, 0, 1.0f, 1.0f, 0, 1.0f, 0, 1.0f, 1.0f,
        1.0f, 1.0f, 0, 0, 0 };
```

```
    // Les index des sommets représentant une face (S1, S2, S3).
    byte indexFaces[] = { 0, 4, 5, 0, 5, 1, 1, 5, 6, 1, 6, 2, 2, 6, 7, 2, 7, 3,
        3, 7, 4, 3, 4, 0, 4, 7, 6, 4, 6, 5, 3, 0, 1, 3, 1, 2 };

    public CubeTexture() {
        // Allocation mémoire des différents tampons (sommets, coordonnées
        // textures et index des faces).
        ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length * 4);
        vbb.order(ByteOrder.nativeOrder());
        mTamponSommets = vbb.asFloatBuffer();
        mTamponSommets.put(vertices);
        mTamponSommets.position(0);

      ByteBuffer tbb = ByteBuffer.allocateDirect(texCoords.length * 4);
       tbb.order(ByteOrder.nativeOrder());
       mTamponTexCoords = tbb.asFloatBuffer();
       mTamponTexCoords.put(texCoords);
       mTamponTexCoords.position(0);

        mTamponIndex = ByteBuffer.allocateDirect(indexFaces.length);
        mTamponIndex.put(indexFaces);
        mTamponIndex.position(0);
    }

    public void draw(GL10 gl) {
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, mTamponSommets);
        // Spécifie le tableau stockant les coordonnées de texture.
        gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, mTamponTexCoords);
        gl.glDrawElements(GL10.GL_TRIANGLES, 36, GL10.GL_UNSIGNED_BYTE,
            mTamponIndex);
    }
}
```

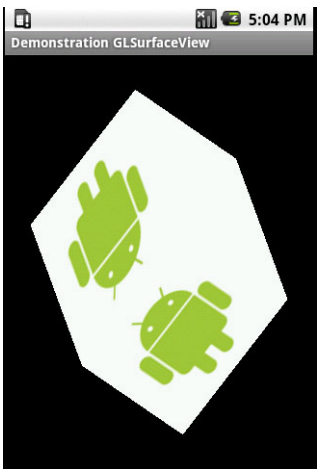


Figure 10–7
Le résultat de l'exemple précédent : un cube sur lequel nous avons appliqué une texture

Gérer les entrées utilisateur

Si vous souhaitez créer une application interactive telle qu'un jeu, vous devez gérer les entrées de l'utilisateur et réagir en accord avec celles-ci. Pour gérer les entrées, vous devez dériver la classe `GLSurfaceView` pour obtenir les événements des entrées.

Tableau 10–1 Les méthodes de gestion des entrées utilisateur

Nom méthode	Description
`onKeyDown/onKeyUp`	Appelée lorsque l'utilisateur appuie ou relâche un bouton du clavier.
`onKeyMultiple`	Appelée lorsque l'utilisateur appuie plusieurs fois sur une touche. Permet de récupérer le nombre de frappes sur la touche.
`onKeyPreIme`	Permet de gérer un événement avant qu'il soit consommé par une méthode d'entrée associée à la vue.
`onKeyShortcut`	Appelée quand l'événement d'un raccourci pas encore géré est lancé.
`onTrackballEvent`	Permet de gérer les événements liés à l'utilisation du trackball.
`onTouchEvent`	Gère les événements de toucher de l'utilisateur.

Vous pouvez aussi affecter des classes écouteur aux différentes méthodes à l'instar de n'importe quelle vue.

Voici un exemple de code gérant le toucher de l'écran par l'utilisateur. En fonction de la zone du toucher, la couleur de l'écran change.

Code 10-6 : Gestion des entrées utilisateur

```
public class DemoGLSurfaceView2 extends Activity {

   private GLSurfaceView mGLView;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      mGLView = new ClearGLSurfaceView(this);
      setContentView(mGLView);
   }

   @Override
   protected void onPause() {
      super.onPause();
      mGLView.onPause();
   }

   @Override
   protected void onResume() {
      super.onResume();
```

```
        mGLView.onResume();
    }
}

class ClearGLSurfaceView extends GLSurfaceView {

    private ClearRenderer mRenderer;

    public ClearGLSurfaceView(Context context) {
        super(context);
        mRenderer = new ClearRenderer();
        setRenderer(mRenderer);
    }

    @Override
    public boolean onTouchEvent(final MotionEvent event) {
        queueEvent(new Runnable() {
            public void run() {
                mRenderer.setColor(event.getX() / getWidth(), event.getY()
                    / getHeight(), 1.0f);
            }
        });
        return true;
    }
}

class ClearRenderer implements GLSurfaceView.Renderer {

    private float mRed = 1.0f;
    private float mGreen = 0.0f;
    private float mBlue = 0.0f;

    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
        // Ne rend rien.
    }

    public void onSurfaceChanged(GL10 gl, int w, int h) {
        gl.glViewport(0, 0, w, h);
    }

    public void onDrawFrame(GL10 gl) {
        gl.glClearColor(mRed, mGreen, mBlue, 1.0f);
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
    }

    public void setColor(float r, float g, float b) {
        mRed = r;
        mGreen = g;
        mBlue = b;
    }
}
```

Aller plus loin avec la surface de rendu

Personnaliser la configuration de la surface de rendu

Par défaut, la vue `GLSurfaceView` vous aide à choisir le type de surface sur laquelle vous souhaitez rendre votre scène. Chaque type d'appareil offre diverses surfaces de rendu pour lesquelles il ne sera pas toujours évident de trouver un réglage commun à tous les appareils. Cela signifie que vous allez devoir choisir la meilleure surface de rendu disponible sur chaque appareil. Par exemple, votre application nécessite peut-être un canal Alpha - pour faire apparaître votre surface en transparence au-dessus d'une autre couche GUI (*Graphical User Interface*) - ou un tampon de profondeur de 8 bits.

Pour pouvoir spécifier à la vue `GLSurfaceView` quelle configuration utiliser en fonction des paramètres ou d'un contexte applicatif particulier, vous pouvez redéfinir la ou les surcharges de la méthode `setEGLSurfaceChoose`. En fonction de la redéfinition de l'une des surcharges de la méthode `setEGLSurfaceChoose`, vous serez capable d'influencer le choix de la surface de rendu.

Tableau 10–2 Les méthodes de configuration de la surface de rendu

Signature de la méthode	Description
`setEGLConfigChooser(boolean needDepth)`	Choisit une configuration la plus proche possible de R5G6B5 avec ou sans tampon de profondeur 16 bits.
`setEGLConfigChooser(int redSize, int greenSize,int blueSize, int alphaSize,int depthSize, int stencilSize)`	Choisit la configuration optimale possédant le plus petit nombre de bits par pixel, au minimum les valeurs spécifiées, pour chaque canal de couleur.
`setEGLConfigChooser(EGLConfigChooser configChooser)`	Cette méthode vous laisse un contrôle total sur le choix de la configuration. Vous retournez votre propre configuration qui pourra analyser les capacités de l'appareil pour choisir la configuration la plus adéquate.

Voici un exemple d'application :

Code 10-7 : Différentes configurations possibles de la surface rendu

```
public int[] getConfigSpec() {
   if (besoinCanalAlpha) {
      // Tampon de couleur RGBA et tampon de profondeur de 16 bits.
      int[] configSpec = {
         EGL10.EGL_RED_SIZE, 8,
         EGL10.EGL_GREEN_SIZE, 8,
         EGL10.EGL_BLUE_SIZE, 8,
```

```
            EGL10.EGL_ALPHA_SIZE, 8,
            EGL10.EGL_DEPTH_SIZE, 16,
            EGL10.EGL_NONE
        };
        return configSpec;
    } else {
        // Nous spécifions uniquement la taille du tampon
        // de profondeur sans nous soucier du tampon de couleur.
        int[] configSpec = {
            EGL10.EGL_DEPTH_SIZE, 16,
            EGL10.EGL_NONE
        };
        return configSpec;
    }
}
```

EN COULISSES EGL (Native Platform Graphics Interface)

L'exemple du code 10-7 n'utilise pas les constantes de GL10 mais celles de EGL10. Cela est dû au fait que comme beaucoup de systèmes embarquant OpenGL ES, Android utilise EGL comme interface entre les API de rendu OpenGL ES et le système natif de fenêtres sous-jacent. EGL gère l'association entre les tampons et la surface, la synchronisation, l'accélération et les modes 2D et 3D utilisés par les API d'OpenGL ES.

Gérer la communication avec le thread de rendu

Tout comme pour la communication des interfaces utilisateur, vous devez utiliser un mécanisme de synchronisation pour communiquer entre le thread du GUI et le thread de rendu.

Vous pouvez utiliser n'importe quelle technique de communication entre threads : méthode synchronisée, etc. Néanmoins, la plate-forme Android 1.5 vous offre un mécanisme natif pour mettre en file des événements d'une façon simple et efficace grâce à la méthode `queueEvent`.

Effectuer un rendu à la demande

La plupart des applications 3D, par exemple les jeux, rendent une scène continuellement animée. Pour ces types d'applications, le comportement par défaut de la vue `GLSurfaceView` est de redessiner continuellement la scène, cela représente bien évidemment une consommation importante de ressources ainsi qu'une baisse importante des performances pour les autres tâches. Si vous développez une application ne nécessitant pas un rafraîchissement de la scène en permanence, mais uniquement sur

action de l'utilisateur par exemple, vous pouvez désactiver le rendu continu en appelant la méthode :

```
GLSurfaceView.setRenderMode(RENDERMODE_WHEN_DIRTY);
```

Si vous souhaitez retrouver le comportement par défaut de la surface de rendu et animer continuellement votre scène, vous pourrez appeler la méthode `requestRender`.

Déboguer une application OpenGL ES

Déboguer une application 3D n'est jamais chose aisée car elle comporte beaucoup de paramètres et une petite modification peut changer radicalement le rendu de la scène. La vue `GLSurfaceView` possède une méthode `setDebugFlags` permettant d'activer/désactiver le débogage et d'enregistrer les erreurs.

En appelant cette méthode avant la méthode `setRenderer`, vous spécifierez vos préférences sur la capacité d'OpenGL ES à vous informer des erreurs éventuelles et à afficher tous les appels :

```
mGLView = new GLSurfaceView(this);
mGLView.setDebugFlags(GLSurfaceView.DEBUG_CHECK_GL_ERROR
   | GLSurfaceView.DEBUG_LOG_GL_CALLS);
mGLView.setRenderer(new TriangleRenderer(this));
```

Vous pouvez consulter les logs dans la fenêtre de LogCat du module ADT.

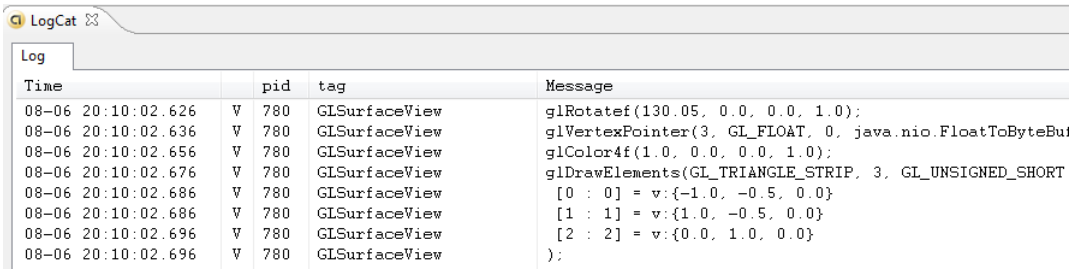


Figure 10–8 La fenêtre LogCat affichant les informations relatives aux appels

Les informations affichées représentent des données importantes sur l'activité de votre application. Néanmoins, les performances de votre application seront impactées et vous pourrez constater quelques ralentissements. N'oubliez pas de désactiver cette option avant de compiler votre application pour la distribuer.

En résumé

Aujourd'hui l'aspect graphique d'une application reste le principal critère de choix de l'utilisateur. Les capacités des appareils couplées aux possibilités offertes par les API de la plate-forme Android autorisent la création d'applications graphiquement riches et innovantes.

Cependant, sachez que tous les appareils ne sont pas équivalents et vous devrez systématiquement vous adapter aux performances et aux caractéristiques de l'appareil sur lequel s'exécute votre application. Certains appareils haut de gamme possèdent une accélération graphique très performante alors que d'autres d'entrée de gamme – souvent les plus nombreux – n'arrivent à afficher que quelques triangles et à gérer un seul éclairage, voire être complètement privés d'une quelconque accélération graphique. Lors du premier lancement, effectuez un rapide test des performances graphiques en proposant une petite cinématique à vos utilisateurs. Vous pourrez ainsi activer ou désactiver des effets graphiques et ajuster au mieux la surface de rendu pour la prochaine exécution de l'application.

En tant que développeur d'application 3D, vous êtes responsable des performances de votre application et vous devrez donc la tester physiquement sur un large panel d'appareils. Cela peut sembler long et fastidieux mais c'est un mal nécessaire qui sera récompensé par la qualité de votre application et la fidélité de vos utilisateurs.

Afin que ce chapitre ne s'éloigne pas trop de son sujet, bon nombre de points n'y ont pas été traités, par exemple la gestion de la lumière, du brouillard, des opérations de matrices, des opérations de tampons, etc. Vous pourrez trouver de plus amples informations sur leur utilisation sur le site Khronos Group (http://www.khronos.org/) ou dans la documentation du SDK.

Les possibilités d'OpenGL ES devraient augmenter avec le temps et l'apparition de nouvelles versions d'Android. Quoi qu'il en soit, cette version d'OpenGL ES permet déjà de réaliser un grand nombre d'applications de très haute qualité.

11

Les services et la gestion des threads

Tirer parti d'une application ne signifie pas toujours que celle-ci doit être visible par l'utilisateur. L'intérêt d'un service s'exécutant en arrière plan tient surtout du fait que contrairement à une application, celui-ci travaille en permanence. Néanmoins savoir notifier cette activité à l'utilisateur sans le déranger réside dans le subtil mélange que le développeur fera de l'utilisation des différentes possibilités de notifications qu'offre la plate-forme Android. La gestion de services en arrière-plan nécessite également de maîtriser la gestion des différents threads.

L'avantage majeur d'un appareil mobile comparé aux ordinateurs portables est sans conteste sa faible taille. Cette caractéristique permet à son propriétaire de l'emporter partout. Et aujourd'hui, le nombre d'applications disponibles et leur diversité permettent de combler idéalement le temps passé dans les transports, en pause, etc. Les applications agrégeant des informations en temps réel sont monnaie courante (résultats sportifs, info-flash, etc.), comme d'autres ne nécessitant aucune interface graphique pour fonctionner (c'est le cas du lecteur de musique lorsqu'il s'exécute en arrière-plan).

À la différence d'autres plates-formes du marché, Android offre un environnement pour ces applications sans interface utilisateur. Ainsi les *services* sont-ils des applications dérivant de la classe `android.app.Service`, qui s'exécutent en arrière-plan et

sont capables de signaler à l'utilisateur qu'un événement ou une information nouvelle requiert son attention.

Ce chapitre présente la façon de créer et de gérer de tels services, sans nuire à l'exécution d'autres applications, et montre les possibilités offertes aux développeurs pour notifier l'utilisateur.

Les services

Contrairement aux activités qui offrent une interface utilisateur, les services en sont complètement dépourvus. Ils peuvent cependant réagir à des événements, mettre à jour des fournisseurs de contenu, effectuer n'importe quel autre traitement..., et enfin notifier leur état à l'utilisateur via des toasts et des notifications (ces éléments seront décrits ci-après dans ce chapitre).

Ces services, complètement invisibles pour l'utilisateur, possèdent un cycle de vie similaire à une activité et peuvent être contrôlées depuis d'autres applications (activité, service et *Broadcast Receiver*). Il est ainsi commun de réaliser une application composée d'activités et d'un service, les activités n'étant là que pour permettre à l'utilisateur de contrôler le service ; un lecteur de musique utilisera une interface de sélection des musiques alors que le service jouera celles-ci en arrière plan.

Les applications qui s'exécutent fréquemment ou continuellement sans nécessiter constamment une interaction avec l'utilisateur peuvent être implémentées sous la forme d'un service. Les applications qui récupèrent des informations (informations, météo, résultats sportifs, etc.) via des flux RSS/Atom représentent de bons exemples de services.

Créer un service

Les services sont conçus pour s'exécuter en arrière plan et nécessitent de pouvoir être démarrés, contrôlés et stoppés par d'autres applications. Voyons d'abord la création d'un service, que nous serons ensuite capable de démarrer et d'arrêter. Nous traiterons l'aspect de sa gestion, via son association à une activité, plus loin dans ce chapitre.

Les services partagent un certain nombre de comportements avec les activités. C'est pourquoi vous retrouverez des méthodes similaires à celles abordées précédemment pour les activités. Pour créer un service, dérivez votre classe de `android.app.Service` puis, à l'instar d'une activité, redéfinissez les méthodes du cycle de vie avant de déclarer le service dans le fichier de configuration de l'application.

Pour créer un service Android, créez une nouvelle classe et dérivez celle-ci de la classe `android.app.Service` :

Code 11-1 : Implémenter un service

```
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

public class MonService extends Service {
   @Override
   public void onCreate() {
      // Placez ici le code qui sera exécuté lors de la création de ce service
   }

   Override
   public void onStart(Intent intent, int startId) {
      // Placez ici le code qui sera exécuté à chaque démarrage du service
   }

   @Override
   public void onDestroy() {
      // Placez ici le code qui sera exécuté lors de la destruction de ce service
   }

   @Override
   public IBinder onBind(Intent arg0) {
      return null;
   }
}
```

Les services peuvent redéfinir trois méthodes : `onCreate`, `onStart` et `onDestroy`. Les méthodes `onCreate` et `onDestroy` sont respectivement appelées lors de la création et de la destruction du service, à l'instar d'une activité.

La méthode `onBind` est la seule méthode que vous devez obligatoirement implémenter dans un service. Cette méthode retourne un objet `IBinder` qui permet au mécanisme IPC – communication interprocessus permettant d'appeler des méthodes distantes – de fonctionner. Dans un premier temps, laissez cette méthode vous retourner un objet `null`. Nous reviendrons plus tard sur celle-ci lorsque nous parlerons du langage de définition d'interface d'Android (AIDL).

Une fois votre service créé, il vous faudra, comme pour les activités d'une application, déclarer celui-ci dans le fichier de configuration de l'application. Ouvrez le fichier `AndroidManifest.xml` et ajoutez un élément `service` à l'intérieur de l'élément `application`.

Code 11-2 : Déclarer un service dans le fichier de configuration de l'application

```
<service android:name=".MonService">
```

Voici les paramètres supplémentaires que vous pouvez spécifier sous forme d'attributs dans l'élément `service` :

Nom de l'attribut	Description
`process`	Spécifie un nom de processus dans lequel le code sera exécuté.
`enabled`	Indique si ce service est actif ou non (peut-être instancié par le système).
`exported`	Booléen indiquant si le service est utilisable par d'autres applications.
`permission`	Spécifie une permission nécessaire pour exécuter ce service.

En plus des attributions dans l'élément `service`, vous pouvez utiliser l'élément `meta-data` pour spécifier des données supplémentaires et une section `intent-filter` à l'instar des activités.

Démarrer et arrêter un service

Les services peuvent être démarrés manuellement (via l'appel de la méthode `startService` ou un objet Intent) ou quand une activité essaie de se connecter au service via une communication interprocessus (IPC).

Pour démarrer un service manuellement, utilisez la méthode `startService`, laquelle accepte en paramètre un objet `Intent`. Ce dernier permet de démarrer un service en spécifiant son type de classe ou une action spécifiée par le filtre d'Intents du service.

Code 11-3 : Démarrer un service

```
// Démarre le service explicitement.
startService(new Intent(this, MonService.class));

// Démarre le service en utilisant une action (enregistré
// dans Intent Filter).
startService(new Intent(MonService.MON_ACTION_SPECIALE));
```

Le moyen le plus simple de démarrer un service est d'utiliser sa surcharge en utilisant le paramètre `class`. Cette option ne fonctionne que si le service est un composant de votre application et non un service tiers. Si le service à démarrer n'a pas été créé par vous, la seule alternative sera d'utiliser l'Intent associé à ce service.

La méthode `startService` peut être appelée plusieurs fois, ce qui aura pour conséquence d'appeler plusieurs fois la méthode `onStart` du service. Sachez donc gérer cette situation, de façon à ne pas allouer de ressources inutiles ou réaliser des traitements, qui rendraient caduque la bonne exécution de votre service.

Pour arrêter un service, utilisez la méthode `stopService` avec l'action que vous avez utilisée lors de l'appel à la méthode `startService` :

Code 11-4 : Arrêter un service

```
stopService(new Intent(this, monService.getClass()));
```

Contrôler un service depuis une activité

L'exécution d'un service requiert le plus souvent d'avoir pu configurer ce dernier, souvent à l'aide d'une interface que l'utilisateur pourra utiliser pour spécifier ou changer les paramètres.

Pour pouvoir communiquer avec un service, vous avez plusieurs possibilités à votre disposition. La première consiste à exploiter le plus possible le mécanisme des objets `Intent` au sein du service. La seconde (dans le cas de deux applications distinctes) consiste à effectuer un couplage fort entre les processus de l'application appelante et le processus du service en utilisant le langage AIDL (*Android Interface Definition Language*), qui permet la définition de l'interface du service au niveau du système d'Android – ce thème sera traité plus loin dans ce chapitre. Il est aussi possible de combiner ces deux possibilités en contrôlant un service à l'aide d'une activité et en les liant, à condition que le service et l'activité fassent partie de la même application. C'est cette dernière façon de faire que nous allons détailler ci-après.

Lorsqu'une activité est liée à un service, celle-ci possède une référence permanente vers le service, qui vous permet de communiquer avec le service comme si ce dernier n'était qu'une simple référence à un objet de votre activité.

Pour permettre à une activité de s'associer à un service, la méthode `onBind` doit être implémentée au niveau du service. Cette méthode retourne un objet de type `IBinder` qui sera utilisé plus tard par l'activité pour récupérer la référence vers le service.

La méthode `onBind` doit renvoyer un type de `IBinder` propre à votre service, ce qui signifie que vous devez créer votre propre implémentation de l'interface `IBinder` :

Code 11-5 : Implémentation de l'interface IBinder

```
private final IBinder mBinder = new MonServiceBinder();

@Override
public IBinder onBind(Intent intent) {
   return mBinder;
}

public class MonServiceBinder extends Binder {
   MonService getService() {
      return MonService.this;
   }
}
```

Une fois la méthode `onBind` implémentée au niveau de votre service, vous devez maintenant connecter l'activité au service. Pour ce faire, vous devez utiliser une connexion de type `ServiceConnection` : ce sont les méthodes de cette dernière qui seront appelées par l'activité pour se connecter ou se déconnecter du service.

Créez la connexion au niveau de l'activité, en redéfinissant les méthodes `onServiceConnected` et `onServiceDisconnected` de la classe `ServiceConnection`. La première méthode sera appelée lors de la connexion de l'activité au service et la seconde lors de la déconnexion.

Code 11-6 : Redéfinir les méthodes de la classe ServiceConnection

```
private MonService monService;

// L'objet ServiceConnection gère la connexion entre l'activité et le service.
private ServiceConnection maConnexion = new ServiceConnection() {

   // Méthode appelée lorsque la connexion est établie. Récupère la référence
   // vers le service associé.
   public void onServiceConnected(ComponentName className, IBinder service) {
      monService = ((MonService.MonServiceBinder)service).getService();
   }

   // Méthode appelée lorsque la connexion est rompue.
   public void onServiceDisconnected(ComponentName className) {
      monService = null;
   }
};
```

Une fois que la méthode `onBind` du service est implémentée et qu'un objet `ServiceConnection` est prêt à faire office de médiateur, l'activité doit encore demander de façon explicite l'association au service, à l'aide de la méthode `bindService`. Cette méthode prend trois paramètres :

- l'objet `Intent` du service que vous souhaitez invoquer, souvent de façon explicite directement en utilisant la `class` du service ;
- une instance de la connexion créée vers le service ;
- un drapeau spécifiant les options d'association, souvent 0 ou la constante `BIND_AUTO_CREATE`. Une autre valeur `BIND_DEBUG_UNBIND` permet d'afficher des informations de débogage. Vous pouvez combiner les deux valeurs à l'aide de l'opérateur binaire `&`.

Code 11-7 : Association du service à l'activité

```
public void onCreate(Bundle bundle) {
   super.onCreate(bundle);
```

```
    // Association du service
    Intent intentAssociation = new Intent(MonActivite.this, MonService.class);
    bindService(intentAssociation, mConnexion, Context.BIND_AUTO_CREATE);
}
```

Une fois l'association effectuée à l'aide de la méthode `bindService`, la méthode `onServiceConnected` sera appelée et la référence vers le service associé sera récupérée. Vous pourrez ainsi appeler toutes les méthodes et propriétés publiques du service au travers de cette référence comme s'il s'agissait d'un simple objet de l'activité, et de cette façon contrôler le service au travers de l'interface utilisateur proposée par l'activité.

Pour vous déconnecter, utilisez la méthode `unbindService` de l'instance de l'activité.

Dans ce qui suit, nous allons créer un service et y connecter une activité qui demande au premier d'afficher son statut en cours.

Le code source suivant illustre la partie service et notamment la redéfinition de la méthode `onBind`.

Code 11-8 : Implémenter un service pouvant s'associer à une activité

```
import android.app.Service;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;
import android.widget.Toast;

public class MonService extends Service {

   private final IBinder mBinder = new MonServiceBinder();

   @Override
   public IBinder onBind(Intent arg0) {
      return mBinder;
   }

   public void AfficheStatut() {
      Toast.makeText(MonService.this, "Voici le statut de mon service !",
         Toast.LENGTH_SHORT).show();
   }

   public class MonServiceBinder extends Binder {
      MonService getService() {
         return MonService.this;
      }
   }
}
```

Voici le code complet de l'activité à laquelle nous allons associer le service précédent, avec la connexion `ServiceConnection` et les appels d'association vers le service.

Code 11-9 : Implémenter une activité avec une connexion vers un service

```
import android.app.Activity;

public class GestionServiceActivite extends Activity {

   private MonService mMonService;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      // Associe le service à cette activité.
      Intent intentAssociation = new Intent(this, MonService.class);
      if (bindService(intentAssociation, mConnexion, Context.BIND_AUTO_CREATE)) {
         // Nous affectons une action du service à
         // un bouton de l'interface utilisateur.
         Button btnService = (Button) findViewById(R.id.BoutonService);
         btnService.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
               mMonService.AfficheStatut();
            }
         });
      }
   }

   @Override
   protected void onDestroy() {
      super.onDestroy();
      unbindService(mConnexion);
   }

   private ServiceConnection mConnexion = new ServiceConnection() {

      // Méthode appelée lorsque la connexion est établie. Récupère la
      // référence vers le service associé.
      public void onServiceConnected(ComponentName className, IBinder service) {
         mMonService = ((MonService.MonServiceBinder) service).getService();
      }

      // Méthode appelée lorsque la connexion est rompue.
      public void onServiceDisconnected(ComponentName className) {
         mMonService = null;
       }
   };
}
```

Voici l'interface utilisateur de l'activité associée :

Code 11-10 : Définition de l'interface de l'activité associée au code 11-9

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical" android:layout_width="fill_parent"
   android:layout_height="fill_parent">

<Button android:text="Appeler le service !" android:id="@+id/BoutonService"
   android:layout_width="fill_parent" android:layout_height="wrap_content">
</Button>
</LinearLayout>
```

Le langage AIDL pour appeler une méthode distante d'un autre processus

Sur la plate-forme Android, basée sur le noyau Linux, chaque application s'exécute dans son propre processus. Vous pouvez néanmoins vouloir communiquer avec un service s'exécutant dans une autre application et donc dans un processus autre que celui de l'interface utilisateur de votre activité.

Pour cela, vous devrez utiliser un mécanisme permettant de transformer les objets échangés en primitives élémentaires, c'est à dire de décomposer un objet dans le processus appelé pour le recomposer dans le processus appelant. C'est le travail qu'AIDL réalise pour vous. Les outils du SDK vont eux vous aider à créer le code de communication inter-processus car vous allez le constater, AIDL n'est pas d'une simplicité exemplaire. Si vous utilisez le module ADT d'Eclipse, les manipulations seront grandement simplifiées et c'est la démarche que nous avons choisie dans ce chapitre.

> **CULTURE Les langages de définition d'interface (IDL)**
>
> AIDL fait partie des langages IDL (*Interface Description Language*) qui aident les développeurs à générer du code permettant de réaliser les interfaces de communication inter-processus. Entre autre, le code généré contiendra une classe `Proxy` faisant le pont entre le client et l'implémentation pour l'échange des données. AIDL est un mécanisme similaire à CORBA ou COM, bien que plus léger et moins chaotique à mettre en place.

Voici les grandes lignes pour appeler une méthode distante :

1. Créer la définition de l'interface : vous définissez, dans un fichier à l'extension `.aidl`, l'interface composée des méthodes et des valeurs exposées.
2. Compiler votre interface : le SDK Android met à votre disposition un compilateur (dans le répertoire `tools/aidl.exe`) pour effectuer manuellement la tâche

(rien ne vous empêche d'intégrer ce dernier dans votre script Ant au besoin) ou alors ADT le fera pour vous. AIDL générera une interface Java adéquate.

3. Implémenter l'interface : vous devrez implémenter les méthodes de votre définition AIDL ainsi qu'implémenter les méthodes la classe interne `Stub` générée.
4. Exposer votre service aux autres : vous devrez exposer votre interface au travers de la méthode d'association `onBind` de votre service.

Création de la définition AIDL

La syntaxe de description d'une interface AIDL est relativement simple et proche de celle de Java. Le principe de l'interface AIDL consiste à déclarer dans un fichier d'extension `.aidl` une interface pourvue de méthodes, chacune d'elles composée d'arguments et d'une valeur de retour. Le type des données peut être de n'importe quel type (primitives ou classes personnalisées), pourvu que vous importiez dans votre fichier AIDL les types inconnus d'AIDL.

Voici la liste des types de données supportés par AIDL :

- tous les types primitifs de Java (`int`, `long`, `float`, etc.) ;
- les classes suivantes (sans nécessiter d'import de votre part) :
  - `String` et `CharSequence` ;
  - `List` : l'utilisation de la version générique de `List` est autorisée. Le type de réception sera toujours de type `ArrayList` ;
  - `Map` : l'utilisation de la version générique n'est pas autorisé. Le type de réception sera toujours de type `HashMap` ;
- les classes personnalisées : ces objets implémentant `Parcelable` (voir plus loin pour plus de détails) sont passés par valeur. Pour les utiliser dans votre définition AIDL, vous devrez les importer avec l'instruction `import` ;
- les interfaces AIDL : ces objets seront passés par référence. Pour les utiliser dans votre définition AIDL, vous devrez les importer avec l'instruction `import`.

Tous les types non primitifs nécessitent un qualificatif de direction pour spécifier dans quel sens la donnée est échangée : `in`, `out` ou `inout`. Les qualificatifs `out` et `inout` permettent au service de changer la valeur et de la renvoyer au client. Par défaut, si vous ne spécifiez aucun qualificatif, les types primitifs seront de type `in`. Pour des raisons de performances, limitez au maximum le retour des données avec `out` car la transformation, nécessaire pour l'échange entre les processus, est consommatrice de ressources mémoire et processeur.

Dans une interface AIDL, vous ne pouvez pas déclarer de gestion d'exception. Vous devrez par conséquent gérer toutes les exceptions dans votre méthode appelante et retourner la nature de l'erreur d'une autre façon (objet `out` ou code de retour). Votre

fichier AIDL comportant votre définition sera ensuite utilisé par l'outil `aidl.exe` pour générer l'interface Java nécessaire.

Créez une interface dans un fichier possédant l'extension `.aidl` portant le même nom que votre interface, par exemple :

Code 11-11 : Création du fichier .aidl de l'interface de votre service

```
package com.eyrolles.android.service.lecteurRSS;

interface ILecteurRSSService {

   List<String> getNomFilRSSAbonnement();

   void abonnementFilRss(in String adresseFluxRSS);
}
```

Implémenter l'interface

AIDL génère un fichier Java possédant le même nom que le fichier `.aidl` (à l'exception près que le fichier possède une extension Java). Si vous utilisez Eclipse, le module ADT fera l'opération pour vous et vous n'aurez pas à appeler `aidl.exe` à chaque changement de votre interface AIDL. Si vous n'utilisez pas Eclipse dans votre processus de développement, vous devrez appeler `aidl.exe` – qui se trouve dans le dossier `/tools` du SDK Android - à chaque modification de la définition de l'interface AIDL.

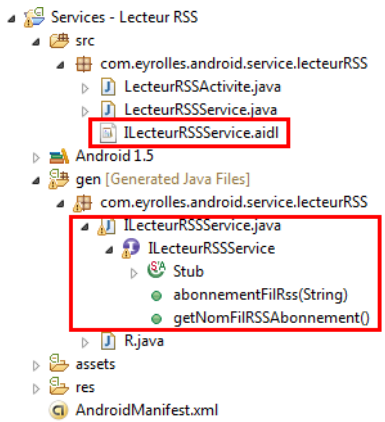


Figure 11–1
Génération automatique de l'interface Java à partir de la définition .aidl

L'interface ainsi générée possède une classe interne abstraite nommée `Stub` qui déclare toutes les méthodes de votre définition AIDL ainsi que quelques méthodes d'aide. Voici un extrait du code Java généré par AIDL :

Code 11-12 : Interface générée (code 12-11) par aidl.exe

```
public java.util.List<java.lang.String> getNomFilRSSAbonnement()
   throws android.os.RemoteException {
   android.os.Parcel _data = android.os.Parcel.obtain();
   android.os.Parcel _reply = android.os.Parcel.obtain();
   java.util.List<java.lang.String> _result;
   try {
      _data.writeInterfaceToken(DESCRIPTOR);
      mRemote.transact(Stub.TRANSACTION_getNomFilRSSAbonnement,
         _data, _reply, 0);
      _reply.readException();
      _result = _reply.createStringArrayList();
   } finally {
      _reply.recycle();
      _data.recycle();
   }
   return _result;
}

public void abonnementFilRss(java.lang.String adresseFluxRSS)
   throws android.os.RemoteException {
   android.os.Parcel _data = android.os.Parcel.obtain();
   android.os.Parcel _reply = android.os.Parcel.obtain();
   try {
      _data.writeInterfaceToken(DESCRIPTOR);
      _data.writeString(adresseFluxRSS);
      mRemote.transact(Stub.TRANSACTION_abonnementFilRss, _data,
         _reply, 0);
      _reply.readException();
   } finally {
      _reply.recycle();
      _data.recycle();
   }
}
```

Pour implémenter votre interface, créez une classe héritant de la classe abstraite `NomDeVotreInterface.Stub` générée par l'outil d'AIDL. Voici l'implémentation de la précédente définition de l'interface AIDL dans le fichier `IlecteurRSSService.aidl`.

Code 11-13 : Création de l'implémentation du Stub de l'interface du service

```
import java.util.List;
import android.os.RemoteException;

public class LecteurRSSServiceStubImpl extends ILecteurRSSService.Stub {
```

```
    public void abonnementFilRss(String adresseFluxRSS) throws RemoteException {
        // Code pour abonner l'utilisateur au flux RSS spécifié
        abonnerUtilisateurFluxRSS(adresseFluxRSS);
    }

    public List<String> getNomFilRSSAbonnement() throws RemoteException {
        // Code pour récupérer les flux auxquels l'utilisateur est abonné
        List<String> retFluxAbo = new List<String>();
        ContentResolver cr = LecteurRSSActivite.this.getContentResolver();
        ...
        return retFluxAbo;
    }
}
```

Rendre disponible votre interface au service

Une fois que vous avez défini l'implémentation de l'interface, il vous reste encore à l'exposer au client (une activité, etc.).

Dans un contexte de service, implémentez la méthode `onBind` de la classe `Service` pour retourner une instance de l'implémentation de l'interface.

Code 11-14 : Implémentant de la méthode onBind du service

```
@Override
public IBinder onBind(Intent intent) {
    // Retourne une instance de l'implémentation de l'interface AIDL.
    return mLecteurRSSServiceStubImpl;
}
```

Créer des classes personnalisées compatibles avec AIDL : l'interface Parcelable

Si vous le souhaitez, vous pouvez échanger entre plusieurs processus, des objets autres que ceux supportés par AIDL. Pour créer des objets « échangeables » par AIDL, vous devez vous assurer que la classe est définie du côté de l'appelé et que cet objet implémente l'interface `Parcelable`.

Le rôle d'un objet `Parcelable` est de transformer un objet complexe en un ensemble de valeurs primitives (`int`, `float`, `String`, etc.) de façon à pouvoir les échanger entre les processus.

Pour implémenter un objet `Parcelable`, vous devez suivre un certain nombre d'étapes :

1. Créez une classe implémentant l'interface `Parcelable.`
2. Implémentez la méthode `writeToParcel(Parcel out)` pour recréer l'état de votre objet dans un objet `Parcel.`
3. Ajoutez un attribut statique `CREATOR` implémentant l'interface `Parcelable.Creator.`

4 Créez un fichier avec l'extension `.aidl` portant le nom de votre classe et déclarez votre classe implémentant `Parcelable`.

Voici un exemple d'une classe `FluxRSS` qui implémente l'interface `Parcelable` :

Code 11-15 : Implémenter une classe *Parcelable* personnalisée pour AIDL

```
package com.eyrolles.android.service.lecteurRSS;

import android.os.Parcel;
import android.os.Parcelable;

public final class FluxRSS implements Parcelable {

   public String mAdresseFlux;

   public FluxRSS()
   { }

   public FluxRSS(String adresseFlux)
   {
      this.mAdresseFlux = adresseFlux;
   }

   private FluxRSS(Parcel in)
   {
      readFromParcel(in);
   }

   @Override
   public int describeContents() {
      return 0;
   }

   @Override
   public void writeToParcel(Parcel out, int flag) {
      out.writeString(mAdresseFlux);
   }

   private void readFromParcel(Parcel in) {
      this.mAdresseFlux = in.readString();
   }

   public static final Parcelable.Creator<FluxRSS> CREATOR =
             new Parcelable.Creator<FluxRSS>() {
       public FluxRSS createFromParcel(Parcel in) {
           return new FluxRSS(in);
```

```
        }

        public FluxRSS[] newArray(int size) {
            return new FluxRSS[size];
        }
    };
}
```

Voici le fichier `.aidl` de déclaration associé :

Code 11-16 : Déclaration dans un fichier AIDL de la classe FluxRSS (code 12-15)

```
package com.eyrolles.android.service.lecteurRSS;

parcelable FluxRSS;
```

En général, vous pouvez décomposer/recomposer un objet en primitives assez simplement. Néanmoins, si vous souhaitez décomposer/recomposer un objet possédant des références vers d'autres classes personnalisées, n'oubliez pas d'implémenter l'interface `Parcelable` dans chacune d'elles.

Appeler une méthode IPC d'AIDL

Maintenant que vous avez créé les implémentations nécessaires, la dernière étape consiste à appeler les méthodes exposées. La procédure est très similaire, à quelques détails près, à la partie précédente sur le contrôle d'un service par une activité.

Voici les éléments essentiels qu'une classe doit vérifier pour appeler une méthode distante :

1. Déclarer une variable du type de l'interface `votreInterface` déclarée dans le fichier `.aidl`.
2. Implémenter l'interface `ServiceConnection.`
3. Appeler la méthode `Context.bindService` en spécifiant votre implémentation de `ServiceConnection` : celle-ci vous retournera une instance de `Ibinder.`
4. Appeler la méthode `votreInterface.Stub.asInterface` en spécifiant l'objet `IBinder` récupéré grâce à la méthode `Context.bindService` afin d'avoir un objet du type de `votreInterface`.
5. Appeler les méthodes définies dans l'interface en prenant soin de gérer les exceptions de type `DeadObjectException` qui seront générées si la connexion est interrompue.

Enfin, pour vous déconnecter du service, utilisez la méthode `Context.unbindService` en spécifiant l'instance de votre interface.

Éléments à considérer avec AIDL

AIDL permet à une application de s'affranchir des limites des processus et de pouvoir échanger des objets de services créés par d'autres développeurs ou éditeurs d'application Android. Une possibilité essentielle pour la plate-forme Android mais pour laquelle vous devez garder à l'esprit quelques limitations :

- gestion des exceptions : les exceptions sont traitées au niveau du processus exécutant et celles-ci ne seront pas retournées à l'appelant ;
- les appels sont synchrones : ce qui signifie que l'appelant est en attente entre l'appel et la réponse. Si votre service est particulièrement gourmand en opérations et que cela dépasse un certain délai, l'utilisateur aura une application qui sera « gelée », voire se verra notifier par Android que l'application ne répond plus. Faites en sorte d'appeler vos méthodes distantes dans un thread séparé de façon à éviter ce genre de déconvenue ;
- déclarations : seules les méthodes peuvent être déclarées dans une définition AIDL ; vous ne pouvez pas déclarer d'attributs statiques par exemple.

Notifier l'utilisateur par le mécanisme des « toasts »

Notifier un utilisateur ne signifie pas le déranger alors qu'il utilise une autre application que la vôtre ou qu'il rédige un SMS. À cette fin, la plate-forme Android propose le concept de toast, ou message non modal s'affichant quelques secondes tout au plus. De cette façon, ni l'utilisateur ni l'application active à ce moment ne sont interrompus.

Les toasts représentent une excellente alternative au système de boîtes de dialogue. Cependant elle ne convient pas à tous les usages. En effet, un toast permet d'afficher une information pendant quelques secondes de façon à ce que l'utilisateur soit informé, ce qui convient très bien pour les applications s'exécutant en arrière-plan telles que les services. Mais si l'utilisateur détourne son regard quelques secondes de l'écran, l'information est définitivement perdue.

L'utilisation d'un toast est rendue aisée par les méthodes statiques de la classe `Toast` et notamment `makeText`. Il vous suffit de passer le `Context` de l'application, le texte et la durée d'affichage pour créer une instance de `Toast` que vous pourrez afficher avec la méthode `show` autant de fois que nécessaire.

Code 11-17 : Création d'un toast

```
// La chaîne représentant le message
String message = "Vous prendrez bien un toast ou deux ?";
// La durée d'affichage (LENGTH_SHORT ou LENGTH_LONG)
int duree= Toast.LENGTH_LONG;
```

```
// Création du Toast (le contexte est celui de l'activité ou du service)
Toast toast = Toast.makeText(this, message, duree);
// Affichage du Toast
toast.show();
```

La durée du toast ne peut être librement fixée : elle peut soit prendre la valeur LENGTH_SHORT (2 secondes) ou LENGTH_LONG (5 secondes).

L'extrait de code précédent affiche le résultat suivant :

Figure 11–2
Affichage d'un toast s'exécutant depuis un service : l'utilisateur est notifié sans entraver son utilisation.

Positionner un toast

Par défaut, Android affiche le message en bas de l'écran de l'utilisateur. Ce comportement peut être redéfini pour en changer la position. Vous pouvez spécifier la disposition d'un toast en spécifiant son ancrage sur l'écran ainsi qu'un décalage sur l'axe horizontal et vertical.

Code 11-18 : Positionner un toast

```
// Création du Toast
Toast toast = Toast.makeText(this, "Vous prendrez bien un toast ou deux ;",
Toast.LENGTH_LONG);
// Spécifie la dispostion du Toast sur l'écran
toast.setGravity(Gravity.TOP, 0, 40);
// Affichage du Toast
toast.show();
```

Le code précédent affiche le message en haut de l'écran avec un décalage vertical.

Figure 11–3
Modifiez la gravité du toast pour le positionner à un endroit approprié.

La position du toast est bien sûr fonction de l'importance du message. Affiché plus haut sur l'écran, le message aura plus de chance d'attirer l'attention de l'utilisateur que s'il se trouve tout en bas.

Personnaliser l'apparence d'un toast

Par défaut, Android utilise une fenêtre grisée pour afficher le texte du toast à l'utilisateur mais ce comportement n'est pas toujours le plus adapté pour communiquer des informations complexes à l'utilisateur. Dans certains cas, un affichage graphique sera plus approprié qu'un affichage textuel.

Sachez qu'Android est capable d'utiliser une vue personnalisée conçue pour l'occasion, en lieu et place du « carré grisâtre aux bords arrondis » par défaut. Pour spécifier cette vue à afficher, utilisez la méthode `setView` de l'objet `Toast`.

Même si vous ne souhaitez pas afficher autre chose que du texte, le simple fait de personnaliser le design de la vue vous permettra de différencier les messages émis par votre application des autres, et pour l'utilisateur de distinguer clairement l'origine des messages.

Notifier l'utilisateur : les notifications

Les notifications représentent un moyen plus standard et efficace de prévenir l'utilisateur d'un événement. Chaque notification est ajoutée comme une nouvelle entrée dans le menu de la barre de statut étendue, avec pour chacune une icône associée.

Les notifications permettent d'alerter l'utilisateur et de garder un historique des dernières notifications dont l'utilisateur n'a pas encore pris connaissance. Pour être plus efficace, une notification sera souvent combinée à un effet sonore : émission d'un son, utilisation du vibreur du téléphone, etc.

Les possibilités sont donc nombreuses mais l'objectif des notifications reste d'alerter l'utilisateur et non, comme le propose le système des toasts, de lui communiquer des informations. L'utilisation des notifications permet d'attirer l'attention d'un utilisateur ayant placé son téléphone à l'écart sur son bureau ou dans sa poche. C'est pourquoi le mécanisme des notifications est souvent le moyen privilégié utilisé par la plupart des services qui ne gèrent pas de l'information (messagerie, réseaux sociaux, etc.), pour alerter l'utilisateur d'un quelconque événement.

Les notifications sont gérées par un gestionnaire central de notifications de type `NotificationManager`. Les applications utilisent la barre de statut du système pour afficher les notifications. Cette barre est par défaut rétractée, mais l'utilisateur peut la dérouler pour afficher toutes les notifications :

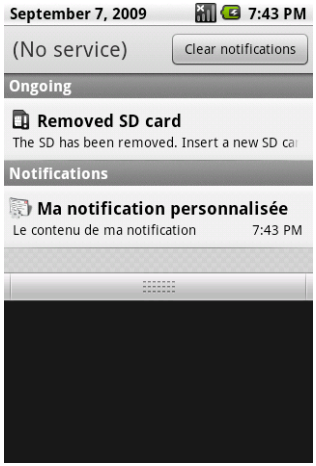


Figure 11–4
Le menu de statut étendu : vous pouvez ajouter une icône, un titre et une description pour chaque notification.

Le gestionnaire de notifications

Pour pouvoir utiliser le gestionnaire de notifications dans votre application, vous devez récupérer l'instance du service des notifications du système Android. Comme pour les autres services Android, vous récupérez le service en utilisant la méthode `getSystemService` avec le paramètre `Context.NOTIFICATION_SERVICE`.

Code 11-19 : Récupérer le gestionnaire de notifications

```
NotificationManager notificationManager =
(NotificationManager)getSystemService(Context.NOTIFICATION_SERVICE);
```

Une fois l'instance du gestionnaire de notifications récupérée, vous pouvez gérer toutes les notifications en termes d'ajout, de suppression et de modification.

Créer une notification

La création d'une notification se fait tout d'abord en créant une instance de type `Notification`. Une notification possède trois caractéristiques : une icône, un texte défilant et une heure (sous la forme d'un *timestamp*).

Code 11-20 : Création d'une notification

```
Notification notification = new
Notification(R.drawable.iconNotification, "Mon message défilant",
System.currentTimeMillis());
```

> **BONNE PRATIQUE Limiter l'utilisation de texte défilant**
>
> Dans la pratique l'utilisation d'une icône attire suffisamment l'attention de l'utilisateur, ne cédez pas à la tentation d'y ajouter inutilement le défilement du texte. De façon générale, mieux vaut changer l'icône uniquement et mettre le paramètre de défilement à `null`.

L'instanciation d'un objet `Notification` ne fait que créer la notification et lui affecter une icône et une heure : cette opération ne configure pas le contenu de la notification qui sera affichée à l'utilisateur. Chaque notification possède en effet un titre et un détail en plus des autres attributs. Pour configurer la notification avec ces informations, utilisez la méthode `setLatestEventInfo`.

Lorsque l'utilisateur visualisera la notification, il doit pouvoir revenir sur l'activité émettrice de la notification. Pour cela, vous devez utiliser un type spécifique d'`Intent`, le `PendingIntent`. Ce type d'Intent sera lancé en différé dès que l'utilisateur aura cliqué sur la notification, ce qui entraînera le retour au premier plan de l'activité.

Code 11-21 : Configuration d'une notification

```
// L'objet pendingIntent sera utilisé lorsque l'utilisateur cliquera
// sur la notification afin de rappeler l'activité émettrice au premier
// plan.
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0,
   new Intent(this, NotificationStatusBarActivite.class), 0);

// Le titre de la notification
String titreNotification = "Titre de ma notification";
// Le détail de la notification
String texteNotification = "Ma notification personnalisée";
```

```
// Spécifie les informations (titre et détail) de la notification
// qui sera affichée à l'utilisateur.
notification.setLatestEventInfo(this, titreNotification,
   textNotification, pendingIntent);
```

Pour envoyer la notification, vous devez appeler la méthode `notify` du `NotificationManager`. Cette méthode prend en argument un identifiant unique de référence, permettant d'identifier par la suite une notification, qui sera à nouveau utilisée par exemple pour annuler la notification.

Code 11-22 : Exécuter la notification

```
// Le nombre indiqué doit être unique, il sera également utilisé
// lors de l'annulation des notifications.
int idRefNotification = 2009;
...
// Envoie la notification.
notificationManager.notify(idRefNotification, notification);
```

Supprimer une notification

La suppression/annulation d'une notification entraîne la disparition de l'entrée dans le menu de la barre de statut ainsi que de l'icône associée.

Pour annuler une notification, appelez la méthode `cancel` de la notification en spécifiant l'identifiant de référence que vous avez communiqué à la méthode `notify`.

Code 11-23 : Annuler l'exécution d'une notification

```
mNotificationManager.cancel(idRefNotification);
```

Modifier et gérer les notifications

De façon générale, mettez toujours à jour les notifications existantes plutôt que d'en recréer systématiquement de nouvelles. Si vous créez de nouvelles notifications, cela ne rendra que plus confuse votre application.

Vous pouvez également utiliser une numérotation pour spécifier à l'utilisateur que votre application a plusieurs notifications en attente. Pour cela, utilisez la propriété `number` de la notification et incrémentez-la. De cette façon l'icône que vous avez spécifiée dans le constructeur de la notification sera complétée par un nombre précisant à l'utilisateur le nombre de messages en attente. Si vous spécifiez une valeur de 0, la valeur sera retirée de l'icône.

Code 11-24 : Spécifier le nombre de notification en attente

```
notification.number = 2;
```

Comme pour toute modification d'une notification, notifiez à nouveau le système avec la méthode `notify` du `NotificationManager`.

Enrichir les notifications : sonnerie et vibreur

Créer des notifications ou des toasts suffit rarement à attirer l'attention de l'utilisateur, il faut souvent y ajouter un effet sonore. Ces effets sont plus adaptés aux notifications, puisque l'objectif des toasts est justement de rester discret et de perturber l'utilisateur le moins possible dans son utilisation.

Il existe trois moyens de notifier un utilisateur avec un téléphone Android : émettre un son, utiliser le vibreur du téléphone ou encore gérer l'éclairage des LEDs. Ce chapitre ne traite pas de cette dernière possibilité car il y a pas ou peu d'implémentation sur les téléphones actuels (l'émulateur Android ne possède pas non plus cette fonctionnalité).

Émettre un son

C'est le moyen utilisé pour les appels entrants depuis des années, celui auquel les oreilles d'un utilisateur de portable sont préparées. Ce sera en général le moyen le plus efficace pour notifier l'utilisateur... si celui-ci n'a pas mis son téléphone en mode vibreur.

Android vous permet de jouer n'importe quel fichier audio de votre téléphone, spécifiez l'URI vers celui-ci à la propriété `sound`, pour lire ce son lors de l'émission de la notification via le `NotificationManager`.

Code 11-25 : Émettre un son lors de la notification

```
Uri uriSon = Uri.parse(getResources().getResourceName(R.raw.bart_hi_caramba));
notification.sound = uriSon;
```

Faire vibrer le téléphone

La vibration représente un bon compromis : elle n'émet pas de son ou très peu, peut se sentir dans la main ou dans une poche et, à l'extrême, faire se déplacer visuellement le téléphone sur votre bureau.

Android permet de déterminer la séquence de vibration comme vous l'entendez en spécifiant un délai de vibration et un délai de pause de façon successive. Par consé-

quent, en fonction de la séquence de vibrations que vous créez, vous pouvez véhiculer plusieurs degrés de notification à l'utilisateur. Si vous souhaitez définir 1 seconde de vibration, suivie d'une demi seconde de pause avant de reprendre par une seconde de vibration, vous allez spécifier la séquence (en millisecondes) : 1000, 500, 1000.

Au niveau du code, cette succession de nombres est représentée par un tableau d'entiers longs qui sera affecté à la propriété `vibrate` de la notification :

Code 11-26 : Ajouter une vibration à une notification

```
notification.vibrate = new long[] {1000,500,1000};
```

À vous de trouver la séquence qui sera au plus proche de ce que l'utilisateur doit interpréter lorsque les vibrations se feront ressentir.

Afin que les vibrations soient autorisées par l'application, vous devez l'autoriser dans le fichier de configuration de l'application en ajoutant la ligne de permission suivante :

Code 11-27 : Ajout de la permission pour autoriser l'utilisation du vibrateur de l'appareil

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

Les alarmes

Contrairement aux services qui s'exécutent en permanence en arrière-plan, l'objectif des alarmes est uniquement de lancer une ou plusieurs Intents à un moment précis. Une alarme ne fait donc pas réellement partie intégrante de l'application et celle-ci s'exécutera d'ailleurs en dehors de l'application.

Une alarme pourra très bien déclencher un objet `Intent` pour votre application, même après sa fermeture. En cela, les alarmes restent un concept à part, n'ayant que peu ou pas de lien avec l'application que vous développez (mais partie intégrante de la logique de votre application).

Les alarmes sont particulièrement intéressantes lorsqu'elles sont combinées avec des *Broadcast Receivers* puisque de cette façon vous n'avez pas besoin que votre application soit en cours d'exécution pour traiter l'intention de l'alarme. Les alarmes fonctionnent même lorsque l'appareil est en veille et permettent justement de le sortir de la veille.

Créer une alarme

Toutes les opérations liées à l'alarme s'effectuent via la classe `AlarmManager` qui n'est rien d'autre qu'un service du système que vous obtiendrez, comme tous les services du système, avec la méthode `getSystemService` :

Code 11-28 : Récupérer le gestionnaire d'alarmes

```
AlarmManager alarmManager =
(AlarmManager)getSystemService(ALARM_SERVICE);
```

Une fois le gestionnaire d'alarmes récupéré, utilisez la méthode `set` pour créer une nouvelle alarme. Cette méthode vous offre la possibilité de spécifier le type de la référence de temps pour laquelle vous donnez une valeur : s'agit-il d'une heure ou bien d'un délai depuis le démarrage de l'appareil ? La valeur que vous spécifiez, qui devra être en millisecondes, représente donc soit une durée soit une échéance en fonction de la référence de temps.

Tableau 11–1 Valeur de référence temporelle de l'alarme

Valeur	Définition
`RTC`	Émet l'objet `pendingIntent` à une heure particulière, mais sans réveiller l'appareil. Si l'appareil est en veille au moment de la diffusion de cet objet, alors ce dernier ne sera délivré qu'après le réveil de l'appareil.
`RTC_WAKEUP`	Identique au type RTC à la différence que l'appareil sera sorti de sa veille si celui-ci est dans cet état.
`ELAPSED_REALTIME`	L'objet `pendingIntent` sera émis sur la base du délai écoulé depuis le dernier démarrage de l'appareil. Si l'appareil est en veille, l'objet ne sera délivré qu'à sa sortie de veille.
`ELAPSED_REALTIME_WAKEUP`	Identique au type `ELAPSED_REALTIME` à la différence que l'appareil sera sorti de sa veille s'il se trouve dans cet état.

La création d'une alarme nécessite la création d'une intention différée – pour rappel, avec la classe `PendingIntent` – qui lui sera spécifiée en paramètre, associée au délai et au type de référence de l'alarme.

Code 11-29 : Création d'une alarme

```
// Récupération du gestionnaire d'alarme
AlarmManager alarmeManager = (AlarmManager)getSystemService(ALARM_SERVICE);
// Création de l'intention à émettre
Intent intentAlarme = new Intent(Intent.ACTION_DIAL);
PendingIntent intentAlarmeSuspend = PendingIntent.getBroadcast(this, 0,
intentAlarme, 0);
// Création d'une alarme à partir d'un délai ayant pour référence le dernier
démarrage de l'appareil.
alarmeManager.set(AlarmManager.ELAPSED_REALTIME_WAKEUP, 60L * 1000L * 2L,
intentAlarmeSuspend);
// Création d'une alarme utilisant une heure précise pour se lancer dans 2
heures après l'appel de cette méthode.
alarmeManager.set(AlarmManager.RTC_WAKEUP, System.currentTimeMillis() + (60 *
1000 * 2), intentAlarmeSuspend);
```

Quand l'alarme arrivera à son terme, l'Intent que vous avez spécifié sera émis. Si vous créez une alarme avec le même objet `Intent`, alors ce dernier remplacera le précédent.

Annuler une alarme

Si finalement l'utilisateur souhaite annuler l'alarme après sa création, cela reste tout à fait possible. Une alarme peut être annulée en utilisant la méthode `cancel`. Notez que les alarmes seront automatiquement supprimées après un redémarrage de l'appareil.

Code 11-30 : Annuler une alarme

```
alarmeManager.cancel(intentAlarmeSuspend);
```

Gestion des threads

Vous vous devez de livrer une application qui réagisse rapidement aux actions de l'utilisateur (un objectif inférieur à 1/4 de seconde est une bonne référence). Au-delà de 5 secondes d'attente, le gestionnaire d'activité `ActivityManager` peut prendre la décision de tuer votre service en le considérant comme une application sans réponse. Bien évidemment, votre application a sûrement du travail à effectuer en arrière-plan, mais au moins la règle est claire !

Il existe cependant des solutions permettant de laisser votre application et votre service actif pour longtemps :

- faire le travail dans un thread en arrière-plan ;
- faire faire le travail par un service d'arrière-plan, en se servant des notifications pour ramener votre utilisateur vers l'activité.

La plate-forme Android offre quelques raffinements pour pouvoir utiliser des threads d'arrière-plan et pouvoir échanger avec le thread de l'interface utilisateur. Ces mécanismes se trouvent implémentés au sein des classes `Runnable` et `Handler`.

Exécution et communication entre threads avec la classe Handler

La façon la plus simple de gérer une tâche en arrière-plan est de créer un thread en implémentant une classe dérivant de `Handler`. Vous pouvez utiliser une même instance de cette classe dans l'activité en cours. La classe `Handler` vous rendra la vie relativement facile puisqu'il suffit simplement de l'instancier pour quelle s'enregistre d'elle-même dans la gestion des threads du système Android.

Le thread ainsi créé, exécutant votre travail d'arrière-plan communique, avec l'instance de `Handler` du thread de l'interface utilisateur. Cela permet au thread de l'interface utilisateur, qui exécute le travail courant, de pouvoir mettre à jour l'interface en fonction de l'activité réalisée en arrière-plan. C'est un point essentiel qui vous servira, car si vous souhaitez mettre à jour l'interface utilisateur, vous devez impérativement utiliser le thread de l'interface utilisateur. Nous reviendrons sur ce point plus loin.

Pour communiquer avec l'instance de `Handler` vous pouvez utiliser les messages ou les objets `Runnable`. Voyons ensemble ces deux possibilités.

Utiliser les messages

Pour communiquer entre les threads d'un `Handler`, vous pouvez utiliser un système de messages. Ce système permet d'ajouter des messages dans une file et de les traiter dans leur ordre d'arrivée.

Pour envoyer un message à l'instance de `Handler`, vous devez d'abord récupérer une instance de `Message` en appelant la méthode `obtainMessage`. Il existe plusieurs surcharges de cette méthode, l'une vous retournant une instance vide d'un `Message` et une autre déjà complétée avec différents identifiants. Vous pouvez utiliser l'une ou l'autre des méthodes, cependant plus la tâche sera complexe plus vous aurez à spécifier de données de façon à ce que l'instance de `Handler` puisse traiter votre demande.

Une fois le message créé et configuré, vous allez devoir placer ce message dans la file de messages avec les méthodes `sendMessage`, `sendMessageAtFrontOfQueue`, `sendMessageDelayed` et `sendMessageAtTime`.

Tableau 11–2 Méthodes pour placer un message dans la file de messages

Méthode	Utilisation
`sendMessage(Message)`	Place le message à la fin de la file des messages.
`sendMessageAtFrontOfQueue(Message)`	Place le message sur le dessus de la file, c'est à dire que votre message est placé devant tous les autres, le rendant prioritaire dans son traitement.
`sendMessageDelayed(Message,long)`	Place le message en fin de file après une attente spécifiée en millisecondes.
`SendMessageAtTime(Message,long)`	Place le message en fin de file au moment spécifié en millisecondes en se basant sur l'*uptime* du système (utilisez la méthode `SystemClock.uptimeMillis` pour récupérer la valeur du système).

Il existe aussi des variantes de ces méthodes pour envoyer des messages vides : il s'agit des méthodes `sendEmptyMessage`, `sendEmptyMessageDelayed` et

`sendEmptyMessageAtTime` qui possèdent le même comportement que leurs grandes sœurs décrites dans le tableau. La seule différence est que vous n'avez pas à fournir d'objet `Message` mais un entier comme seule donnée de communication.

Une fois le message envoyé et pour chaque message dans la file, c'est la méthode `handleMessage` de votre implémentation de `Handler` qui sera appelée pour leur traitement. De cette façon là, vous pourrez mettre à jour votre interface utilisateur directement dans le handler. Cependant, sachez que pendant que vous mettez à jour votre interface utilisateur, les autres threads destinés à l'interface utilisateur seront interrompus.

Un exemple typique de l'utilisation de la classe `Handler` pour gérer les threads est l'intégration d'une barre de progression dans l'interface.

Voici le fichier de la mise en page de l'activité :

Code 11-31 : Interface de l'activité contenant une barre de progression

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical" android:layout_width="fill_parent"
   android:layout_height="fill_parent">
   <TextView android:text="Barre de progression avec Handler :"
             android:id="@+id/text_barre"
      android:layout_width="wrap_content" android:layout_height="wrap_content"
      android:gravity="center_horizontal" />

   <ProgressBar android:id="@+id/barre_progression"
      style="?android:attr/progressBarStyleHorizontal"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content" />
</LinearLayout>
```

Voici le code associé utilisant `Handler` pour communiquer entre le thread d'arrière-plan et celui de l'interface utilisateur :

Code 11-32 : Activité utilisant un Handler pour gérer une barre de progression

```
import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.widget.ProgressBar;

public class HandlerActivite extends Activity {

   // La barre de progression à mettre à jour
   ProgressBar barreProgression;
```

```
    // Comme pour les threads classiques, nous gardons la trace de son activité.
    boolean enExecution = false;

    // Notre sous-classe Handler qui gère le traitement des messages.
    final Handler handler = new Handler() {
       @Override
       public void handleMessage(Message msg) {
          barreProgression.incrementProgressBy(10);
       }
    };

    @Override
    public void onCreate(Bundle icicle) {
       super.onCreate(icicle);
       setContentView(R.layout.main);
       barreProgression = (ProgressBar) findViewById(R.id.barre_progression);
    }

    public void onStart() {
       super.onStart();
       barreProgression.setProgress(0); // Réinitialise la barre de progression.
       // Création d'un thread d'arrière-plan qui envoie un message au handler
       // toutes les secondes.
       Thread threadArrierePlan = new Thread(new Runnable() {
          public void run() {
             try {
                for (int i = 0; enExecution && i < 10; ++i) {
                   // Mise en pause d'une seconde
                   Thread.sleep(1000);
                   // Création d'un message vide
                   Message msg = handler.obtainMessage();
                   // Envoi du message au handler
                   handler.sendMessage(msg);
                }
             } catch (Throwable t) { }
          }
       });
       enExecution = true;
       // Lancement du thread d'arrière-plan
       threadArrierePlan.start();
    }

    public void onStop() {
       super.onStop();
       enExecution = false;
    }
}
```

Nous créons notre propre implémentation de la méthode `handleMessage` de l'instance de `Handler` directement dans l'activité. Dans cette implémentation, chaque fois qu'un message est traité, nous demandons à la barre de progression d'avancer de 10 %.

La méthode `onStart` crée un thread d'arrière-plan afin d'envoyer des messages toutes les secondes qui seront traités par le thread du handler. Cela a pour résultat de faire progresser la barre de progression de 10 % toutes les secondes.

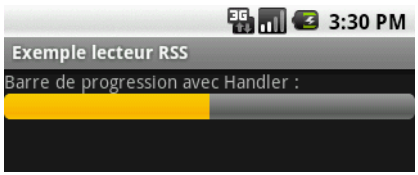


Figure 11–5
Barre de progression animée par l'envoi de messages à un Handler

Utiliser Runnable

En lieu et place de l'utilisation des messages, vous pouvez spécifier un objet `Runnable` qui exécutera sa méthode `run`. Pour cela, utilisez les méthodes `post`, `postAtFrontOfQueue`, `postDelayed` et `postAtTime` de la classe `Handler` pour passer un objet `Runnable`.

Savoir gérer les threads

Bien que la classe `Handler` facilite la vie des développeurs Android, sachez que l'utilisation de threads dans une application apporte une certaine complexité à cette dernière. Que ce soit au niveau de la logique de l'application, du débogage ou de la maintenance, cette complexité n'est pas le seul impact, puisque les threads ont aussi une influence sur la mémoire, la puissance de calcul utilisée et donc sur la durée de vie de la batterie.

Lorsque vous choisissez d'utiliser des threads pour diviser vos traitements, prenez ces remarques en compte. D'autant qu'avec l'utilisation des threads apparaissent vite les écueils de la concurrence d'accès aux objets partagés. Bien que la gestion complexe des threads ne soit pas traitée dans ce livre, certains problèmes peuvent néanmoins être résolus élégamment grâce à l'utilisation des classes situées dans le paquetage `java.util.concurrent` qui est apparu avec Java 5.

Lorsque vous exécutez un thread, vous ne savez pas toujours dans quel contexte vous vous situez : êtes-vous dans un thread d'arrière-plan ou dans le thread de l'interface utilisateur ? Pour le savoir, utilisez la méthode `runOnUiThread` de la classe `Activity`.

En résumé

Ce chapitre a été l'occasion d'expliquer la gestion des tâches en arrière-plan – tant des services que des threads gérés par `Handler`. Nous avons aussi vu comment communiquer leurs activités à l'utilisateur à l'aide de notifications ou d'alarmes. Les notifications peuvent prendre la forme d'un toast, d'une notification dans la barre de statut, d'une vibration du téléphone, etc. Chaque forme de notification possède des avantages et des inconvénients qu'il est bon d'avoir en tête au regard de l'effet recherché.

Nous avons aussi vu comment gérer la problématique de la communication entre un service et une activité situés dans deux applications différentes à l'aide d'AIDL.

L'utilisation de service d'arrière-plan est le propre de tout système d'exploitation moderne. Néanmoins dans le monde de la mobilité, ceux-ci peuvent représenter le talon d'Achille du système si certains développeurs n'y prennent pas garde. À vous de trouver le meilleur compromis entre fonctionnalité, réactivité et ressources consommées.

12

Téléphonie

Bien qu'Android ne soit pas uniquement destiné aux téléphones, la téléphonie reste un composant essentiel de la plate-forme. Il y a encore beaucoup d'applications à inventer ou à perfectionner pour améliorer le quotidien de l'utilisateur, comme celles qui règlent le volume de la sonnerie ou la disponibilité pour la prise d'appels en fonction de la position géographique de l'utilisateur entre sa maison et son lieu de travail !

Votre téléphone, ou votre appareil qui intègre des fonctions de téléphonie, vous offre déjà des applications qui gèrent les appels et les SMS. Pour autant, toutes les applications Android sont égales. Il est ainsi aisé de remplacer ces applications ou d'en ajouter. Par exemple, on peut concevoir un système qui filtre les appels en fonction du numéro ou de la personne qui appelle. On peut aussi créer un répondeur automatique par SMS.

Dans un premier temps, nous allons nous familiariser avec les moyens de simulation pour tester les applications : comment simuler un appel ou un SMS et comment placer l'émulateur dans différents états (en cours d'appel, etc.). Puis nous aborderons la collecte d'informations, l'émission d'appels et finalement l'envoi et la réception de SMS.

Utilisation du kit de développement pour la simulation

Commençons par présenter l'utilisation de l'émulateur. Ce dernier permet de manipuler toutes les fonctionnalités que nous allons utiliser, et ce, sans frais. En effet, au-delà du fait que tous les développeurs ne possèdent pas forcément un téléphone

Android pour tester leur application, les fonctions liées aux appels et aux SMS se distinguent par leur coût. Un émulateur permettra donc de réaliser des appels et des envois de SMS factices sans utiliser votre forfait !

Affichage de l'interface graphique

La simulation des fonctions de téléphonie est rendue possible par l'outil DDMS (*Dalvik Debug Monitor Service*).

> **OUTIL Rappel sur DDMS**
>
> L'environnement de développement Android est distribué avec un outil de débogage appelé *Dalvik Debug Monitor Service* (DDMS) qui met à votre disposition une multitude de fonctions comme la capture d'écran ou la surveillance des tâches. Ce chapitre nous donne l'occasion d'utiliser ses outils de gestion téléphonique comme la simulation d'appels.

DDMS peut être utilisé de deux manières. La première consiste à utiliser l'interface graphique complète lancée en ligne de commande par la commande `ddms` (si le répertoire `tools` du SDK Android est ajouté à la variable d'environnement `path`). L'onglet qui nous intéresse est *Emulator Control*.

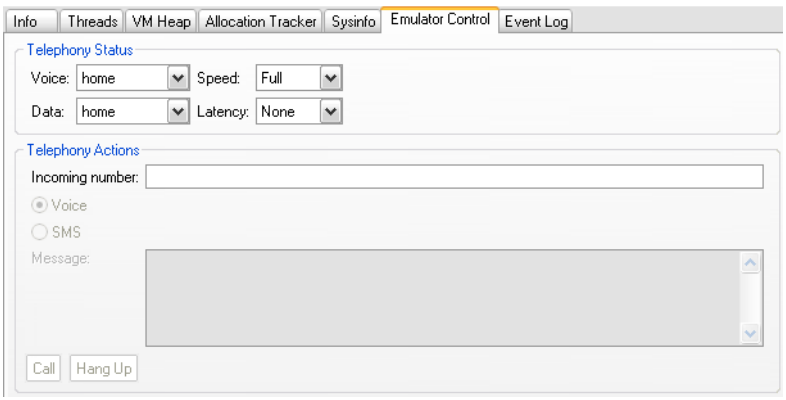


Figure 12–1
Aperçu des onglets de l'outil DDMS lancé depuis la console

La seconde consiste à utiliser le module ADT d'Eclipse. Ce dernier nous permet d'ajouter une vue à Eclipse, qui est une vue frontale pour DDMS. Pour cela, il suffit de se rendre dans le menu *Window>Show view* et de sélectionner la vue *Emulator Control* rangée dans la rubrique *Android*.

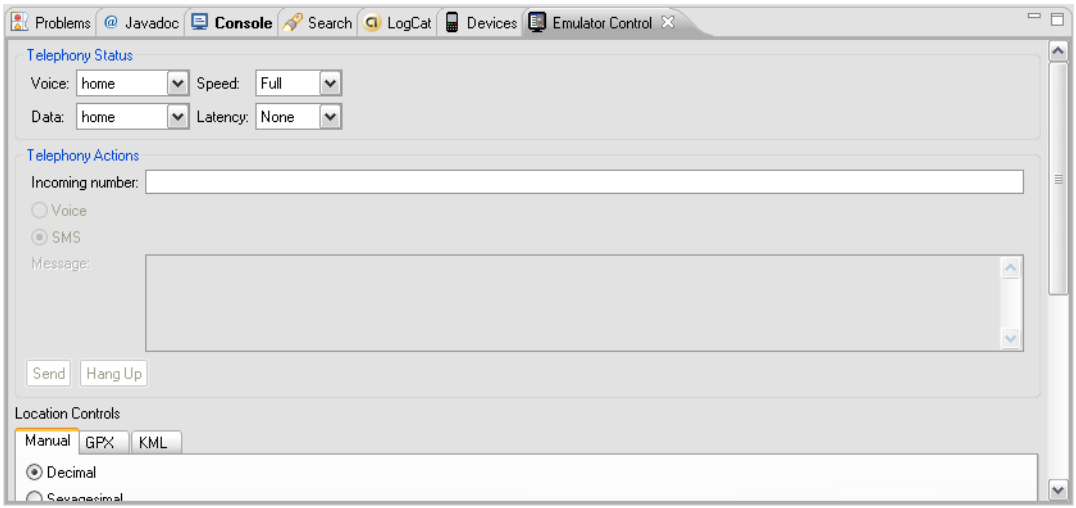


Figure 12–2 Vue intégrée à Eclipse

Une fois le choix arrêté, les deux interfaces sont en tous points semblables, tout comme leur utilisation.

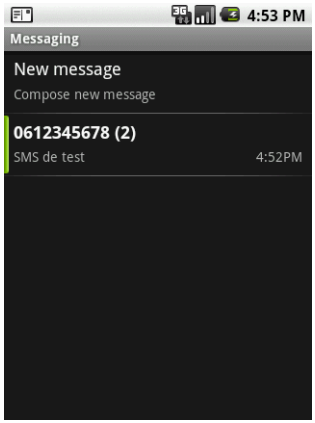
Simulation de l'envoi d'un SMS

Le champ *Incoming number* permet de saisir le numéro de téléphone qui sera l'émetteur du SMS. En cochant le bouton radio *SMS*, vous activez le champ *Message* qui contient le texte du SMS à envoyer.

Figure 12–3 La vue Eclipse contenant le message à envoyer

Le SMS saisi est reçu directement par l'émulateur lancé.

Figure 12–4
L'émulateur reçoit le SMS



Simulation d'un appel

La simulation d'un appel fonctionne de façon analogue. Le bouton radio *Voice* permet de sélectionner une action voix et le bouton *Call* permet de déclencher l'appel.

Figure 12–5 La vue Eclipse avec les propriétés de l'appareil

Figure 12–6
Réception de l'appel sur l'émulateur

Gestion de l'état de la fonction téléphone

Il est possible de changer l'état de la fonction téléphone beaucoup plus finement en se connectant via le protocole Telnet à l'émulateur en utilisant l'outil du même nom.

> **CULTURE Telnet, protocole de commande distante**
>
> Telnet est un protocole datant du début des années 1970 qui permet d'exécuter des commandes sur une machine distante en les saisissant au clavier depuis un autre ordinateur. L'outil `telnet` utilisé ici est une implémentation client de ce protocole qui nous permet de nous connecter à une machine distante : l'émulateur. Cet outil existe sur différentes plates-formes, Windows (pas toujours installé ou activé par défaut, notamment à partir de Windows Vista) ou Linux.
> La commande pour initier la connexion est la suivante :
> `telnet nom-du-serveur`
> Le nom du serveur est une adresse IP ou le nom d'une machine à laquelle on veut se connecter. La commande peut être complétée par un numéro de port, sachant que le numéro de port normalisé pour le protocole Telnet est 23. On obtient alors une commande de la forme :
> `telnet nom-du-serveur port`
> Rappelons qu'en simplifiant, le numéro de port permet de déterminer pour une machine donnée le protocole concerné par une connexion. Il sert en quelque sorte de multiplexage. Par exemple, sur une même machine, on peut avoir un serveur FTP qui attendra des connexions sur le port 21 et en même temps un serveur web qui répondra quant à lui aux requêtes des navigateurs sur le port 80 par défaut.

Code 12-1 : Deux commandes équivalentes pour utiliser l'outil telnet

```
telnet localhost 5554

telnet 127.0.0.1 5554
```

Le numéro de port à utiliser est affiché dans la barre de titre de votre émulateur. On peut utiliser plusieurs émulateurs en même temps et les différencier pour se connecter précisément à l'un d'entre eux grâce à cette notion de port.

Figure 12–7
Numéro de port lié à l'émulateur

Une fois connecté, nous allons nous préoccuper uniquement des commandes concernant la téléphonie, classées sous l'appellation GSM (le *Global System for Mobile Communications* est un standard de téléphonie mobile mais GSM désigne par extension un téléphone mobile).

Les différentes possibilités de commandes sont obtenues en utilisant la commande `help gsm`.

Tableau 12–1 Liste des commandes concernant la téléphonie avec leur rôle

Commande	Effet
`gsm list`	Liste des appels en cours.
`gsm call`	Crée un appel entrant vers l'émulateur. Exemple : `gsm call 0612345678`.
`gsm busy`	Ferme un appel sortant car occupé. Exemple : `gsm busy 123`.
`gsm hold`	Met un appel sortant (à partir du émulateur) en pause.
`gsm accept`	Change l'état d'un appel sortant à actif.
`gsm cancel`	Interrompt un appel (entrant ou sortant). Exemple : `gsm cancel 0612345678`.
`gsm data`	Modifie l'état de connexion données.
`gsm voice`	Modifie l'état de connexion voix.
`gsm status`	Affiche l'état voix et données du simulateur.

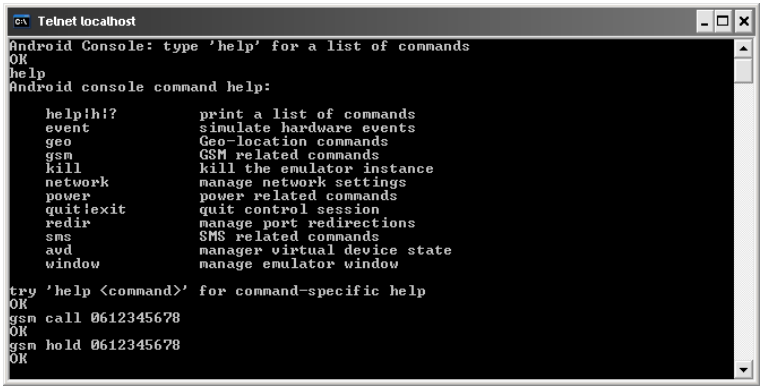


Figure 12–8
Un appel, puis une mise en pause seront obtenus grâce à la commande gsm.

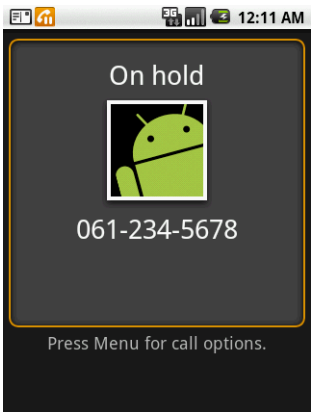


Figure 12–9
L'émulateur répond immédiatement.

Informations sur l'état de l'appareil

La classe centrale pour la récupération d'informations concernant la téléphonie est `TelephonyManager` qui se trouve dans le paquetage `android.telephony`. Cette classe va nous permettre à la fois de récupérer des informations sur l'appareil et la carte SIM, mais également d'obtenir, voire de se tenir informer du statut et des changements d'états de la fonction téléphone (en conversation, appel en pause, etc.).

Pour pouvoir lire des informations concernant le téléphone avec la classe `TelephonyManager`, la permission `READ_PHONE_STATE` est à ajouter au fichier de configuration de l'application :

Code 12-2 : Permission de lecture des informations de téléphonie

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

Informations sur l'appareil et le réseau

Toutes les informations sont collectées en utilisant la classe `TelephonyManager` et en invoquant les diverses méthodes qui sont disponibles.

Code 12-3 : Utilisation de la classe TelephonyManager pour collecter des informations

```
TelephonyManager telephonyManager =
    (TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);

// Infos réseau.
String nomOperateurReseau = telephonyManager.getNetworkOperatorName();
Log.d("Nom de l'operateur réseau ", nomOperateurReseau);

// Récupération de l'état de la fonction téléphone
int etat = telephonyManager.getCallState();
switch(etat) {
  // Pas d'activité
  case TelephonyManager.CALL_STATE_IDLE:
    Log.d("Etat", "Inactif");
    break;
  // Appel venant d'arriver
  case TelephonyManager.CALL_STATE_RINGING:
    Log.d("Etat", "Sonne");
    break;
```

```
  // Un appel en cours de composition, actif ou en pause
  case TelephonyManager.CALL_STATE_OFFHOOK:
    Log.d("Etat", "Appel en cours");
    break;
  default:
    Log.d("Etat", "Autre : " + etat);
  break;
}

// Informations sur l'appareil.
String idAppareil = telephonyManager.getDeviceId();
Log.d("Identifiant de l'appareil", idAppareil);

String versionSoftwareAppareil =
    telephonyManager.getDeviceSoftwareVersion();
if (versionSoftwareAppareil != null)
    Log.d("Version logicielle", versionSoftwareAppareil);

// Informations sur la carte SIM.
String numeroSerieSIM = telephonyManager.getSimSerialNumber();
Log.d("SIM", numeroSerieSIM);
```

> **RAPPEL À propos des résultats obtenus**
>
> Il faut garder en tête que les données obtenues grâce à l'émulateur sont proches de la réalité mais ne sont pas opérationnelles. Aussi, veillez à ne pas baser vos raisonnements uniquement sur ces résultats – un test sur un téléphone réel sera toujours nécessaire.

Par ailleurs, on peut récupérer d'autres informations, c'est la démarche globale qui est présentée ici (de plus, la récupération des informations sur les cellules, par exemple, dépend d'une autre permission).

Être notifié des changements d'états

Ce qui nous intéresse le plus souvent, c'est d'être tenu au courant des évolutions des différentes valeurs obtenues précédemment. Pour cela, il faut manipuler la classe `PhoneStateListener` afin de préparer un écouteur et l'enregistrer auprès du manager. Faisons-le pour être tenu au courant des changements de l'état de la fonction téléphone :

Code 12-4 : Ajouter un écouteur de changements d'états

```
TelephonyManager telephonyManager =
    (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);
telephonyManager.listen(new EcouteurEtatTelephone(),
    PhoneStateListener.LISTEN_CALL_STATE);
```

L'indication LISTEN_CALL_STATE laisse tout de suite imaginer qu'on peut créer des écouteurs pour d'autres types d'informations, et effectivement c'est le cas. On peut, entre autres, obtenir des renseignements sur la connexion donnée avec la constante LISTEN_DATA_CONNECTION_STATE, sur la force du signal avec LISTEN_SIGNAL_STRENGTH. Pour aller plus loin, jetez un coup d'œil à la documentation de la classe PhoneStateListener.

Code 12-5 : Écouteur de changements d'états de la fonction téléphone

```
package com.eyrolles.android.telephonie;

import android.telephony.PhoneStateListener;
import android.telephony.TelephonyManager;
import android.util.Log;

public class EcouteurEtatTelephone extends PhoneStateListener { (1)

    private static final String LOG_ETAT_TELEPHONE =
"EcouteurEtatTelephone";

    public void onCallStateChanged(int etat, String numero) { (2)
        super.onCallStateChanged(etat, numero);
        switch(etat) { (3)
        // Pas d'activité.
        case TelephonyManager.CALL_STATE_IDLE:
            Log.d(LOG_ETAT_TELEPHONE, "Inactif");
            break;
        // Appel venant d'arriver.
        case TelephonyManager.CALL_STATE_RINGING:
            Log.d(LOG_ETAT_TELEPHONE, "Sonne (" + numero + ")");
            break;
        // Appel en cours, actif ou en pause.
        case TelephonyManager.CALL_STATE_OFFHOOK:
           Log.d(LOG_ETAT_TELEPHONE, "Appel en cours (" + numero + ")");
            break;
        default:
            Log.d(LOG_ETAT_TELEPHONE, "Autre : " + etat +
                " (" + numero + ")");
        break;
        }
            // Vous pouvez décider d'une action à effectuer
            // Par exemple : racrocher

    }
}
```

On étend la classe `PhoneStateListener` (1), ce qui nous amène à donner corps à la méthode `onCallStateChanged` (2). C'est dans cette dernière que l'on traite au cas par cas (3) les différents états possibles.

Passer des appels

Pour passer des appels, on peut envisager deux approches différentes :

- préremplir le numéro de téléphone dans le composeur de numéro ;
- préremplir le numéro et déclencher l'appel.

Quelle que soit la démarche choisie, on utilisera les actions adéquates.

Préremplir le numéro à composer

Il faut employer l'objet `Intent` en spécifiant l'action `ACTION_DIAL`. L'URI doit être de la forme `tel:numero`.

Le code est le suivant :

Code 12-6 : Démarrage de l'application de composition de numéro de téléphone

```
String telURI = "tel:" + "0612345678";
Intent intent = new Intent(Intent.ACTION_DIAL, Uri.parse(telURI));
startActivity(intent);
```

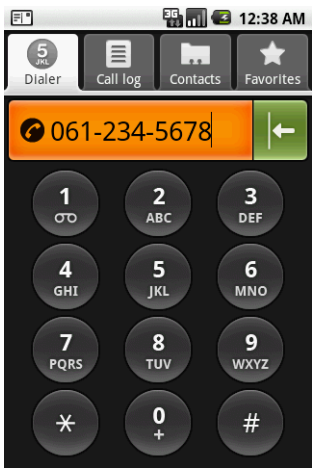


Figure 12–10
Résultat du préremplissage du numéro

Déclencher l'appel

Il faut là encore utiliser un objet `Intent` mais avec l'action `ACTION_CALL`. L'URI a la même forme, à savoir `tel:numero`.

Code 12-7 : Démarrage de l'application d'appels téléphoniques

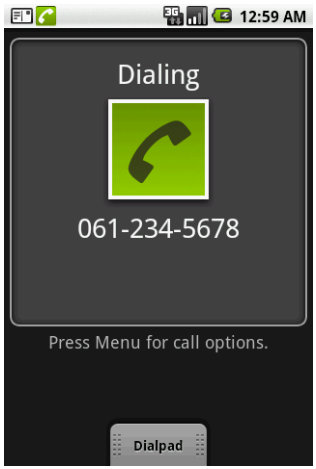
```
String telURI = "tel:" + "0612345678";
Intent intent = new Intent(Intent.ACTION_CALL, Uri.parse(telURI));
startActivity(intent);
```

Il faut également penser à ajouter à la configuration de l'application la permission pour passer des appels :

Code 12-8 : Ajout de la permission nécessaire pour appeler un correspondant

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

Figure 12–11
Émission d'un appel



Gérer les SMS

Plus les années passent, plus le nombre de SMS échangés augmente. Ce service de messages courts (160 caractères), très prisé du jeune public, est un incontournable si votre application doit envoyer ou gérer la réception de certains SMS.

Envoi de SMS

Pour envoyer des SMS, il faut tout d'abord ajouter la permission adéquate au fichier de configuration de l'application. Il s'agit de la permission `SEND_SMS`.

Code 12-9 : Ajout de la permission pour envoyer des SMS

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

Code 12-10 : Application d'envoi de SMS

```
package com.eyrolles.android.telephonie;

import android.app.Activity;
import android.os.Bundle;
import android.telephony.gsm.SmsManager;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class EnvoiSMS extends Activity {

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        Button sendBtn = (Button) findViewById(R.id.envoiBouton); (1)
        sendBtn.setOnClickListener(new OnClickListener() {
            public void onClick(View view) {
                envoiSMS();
            }
        });
    }

    private void envoiSMS() {

        // Récupération des champs texte pour le numéro et le message.
        EditText numeroEditText =
            (EditText)findViewById(R.id.numeroEditText);
        EditText messageEditText =
            (EditText) findViewById(R.id.messageEditText);
        String numero = numeroEditText.getText().toString();
        String message = messageEditText.getText().toString();
```

```
        SmsManager smsManager = SmsManager.getDefault(); ❷
        try {
            smsManager.sendTextMessage(numero, null, message, null, null);❸
            Toast toast = Toast.makeText(EnvoiSMS.this, "SMS envoyé",
                Toast.LENGTH_LONG); ❹
            toast.show();
        } catch (Exception e) {
            Toast toast = Toast.makeText(EnvoiSMS.this, "Erreur !",
                Toast.LENGTH_LONG); ❺
            toast.show();
        }
    }
}
```

On associe à un bouton d'envoi l'invocation de la méthode `envoiSMS` ❶. On récupère le manager de SMS ❷ et on invoque sur l'instance obtenue la méthode `sendTextMessage` en passant le numéro et le message en paramètres ❸. Suivant la réussite ❹ ou l'échec de l'opération ❺, un message est affiché à destination de l'utilisateur.

Réception de SMS

Pour réceptionner des SMS, il faut avant tout obtenir la permission de les recevoir... À cette fin, il faut ajouter la ligne suivante à la section `manifest` du fichier de configuration de l'application.

Code 12-11 : Ajout de la permission pour recevoir des SMS

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

Ensuite, l'écriture d'un `BroadcastReceiver` qui sera à l'écoute des SMS reçus est nécessaire. L'action à spécifier dans le filtre d'Intents pour associer ce récepteur d'Intents à l'action de réception de SMS est `android.provider.Telephony.SMS_RECEIVED`.

Code 12-12 : Fichier de configuration complété pour la réception de SMS

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.eyrolles.android.telephonie" android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon"android:label="@string/app_name">
```

```
        <activity android:name=".ReceptionSMS" android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <receiver
android:name="com.eyrolles.android.telephonie.SMSBroadcastReceiver"> (1)
            <intent-filter> (2)
               <action android:name="android.provider.Telephony.SMS_RECEIVED" />
            </intent-filter>
        </receiver>
    </application>
    <uses-permission android:name="android.permission.RECEIVE_SMS" /> (3)
    <uses-sdk android:minSdkVersion="3" />
</manifest>
```

Le nom du récepteur (1) est important, il devra correspondre à la classe créée par la suite. Le filtre d'Intents permet de définir l'action qui utilisera le récepteur (2). Enfin, comme d'habitude, il ne faut surtout pas oublier de déclarer les permissions (3) faute de quoi une erreur surviendra.

Code 12-13 : Récepteur qui intercepte les SMS

```
package com.eyrolles.android.telephonie;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.telephony.gsm.SmsMessage;

public class SMSBroadcastReceiver extends BroadcastReceiver { (1)
    private static final String ACTION =
        "android.provider.Telephony.SMS_RECEIVED"; (2)

    public void onReceive(Context context, Intent intent) {

        if (ACTION.equals(intent.getAction())) { (3)

            Object[] pdus = (Object[]) intent.getExtras().get("pdus"); (4)
            SmsMessage[] messages = new SmsMessage[pdus.length];
            for (int i = 0; i < pdus.length; i++) {
                messages[i] = SmsMessage.createFromPdu((byte[]) pdus[i]); (5)
            }

            // Traitement des messages reçus (6).
        }
    }
}
```

Ce code correspond donc à un `BroadcastReceiver` (1) qui intercepte les messages SMS. On vérifie dans la méthode `onReceive` que l'action déclenchée correspond à nos attentes (2)(3).

Ensuite, on extrait le message qui est contenu dans l'objet `Intent` grâce à la méthode `getExtras`. Les instances de `SmsMessage` sont créées (5) grâce à la conversion du tableau d'objets (4) au standard de représentation de messages PDU (*Protocol Description Unit*).

Il reste à utiliser les objets `SmsMessage` créés (6) en récupérant les informations contenues grâce aux méthodes d'accès standards.

Parcourir les répertoires SMS du téléphone

Tout comme pour l'envoi ou la réception de messages texte, une permission est nécessaire afin de scruter le contenu des répertoires dans lesquels les SMS sont classés. Il s'agit de la permission `android.permission.READ_SMS` qui s'ajoute comme suit au fichier de configuration de l'application.

Code 12-14 : Ajout de la permission de lecture des SMS

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

Les dossiers où sont stockés les SMS sur le téléphone sont accessibles à l'aide d'une URI pointant sur le répertoire désiré. Une requête est alors lancée en utilisant l'URI sélectionnée.

Tableau 12–2 Dossiers où sont stockés les SMS

Dossier	URI
Boîte de réception (*Inbox*)	`content://sms/inbox`
Envoyés (*Sent*)	`content://sms/sent`
Brouillons (*Draft*)	`content://sms/draft`
Non remis (*Undelivered*)	`content://sms/undelivered`
En échec (*Failed*)	`content://sms/failed`
Tous (*All*)	`content://sms/all`

Le projet suivant permettra de créer une liste qui contient le corps de chacun des messages disponibles dans la boîte de réception SMS du téléphone.

Code 12-15 : Le fichier XML de mise en page

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:id="@+id/list" android:layout_width="fill_parent"
        android:layout_height="fill_parent" /> (1)
</LinearLayout>
```

Le point à retenir est la présence d'une vue de type `TextView` avec un identifiant `list` (1).

Code 12-16 : L'activité qui liste les SMS du téléphone

```
package com.eyrolles.android.telephonie;

import android.app.ListActivity;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.widget.ListAdapter;
import android.widget.SimpleCursorAdapter;

public class RepertoiresSMS extends ListActivity { (1)

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        lectureSMSBoiteReception();
    }

    private void lectureSMSBoiteReception() {
       Uri uriBoiteReceptionSMS = Uri.parse("content://sms/inbox"); (2)

       Cursor curseur = getContentResolver().query(uriBoiteReceptionSMS,
            null, null, null, null); (3)
       startManagingCursor(curseur);

       String[] nomDesColonnes = new String[] { "body" }; (4)
       int[] vuesQuiAffichentLesColonnes = new int[] { R.id.list};
        ListAdapter adapter = new SimpleCursorAdapter(this, R.layout.main,
            curseur, nomDesColonnes, vuesQuiAffichentLesColonnes); (5)
        setListAdapter(adapter); (6)
    }
}
```

L'activité est de type `ListActivity` (1) ce qui nous permettra un peu plus tard de fixer un adaptateur (5)(6). L'accès aux SMS est possible grâce à la création d'une URI (2) et

l'exécution d'une requête ❸. L'information sélectionnée est le corps du message ❹ grâce à la chaîne `"body"`, mais on aurait pu utiliser `"address"` pour récupérer le numéro de téléphone, `"date"` pour obtenir des informations sur l'heure de réception, etc., c'est-à-dire toutes les informations consultables sur un SMS.

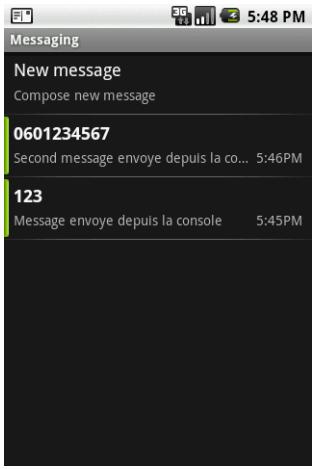


Figure 12–12
Les SMS présents sur l'application par défaut du téléphone

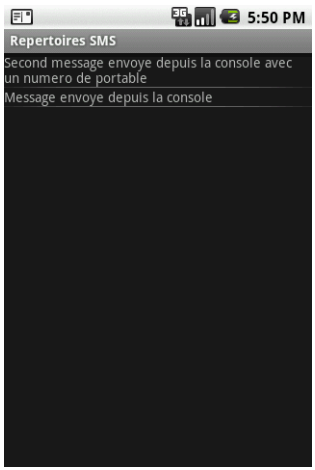


Figure 12–13
Les SMS lus par notre application dans la boîte de réception

En résumé

L'utilisation des API pour gérer les fonctions de téléphonie ne concernera peut-être pas tous vos développements, mais avec le SDK Android, vous disposez d'une grande latitude à la fois pour passer ou filtrer des appels, envoyer ou recevoir des SMS. De plus, tous ces aspects peuvent être testés dans un émulateur assez complet.

Au-delà de l'utilisation des fonctions émettrices qui ont potentiellement un coût pour l'utilisateur, toutes les possibilités de traitement et de listage des appels ou des SMS ouvrent des possibilités pour réaliser des applications comme `chompSMS`. Cette dernière est un bon exemple parmi les applications destinées à remplacer les applications livrées en standard avec Android par d'autres proposant des fonctionnalités plus poussées. Dans le cas précis de `chompSMS`, la présentation des SMS/MMS reçus est totalement revue pour mettre en place une interface semblable à celle disponible sur l'iPhone.

13

Géolocalisation, Google Maps et GPS

Parmi les fonctionnalités les plus appréciées sur les plates-formes mobiles modernes, la géolocalisation permet de réaliser des applications innovantes. Grâce à Google Maps notamment, elle est au cœur d'Android !

La démocratisation des GPS pour l'automobile et les randonneurs a rapidement conduit les constructeurs de téléphones à ajouter des fonctions de géolocalisation à leurs appareils pour créer des applications nomades. Les services de géolocalisation d'Android sont divisés en deux grandes parties :

- les API qui gèrent les plans (dans l'espace de noms `com.google.android.maps`) ;
- les API qui gèrent la localisation à proprement parler (dans l'espace de noms `android.location`).

Nous allons d'abord voir comment déterminer la position d'un appareil sur Android. Ensuite nous verrons en deux temps comment utiliser l'émulateur pour simuler des positions en étudiant d'abord la génération de fichiers de points et ensuite leur emploi. La suite du chapitre sera consacrée aux autres capacités de la plate-forme et notamment à l'affichage de cartes Google Maps ainsi qu'aux différents dessins ou affichages réalisables grâce à ces mêmes cartes.

À LIRE

📖 Paul Corréia, *Mon GPS en action ! Créer et enrichir ses cartes avec Google Earth, Google Maps, OpenStreetMap...*, Eyrolles 2010

Déterminer la position courante : plusieurs moyens

Lorsqu'il s'agit de déterminer la position courante d'un appareil, plusieurs problèmes peuvent être rencontrés :

- tous les appareils ne disposent pas tous du même matériel de géolocalisation (certains n'ont pas de récepteur GPS) ;
- les conditions d'utilisation du téléphone peuvent rendre inutilisable une méthode de localisation (cas des fonctionnalités GPS dans un tunnel, par exemple).

C'est pourquoi Android offre plusieurs moyens de localisation au travers d'une liste de *fournisseurs de position* : selon les conditions du moment ou vos propres critères, Android se chargera de sélectionner le plus apte à donner la position de l'appareil.

Il faudra donc récupérer cette liste de fournisseurs et déterminer celui qui est le plus adapté à l'utilisation souhaitée. De manière générale, on distingue deux types de fournisseurs naturels :

- le fournisseur basé sur la technologie GPS, de type `LocationManager.GPS_PROVIDER`. C'est le plus précis des deux, mais c'est également le plus consommateur en termes de batterie ;
- le fournisseur qui se repère grâce aux antennes des opérateurs mobiles et aux points d'accès Wi-Fi, de type `LocationManager.NETWORK_PROVIDER`.

ÉMULATEUR **Spécificités pour l'utilisation virtuelle des fournisseurs de position**

L'unique fournisseur de position factice disponible au travers de l'émulateur est basé sur la réception GPS : `LocationManager.GPS_PROVIDER`. Inutile donc de chercher à utiliser d'autres méthodes avec cet outil. Gardez cependant bien en tête qu'il existe d'autres fournisseurs de position.

Mettons de côté la recherche d'un fournisseur en fonction de critères (ce que nous aborderons plus loin) pour nous concentrer sur l'obtention de la liste de fournisseurs.

Obtenir la liste des fournisseurs de position

La classe des fournisseurs de position, `LocationProvider`, est une classe abstraite : chacune de ses différentes implémentations (correspondant aux différents moyens de

localisation à notre disposition) fournit l'accès aux informations de localisation d'une manière qui lui est propre.

Le code suivant permet d'obtenir la liste de tous les fournisseurs.

```
LocationManager locationManager =
   (LocationManager)getSystemService(Context.LOCATION_SERVICE);
List<String> fournisseurs = locationManager.getAllProviders();
```

À un instant donné, tous les outils de géolocalisation de l'appareil ne sont peut-être pas disponibles : l'utilisateur peut en effet décider de désactiver certains moyens (par exemple, le GPS peut être désactivé pour économiser les batteries). Le code suivant vous permet d'obtenir la liste des fournisseurs activés :

```
LocationManager locationManager =
   (LocationManager)getSystemService(Context.LOCATION_SERVICE);
List<String> fournisseurs = locationManager.getAllProviders(true);
```

Si l'on souhaite obtenir un fournisseur particulier, voici le code à taper :

```
String fournisseur = LocationManager.GPS_PROVIDER;
LocationProvider gpsProvider;
gpsProvider = locationManager.getProvider(fournisseur);
```

Comme pour la plupart des fonctionnalités sous Android, pour utiliser ces quelques lignes de code, il ne faut pas oublier de déclarer les permissions appropriées dans la section `manifest` du fichier de configuration de l'application.

L'utilisation du fournisseur de type GPS est liée à la déclaration de permission suivante :

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

L'utilisation du fournisseur réseau dépend quant à elle de la permission :

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

On retrouve ici la notion de localisation grossière ou fine suivant la méthode utilisée par l'implémentation.

Choix d'un fournisseur de position en fonction de critères

Une méthode existe dans Android pour choisir un fournisseur à partir de besoins que vous formulez, ce qui permet de déterminer le fournisseur le plus adapté à vos attentes.

La classe centrale pour la définition des critères est `Criteria`. En voici un exemple d'utilisation :

```
Criteria criteres = new Criteria();
// Localisation la plus précise possible
criteres.setAccuracy(Criteria.ACCURACY_FINE);
// Altitude fournie obligatoirement
criteres.setAltitudeRequired(true);
```

Différents critères sont utilisables comme la vitesse, la consommation d'énergie, l'utilisation éventuelle de services payants, etc.

Une fois les critères définis, on utilise la méthode `getBestProvider` de l'instance de la classe `LocationManager` récupérée un peu plus tôt. Le booléen en paramètre donne la possibilité de se restreindre aux fournisseurs activés uniquement.

```
String fournisseurSelectionne =
   locationManager.getBestProvider(criteria, true);
```

Si aucun fournisseur correspondant à vos critères n'est trouvé, la documentation précise que ces derniers sont abandonnés un par un pour essayer de trouver la meilleure solution possible avec en premier critère la consommation d'énergie, puis la précision et enfin les informations comme l'altitude et la vitesse.

Les formats des fichiers de position pour l'émulateur

Les positions proposées varient avec les variations physiques subies par le matériel.

Or, dans le cas de l'utilisation de l'émulateur, nous ne disposons pas de ces variations physiques puisque notre matériel est virtualisé – et vous n'allez sans doute pas développer sur votre ordinateur portable, installé sur le siège arrière d'une voiture en mouvement.

Pour pallier ce manque, le kit de développement Android permet de simuler un fournisseur de position pour donner à l'émulateur une position ou une suite de positions suivant différents formats.

Le format GPX

Le format GPX (*GPS eXchange*) est un format qui exploite le métalangage XML de façon relativement légère pour échanger des données GPS entre applications. Il est utilisé par de nombreux logiciels et appareils et sa documentation complète est disponible sur le site http://www.topografix.com.

La structure de ce format est formalisée dans un schéma (disponible sur le site indiqué précédemment) : vous pouvez le comparer à vos fichiers XML pour valider leur conformité. Des données de trois types différents peuvent être spécifiées :

- les *waypoints*, qui correspondent à une liste de points sans ordre précis (par exemple, toutes les stations services de France) ;
- les *tracks*, qui correspondent aux enregistrements des déplacements d'une personne ;
- les *routes*, qui sont des planifications d'itinéraires destinés à être parcourus.

Voici un extrait d'un exemple disponible sur le site TopoGrafix, avec des éléments *waypoint* (1) et *track* (2) :

Extrait du fichier GPX Duathlon Rockbuster sur le site TopoGrafix

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<?xml-stylesheet type="text/xsl" href="details.xsl"?>
<gpx
version="1.0"
creator="ExpertGPS 1.1.1 - http://www.topografix.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.topografix.com/GPX/1/0"
xmlns:topografix="http://www.topografix.com/GPX/Private/TopoGrafix/0/1"
xsi:schemaLocation="http://www.topografix.com/GPX/1/0 http://
www.topografix.com/GPX/1/0/gpx.xsd http://www.topografix.com/GPX/Private/
TopoGrafix/0/1 http://www.topografix.com/GPX/Private/TopoGrafix/0/1/
topografix.xsd">

<name><![CDATA[Rockbuster Duathlon at Ashland State Park]]></name>
<desc><![CDATA[Team TopoGrafix tracklogs from the Rockbuster Duathlon at Ashland
State Park, April 21st, 2002.]]></desc>
<author><![CDATA[Vil and Dan]]></author>
<email>trails@topografix.com</email>
<url><![CDATA[http://www.topografix.com/team/photos.asp]]></url>
<urlname><![CDATA[Team TopoGrafix Pics at the Rockbuster Duathlon]]></urlname>
<time>2002-04-23T15:35:23Z</time>
<keywords><![CDATA[mountain biking, racing, ashland, framingham,
rockbuster]]></keywords>
<bounds minlat="42.223808" minlon="-71.493169" maxlat="42.261090"
        maxlon="-71.457800"/>
<wpt lat="42.246950" lon="-71.461807"> (1)
<name><![CDATA[AQUADUCT]]></name>
<desc><![CDATA[Aquaduct]]></desc>
<sym>Dam</sym>
<type><![CDATA[Dam]]></type>
</wpt>
<wpt lat="42.261090" lon="-71.463360">
<ele>57.302400</ele>
```

```
<desc><![CDATA[Ashland]]></desc>
<sym>City (Medium)</sym>
<type><![CDATA[Populated Place]]></type>
</wpt>
<trk> 2
<name><![CDATA[ASP QUARRY]]></name>
<desc><![CDATA[Quarry Route in Ashland]]></desc>
<number>11</number>
<topografix:color>c0c0c0</topografix:color>
<trkseg>
<trkpt lat="42.233167" lon="-71.475141">
<sym>Waypoint</sym>
</trkpt>
<trkpt lat="42.231813" lon="-71.477802">
<sym>Waypoint</sym>
</trkpt>
</trkseg>
</trk>
</gpx>
```

Le format KML de Google Earth

KML (*Keyhole Markup Language*) est un langage basé sur XML et utilisé pour afficher des données géospatiales dans les applications Google comme Google Earth et Google Maps.

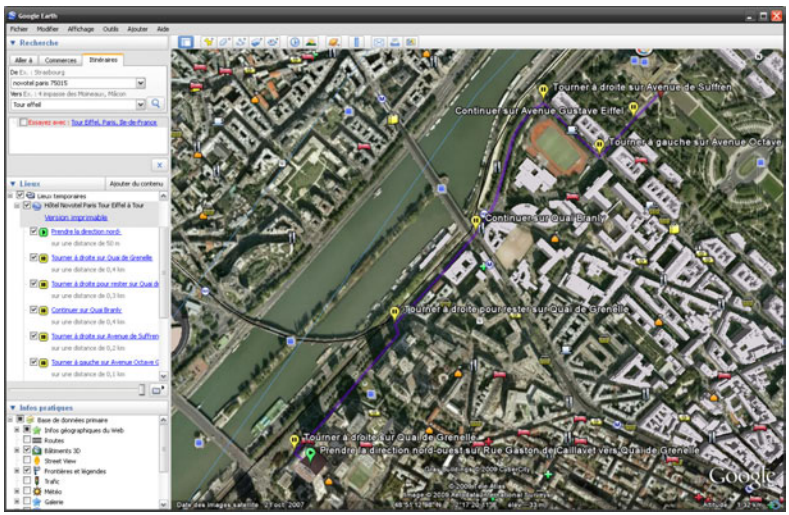


Figure 13–1
Création d'un fichier KML avec Google Earth

Avant d'étudier rapidement sa structure, générons un fichier KML grâce à l'outil Google Earth. Ce logiciel permet de visualiser la Terre, grâce à des images satellites, ou même des bâtiments en 3D que chacun peut construire. Vous pouvez en télécharger une version gratuite à l'adresse suivante : http://earth.google.fr/. Une fois le logiciel installé, créez un itinéraire et sauvegardez-le sous forme de fichier KML.

En prenant un itinéraire qui va d'un hôtel parisien à la Tour Eiffel, on obtient un fichier qui, à deux ou trois simplifications près, ressemble à l'extrait suivant.

Extrait du fichier KML pour aller d'un hôtel parisien à la Tour Eiffel

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2"
     xmlns:gx="http://www.google.com/kml/ext/2.2"
     xmlns:kml="http://www.opengis.net/kml/2.2"
     xmlns:atom="http://www.w3.org/2005/Atom">
<Document>
   <name>Hôtel Novotel Paris Tour Eiffel à Tour Eiffel, Paris</name>
   <open>1</open>
   <Snippet maxLines="2"><![CDATA[<font size=+1><a href="http://maps.google.fr/
maps?saddr=novotel+paris+75015&daddr=Tour+Eiffel,+Paris,+Ile-de-
France&ie=UTF8&v=2.2&cv=5.0.11733.9347&hl=fr&ct=clnk&cd=1&f=d&ge_fileext=.kml">
Version imprimable</a></font>]]></Snippet>
   <Placemark> (1)
      <name>Prendre la direction nord-ouest sur Rue Gaston de Caillavet vers
Quai de Grenelle</name>
      <address>Hôtel Novotel Paris Tour Eiffel</address>
      <description>sur une distance de 50&amp;#160;m</description>
      <LookAt>
         <longitude>2.2834</longitude>
         <latitude>48.84954</latitude>
         <altitude>0</altitude>
         <range>100</range>
         <tilt>45</tilt>
         <heading>308.237518</heading>
      </LookAt>
      <StyleMap>
      (...)
      </StyleMap>
      <Point> (2)
         <coordinates>2.2834,48.84954000000001,0</coordinates>
      </Point>
   </Placemark>
   (...)
</Document>
</kml>
```

A priori, vous n'aurez pas à créer vos fichiers KML manuellement mais si toutefois vous deviez le faire, toute la documentation nécessaire est disponible sur le site suivant : http://code.google.com/intl/fr/apis/kml/.

Retenez simplement la présence des balises `Placemark` (1) qui contiennent des éléments `Point` (2) avec les coordonnées des points clés sur le trajet.

Malheureusement, la théorie n'est pas tout à fait en adéquation avec la pratique. Avec les versions logicielles utilisées, le KML doit avoir un espace de noms (*namespace*) bien particulier. La balise ouvrante du XML doit ainsi être la suivante.

```
<kml xmlns="http://earth.google.com/kml/2.x">
```

Lors du chargement du fichier, l'analyse réalisée de façon automatique ne recherche que quelques informations dans notre fichier. Le fichier KML peut ainsi être simplifié, pour obtenir le fichier suivant, qui pourra vous servir de référence pour débuter vos applications. Notez bien le nouvel espace de noms (1) et la simplicité du contenu des balises `Placemark` (2).

Exemple de fichier KML simplifié et préparé pour nos outils

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.x"> (1)
<Document>
   <name>Hôtel Novotel Paris Tour Eiffel à Tour Eiffel, Paris</name>
   <Placemark>
      <name>Prendre la direction nord-ouest sur Rue Gaston de Caillavet
vers Quai de Grenelle</name>
      <address>Hôtel Novotel Paris Tour Eiffel</address>
      <Point>
         <coordinates>2.2834,48.84954000000001,0</coordinates>
      </Point>
   </Placemark>
   <Placemark> (2)
      <name>Tourner à droite sur Quai de Grenelle</name>
      <Point>
         <coordinates>2.28618,48.85261,0</coordinates>
      </Point>
   </Placemark>
   <Placemark>
      <name>Continuer sur Quai Branly</name>
      <Point>
         <coordinates>2.28887,48.85459,0</coordinates>
      </Point>
   </Placemark>
```

```
    <Placemark>
       <Point>
          <coordinates>2.2911,48.85743999999999,0</coordinates>
       </Point>
    </Placemark>
    <Placemark>
       <name>Tourner à gauche sur Avenue Octave Greard</name>
       <Point>
          <coordinates>2.29296,48.85625,0</coordinates>
       </Point>
    </Placemark>
    <Placemark>
       <name>Continuer sur Avenue Gustave Eiffel</name>
       <Point>
          <coordinates>2.29408,48.85706,0</coordinates>
       </Point>
    </Placemark>
    <Placemark>
       <name>Arrivée à : Tour Eiffel, Paris</name>
       <Point>
          <coordinates>2.29498,48.85767,0</coordinates>
       </Point>
    </Placemark>
    </Document>
</kml>
```

BON À SAVOIR **À propos des fichiers .kmz**

Les fichiers portant l'extension `.kmz` sont également des fichiers au format KML mais compressés.

Simulateur et fausses positions

Une fois le format choisi et les données préparées, pour communiquer les positions au simulateur, deux options s'offrent à vous : soit l'utilisation de l'interface graphique du complément ADT pour Eclipse, soit l'utilisation de la ligne de commande.

Envoi des coordonnées à l'aide d'une interface graphique

Le complément ADT d'Eclipse met à votre disposition trois onglets, *Manuel*, *GPX* et *KML*, qui regroupent les commandes possibles dans des formulaires. Chacun de ces onglets est assez explicite et permet d'envoyer une position fixe ou de parcourir une séquence de positions préparée dans l'un des deux formats et chargée.

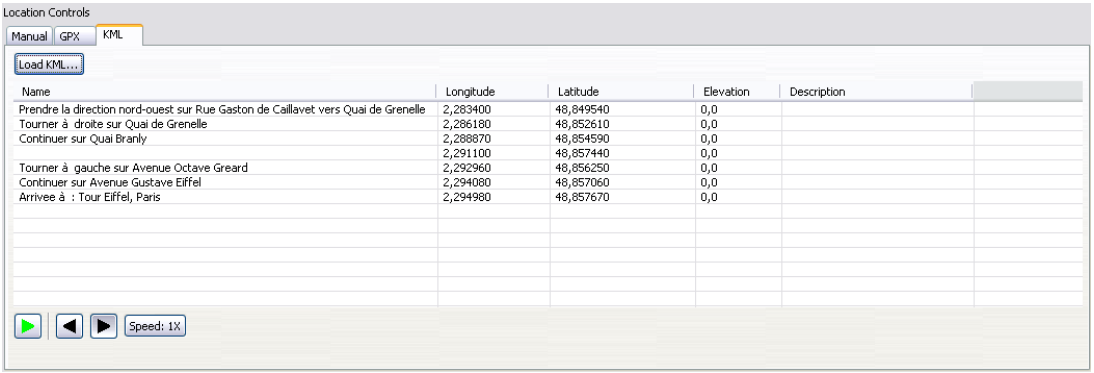


Figure 13–2 Rendu de l'import d'un fichier KML dans la vue Eclipse

> **GPX ou KML Quel format choisir ?**
>
> KML est le format que nous utiliserons pour les exemples, notamment parce qu'il est très facile à générer et s'interface idéalement avec les outils Google. Cependant, sachez que DDMS ne supporte pas encore (au moment de l'écriture de ce livre tout du moins) toutes les fonctionnalités de KML, en ce qui concerne la prise en compte des temps. Si vous choisissez KML, DDMS prendra toutes les secondes les informations d'une nouvelle balise `Placemark`.

Envoi des positions en ligne de commande

Il est donc également possible d'envoyer des coordonnées via des commandes texte. Pour cela, il faut lancer une session Telnet sur le numéro de port indiqué par votre émulateur dans la fenêtre de titre. (Reportez-vous au chapitre précédent pour davantage d'informations sur le protocole Telnet et son utilisation.) Par exemple, si le numéro de port est 5554, la commande initialisant une session Telnet sera :

```
telnet localhost 5554
```

Ensuite, la commande `help` permettra d'obtenir toutes les opérations réalisables. Celle qui nous intéresse est la suivante, pour indiquer un point d'une certaine latitude et longitude :

```
geo fix latitude longitude
```

> **Bon à savoir Problème de fonctionnement de l'émulateur**
>
> Toutes ces explications peuvent ne pas fonctionner – notamment en version 1.5 – tant que vous n'aurez pas paramétré la langue de votre système en anglais. Espérons que cela sera vite résolu dans les versions suivantes.

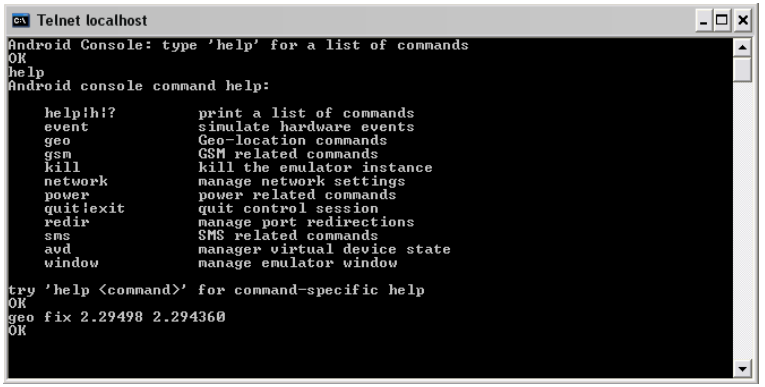


Figure 13–3
Session Telnet pour fournir des coordonnées à l'émulateur

Déterminer votre position

La plupart des applications qui utilisent les fonctionnalités de ce chapitre auront au moins un point en commun : elles ont besoin de savoir où se trouve l'appareil que vous utilisez.

Ensuite, ce sera l'utilisation de cette donnée qui variera : Google Sky déterminera les étoiles visibles de là où vous vous situez, les applications de recherche contextuelles chercheront les points d'intérêt à proximité, etc.

Obtenir sa position

Une seule ligne de code suffit pour récupérer sa position. Il faut cependant choisir auparavant un fournisseur de position. Une fois cette étape réalisée, on utilise la méthode `getLastKnownLocation` de l'élément `LocationManager`, avec comme paramètre le fournisseur de position choisi. Le retour de cette méthode est une instance d'un objet `Location` si une position a été calculée, sinon son résultat vaut `null`.

Nous allons créer un programme qui listera les fournisseurs activés et les positions calculées par chacun de ces fournisseurs. N'oubliez pas d'ajouter les permissions indiquées précédemment à la section `manifest` du fichier de configuration de votre application.

Code 13-1 : Coordonnées de l'appareil avec les différents fournisseurs

```
import java.util.List;
import android.app.Activity;
import android.content.Context;
import android.location.Location; (1)
import android.location.LocationManager;
import android.os.Bundle;
```

```
import android.widget.TextView;

public class MaPosition extends Activity {
   LocationManager locationManager;

   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      String locationContext = Context.LOCATION_SERVICE; (2)
      locationManager = (LocationManager) getSystemService(locationContext);
      majCoordonnes();
   }

   public void majCoordonnes() {
      StringBuilder stringBuilder = new StringBuilder();
      stringBuilder.append("Fournisseurs :");
      List<String> providers = locationManager.getProviders(true); (3)
      // Pour chaque fournisseur ...
      for (String provider : providers) {
         stringBuilder.append("\n");
         stringBuilder.append(provider);
         stringBuilder.append(" : ");

         // ... on affiche la position.
         Location location = locationManager.getLastKnownLocation(provider);
         if (location != null) { (4)
            double latitude = location.getLatitude();
            double longitude = location.getLongitude();
            stringBuilder.append(latitude);
            stringBuilder.append(", ");
            stringBuilder.append(longitude);
         } else {
            stringBuilder.append("Non déterminée");
         }
      }
      Log.d("MaPosition", stringBuilder.toString()); (5)
   }
}
```

Notez qu'on utilise les classes du paquetage `android.location` (1). La première opération consiste à récupérer l'instance du `LocationManager` (2). Ensuite, on invoque la méthode `majCoordonnees` qui va se charger de récupérer des coordonnées (4) pour chacun des fournisseurs de position (3). Chaque affichage est logué (5) et donc consultable facilement grâce à la vue *LogCat* du complément ADT pour Eclipse.

Détecter le changement de position

Détecter le changement relève de deux problématiques : recevoir des mises à jour de sa position et détecter le mouvement.

Ces deux problématiques de changement se résolvent par l'utilisation de la même méthode du `LocationManager`.

```
requestLocationUpdates(fournisseur, temps, distance, locationListener)
```

Le premier paramètre est le fournisseur de position souhaité. Ensuite, le temps entre deux mises à jour est exprimé en millisecondes. Attention à ne pas mettre de valeur trop faible, qui accaparerait les ressources du téléphone et viderait votre batterie. La documentation du SDK recommande une valeur minimale de 60 000 ms. La distance correspond au nombre de mètres qui doivent être parcourus avant de recevoir une nouvelle position. Si elle est supérieure à zéro, la mise à jour s'effectuera uniquement une fois cette distance parcourue. Enfin, le dernier paramètre est l'écouteur (une implémentation de l'interface `LocationListener`) qui recevra les diverses notifications.

Pour arrêter les mises à jour, il faut fournir au gestionnaire de localisation l'instance de l'écouteur à retirer.

```
locationManager.removeUpdates(locationListener);
```

Code 13-2 : Activité qui met à jour la position de l'appareil

```
import android.app.Activity;
import android.content.Context;
import android.location.Location;
import android.location.LocationManager;
import android.os.Bundle;
import android.widget.TextView;

public class MaPositionMaj extends Activity {
   private LocationManager locationManager;
   private String locationProvider = LocationManager.GPS_PROVIDER; (1)

   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      String locationContext = Context.LOCATION_SERVICE; (2)
      locationManager = (LocationManager) getSystemService(locationContext);
      if (locationManager != null && locationProvider != null) {
         majCoordonnes(); (3)
         locationManager.requestLocationUpdates(locationProvider, 6000,
```

```
            100, new MajListener());4
      }
   }

   /**
    * Méthode de l'exemple précédent reprise et adaptée
    * pour loguer les mises à jour.
    */
   public void majCoordonnes() {
      StringBuilder stringBuilder = new StringBuilder();
      stringBuilder.append("Fournisseurs :");
      stringBuilder.append("\n");
      stringBuilder.append(locationProvider);
      stringBuilder.append(" : ");
      Location location =
         locationManager.getLastKnownLocation(locationProvider);
      if (location != null) {
         double latitude = location.getLatitude();
         double longitude = location.getLongitude();
         stringBuilder.append(latitude);
         stringBuilder.append(", ");
         stringBuilder.append(longitude);
      } else {
         stringBuilder.append("Non déterminée");
      }
      Log.d("MaPositionMaj", stringBuilder.toString());
   }


   /**
    * Écouteur utilisé pour les mises à jour des coordonnées.
    */
   class MajListener implements android.location.LocationListener { 5
      public void onLocationChanged(Location location) { 6
         majCoordonnes();
      }
      public void onProviderDisabled(String provider){
      }
      public void onProviderEnabled(String provider){
      }
      public void onStatusChanged(String provider, int status, Bundle extras){
      }
   };
}
```

Nous ne donnerons pas de capture d'écran pour cet exemple dans la mesure où il s'agit uniquement de la mise à jour de coordonnées au format texte.

Nous choisissons le fournisseur de contenu GPS (1) dont nous obtenons une instance (2). Concernant la méthode de mise à jour des coordonnées en fonction des fournisseurs de position (3), elle est identique à celle utilisée dans précédemment dans la section précédente. Elle sert pour obtenir une première position mais également lorsqu'un changement a été détecté.

Le point important est la demande de notification de mise à jour demandée au `LocationManager` en fonction d'un temps et d'une distance (4). On fournit également à cette méthode un `LocationListener` (5) qui sera notifié des mises à jour (6), ce qui nous permettra de changer les coordonnées avec la méthode `majCoordonnees`.

Alertes de proximité

À la différence des techniques étudiées précédemment qui vous permettent de connaître votre localisation précise ou de savoir si vous êtes en mouvement, les alertes de proximité indiquent qu'on se rapproche d'un lieu prédéfini.

La méthode `addProximityAlert` de la classe `LocationManager` permet d'enregistrer un `PendingIntent` qui sera déclenché quand l'appareil se trouvera à une certaine distance d'un point fixé. La méthode a la signature suivante :

```
public void addProximityAlert(double latitude, double longitude, float
rayon, long expiration, PendingIntent pendingIntent)
```

où :

- `latitude` correspond à la latitude du point d'alerte ;
- `longitude` est la longitude du point d'alerte ;
- `rayon` correspond au rayon autour du point d'alerte qui déterminera la sensibilité de la détection ;
- `expiration` définit une période en millisecondes à la fin de laquelle l'élément `LocationManager` retirera le point d'alerte des alertes surveillées. La valeur -1 signifie que l'enregistrement est valide jusqu'à ce qu'il soit retiré manuellement par la méthode `removeProximityAlert ;`
- `pendingIntent` est utilisé pour générer l'intention déclenchée quand l'appareil rentre ou sort de la zone désignée par le point et le rayon.

En lisant attentivement la documentation de la fonction `addProximityAlert`, vous constaterez que l'intention `PendingIntent` est étendue avec un extra dont la clé est `KEY_PROXIMITY_ENTERING`. On obtient un booléen dont la valeur est `true` si on pénètre dans la zone définie et `false` si on la quitte.

La gestion de l'alerte se réalise grâce à un `BroadcastReceiver` :

```
public class ProximityIntentBroadcastReceiver extends BroadcastReceiver {
   public void onReceive (Context context, Intent intent) {
      // Test pour détecter si on entre ou si on sort de la zone.
      String key = LocationManager.KEY_PROXIMITY_ENTERING;
      Boolean entering = intent.getBooleanExtra(key, false);
   }
}
```

Le code suivant permet de l'enregistrer :

```
IntentFilter intentFilter = new IntentFilter(PROXIMITY_ALERT);
registerReceiver(new ProximityIntentBroadcastReceiver(), intentFilter);
```

POUR ALLER PLUS LOIN Fonctionnement avancé des alertes de proximité

La documentation relative aux API apporte quelques informations supplémentaires sur le mécanisme des alertes de proximité :

- quand l'écran s'éteint, la vérification pour déclencher ces alertes n'a plus lieu que toutes les 4 minutes pour économiser la batterie ;
- les réglages utilisés en interne sont `NETWORK_PROVIDER` et `GPS_PROVIDER`.

Les API Google

L'utilisation des cartes Google Maps et de certaines fonctionnalités comme le géo(dé)codage sont liées à l'API Google. Il faut préparer l'émulateur pour cette utilisation particulière, opération qui nous servira un peu plus loin quand nous manipulerons les cartes. Nous allons aussi apprendre à générer une clé - fournie par Google - qui nous permettra d'utiliser Google Maps.

Présentation

Le complément Google API est une extension de l'environnement de développement, le SDK d'Android. Ses fonctionnalités permettent aux applications que vous développerez d'utiliser facilement les services Google et d'accéder aux données associées.

Pour le moment, ces bibliothèques ne sont disponibles que sous forme de modules et il faut installer entièrement l'environnement de développement sur votre ordinateur. Vous pouvez ensuite accéder aux classes de la bibliothèque Maps et compiler vos applications avec cette dernière. L'extension Google API est fournie avec une image système qui permet de tester vos applications sur l'émulateur. D'ailleurs, vous consta-

terez que dans l'émulateur avec une configuration standard 1.5 par exemple, il n'y a pas d'icône pour consulter Google Maps, l'application n'est donc pas disponible.

Les API Google additionnelles sont donc composées :

- de la bibliothèque Maps ;
- d'une image système avec la bibliothèque incorporée ;
- d'une application de démonstration ;
- de la documentation complète pour utiliser les fonctionnalités proposées.

Utilisation des API Google avec l'émulateur

Puisque ces fonctionnalités additionnelles ne sont pas disponibles en téléchargement indépendant au moment de l'écriture de ce livre, il n'y a pas d'étape d'installation supplémentaire.

Les SDK 2.x ajoutent cependant la notion de package qui permet de sélectionner ou non ces API et donc de ne pas les installer. Assurez-vous de ne pas les avoir oubliées !

Ensuite, il faut configurer l'environnement de développement correctement.

Configuration d'Eclipse

Tout d'abord, la première étape est la création d'un nouvel appareil virtuel (AVD) qui utilisera l'image système adéquate. Pour cela, lancez l'interface *Android Virtuel Devices Manager* du plug-in Eclipse et créez un nouvel appareil virtuel qui aura comme cible Google API, comme le montre la capture d'écran suivante.

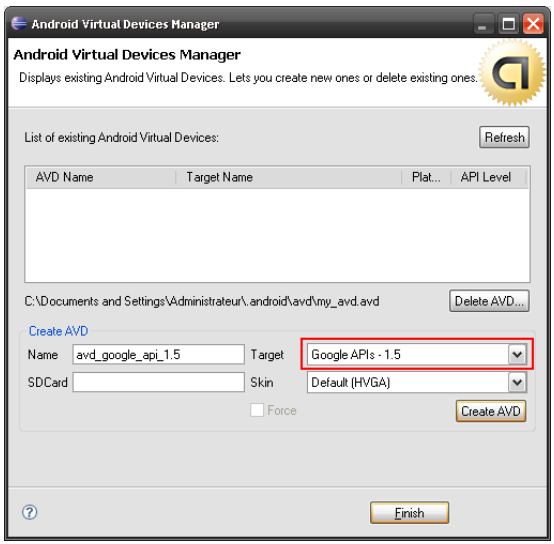


Figure 13–4
Création du nouvel appareil virtuel (AVD)

S'il y a plusieurs appareils virtuels, il vous faudra choisir lequel utiliser dans les propriétés d'exécution. Si l'appareil sélectionné pour l'exécution n'est pas compatible, une recherche sera lancée pour en trouver un automatiquement afin d'éviter un échec.

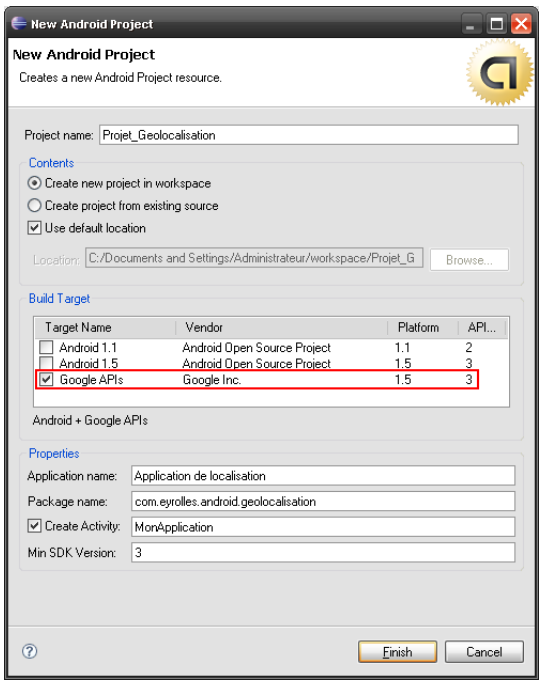


Figure 13–5
Réglage des propriétés du projet

Ensuite, il faut soigneusement régler les propriétés du projet, soit en cours d'utilisation, soit à la création. Ceci permettra à Eclipse de lier correctement les archives Java au projet et ainsi de permettre la complétion en cours de développement et d'assurer le succès des compilations du projet. Sélectionnez comme cible les API Google qui correspondent à la plate-forme souhaitée, ici en version 1.5.

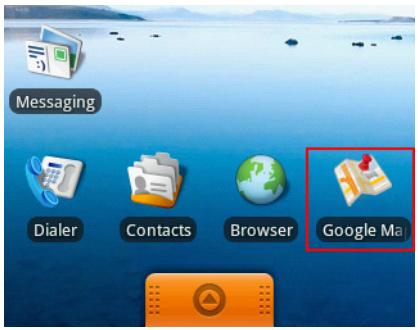


Figure 13–6
Rendu sur l'émulateur

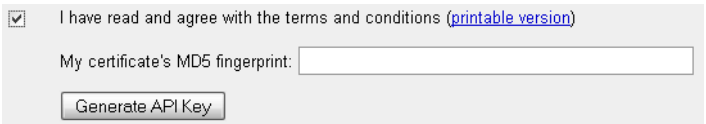
Une fois ces étapes réalisées, vous devez constater la présence de l'archive `maps.jar` dans le chemin de compilation du projet. De plus, au lancement de l'émulateur, l'application Google Maps est disponible.

Obtention d'une clé pour utiliser Google Maps

Nous aurons également besoin de cette configuration pour toute la partie concernant la manipulation de cartes via l'API Google Maps. Une étape supplémentaire sera alors nécessaire : l'obtention d'une clé de licence.

Pour cela, la première étape consiste à se rendre sur le site proposé par Google. Il vous faudra également un compte Google pour compléter l'opération. Enfin, pour obtenir cette clé, l'interface du site vous demande une empreinte MD5 d'un certificat.

Figure 13–7
Formulaire de génération d'une clé Google Maps



De quoi s'agit-il ? Eh bien Android n'autorise l'installation que des applications signées. Avant d'installer une application via l'émulateur, Eclipse signe votre application en utilisant un certificat de débogage qui est fourni avec le SDK d'Android.

Eclipse crée le fichier `debug.keystore` lors de la compilation du projet, ce fichier étant stocké dans le répertoire `.android` dont le chemin varie en fonction du système d'exploitation. Pour un système sous Windows XP, par exemple, le chemin sera de la forme `C:\Documents and Settings\Nom d'Utilisateur\.android`.

La clé générée pour utiliser l'API Google Maps est basée sur ce certificat de débogage. Pour en obtenir l'empreinte (et en se basant sur un système XP logué en administrateur), il faut utiliser la commande suivante.

```
keytool -list -keystore "C:\Documents and Settings\Administrateur\
.android\debug.keystore"
```

Une fois l'empreinte saisie dans le formulaire et envoyée au site, une clé est générée ainsi qu'un morceau de code XML que nous mettons de côté pour l'instant.

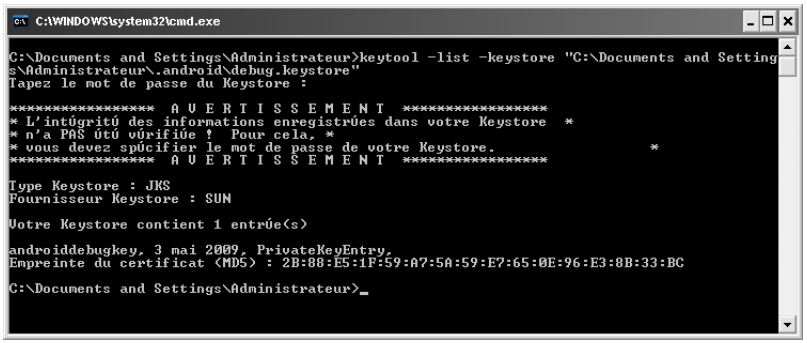


Figure 13–8
Résultat de la commande keytool pour obtenir l'empreinte du certificat de débogage

> Pour des informations concernant la signature d'une application, notamment avant sa publication, reportez-vous au chapitre 15 sur l'utilisation de l'Android Market.

Conversion d'adresses et d'endroits

Le géocodage permet de déterminer des coordonnées en latitude et longitude à partir d'une adresse ou d'une description d'un endroit. À l'inverse, le géodécodage permet de retrouver un lieu en fonction de ses coordonnées. Ces fonctionnalités sont liées à l'API Google. Pour pouvoir les manipuler, découvrez-les et préparez votre émulateur grâce à la section précédente qui décrit pas à pas la configuration d'un projet.

Géocodage

Le géocodage permet donc d'obtenir des coordonnées à partir d'un texte décrivant un endroit. Le format des adresses, noms de stations ou codes postaux dépend évidemment de la zone géographique concernée. Pour cela, on utilise la classe `Geocoder` en créant une instance relative à la localisation dans laquelle nous allons faire nos recherches. Ensuite, la méthode `getFromLocationName` se charge de trouver *n* résultats au maximum correspondant à une adresse passée en paramètre.

Code 13-3 : Exemple d'utilisation du géocodage

```
private void rechercheCoordonnees(String adresse) {
   Geocoder geocoder = new Geocoder(this, Locale.FRANCE); (1)
   List<Address> endroits = null;
   try {
      endroits = geocoder.getFromLocationName(adresse, 5); (2)
```

```
        for (Address address : endroits) { 3
            double latitude = address.getLatitude(); 4
            double longitude = address.getLatitude();
            Log.i("Geocoder", "(" + latitude + "," + longitude + ")");
        }
    } catch (IOException e) {
        Log.e("Geocoder", e.getMessage());
    }
}

// Utilisation de la méthode créée plus haut.
String adresseATransformer = "10 rue de la Convention, 75015, Paris";
rechercheCoordonnees(adresseATransformer);
```

Dans cet exemple, nous recherchons une adresse dans Paris dans un contexte français 1. On aurait pu fixer cette locale grâce à l'appel `Locale.getDefault` pour déterminer la locale par défaut. L'adresse que nous cherchons est passée en paramètre à la méthode de l'instance `Geocoder` avec le nombre maximal de résultats retournés 2. Les résultats sont parcourus 3 et on extrait les coordonnées 4.

À noter qu'il existe une surcharge de cette méthode qui permet de restreindre les résultats à une zone géographique définie par des coordonnées.

```
Geocoder                Lancement de la recherche geo
InetAddress             www.google.com: 209.85.227.99 (family 2, proto 6)
InetAddress             www.google.com: 209.85.227.104 (family 2, proto 6)
InetAddress             www.google.com: 209.85.227.147 (family 2, proto 6)
InetAddress             www.google.com: 209.85.227.103 (family 2, proto 6)
LocationMasfClient      getAddressFromProtoBuf(): Ignore feature 9,10
LocationMasfClient      getAddressFromProtoBuf(): Ignore feature 0,15th arrondissement of Paris
Geocoder                Nb endroits : 1
Geocoder                (48.8453972,48.8453972)
ActivityManager         Displayed activity com.eyrolles.android.geolocalisation/.ExempleGeocoder: 1211 ms
```

Figure 13–9 Résultat de la transformation d'une adresse en coordonnées

Géodécodage

À l'inverse, le géodécodage retrouvera une situation géographique en fonction des coordonnées choisies.

Code 13-4 : Exemple d'utilisation du géodécodage

```
private void rechercheEndroit(double latitude, double longitude) {
    String strEndroit = "Non trouvé";
    Geocoder geocoder = new Geocoder(this, Locale.FRANCE); 1
    try {
        List<Address> adresses = geocoder.getFromLocation(latitude, longitude, 2); 2
        StringBuilder sb = new StringBuilder();
        for (Address adresse : adresses) { 3
            for (int i = 0; i < adresse.getMaxAddressLineIndex(); i++) {
                sb.append(adresse.getAddressLine(i));
```

```
            sb.append("\n");
         }
      }
      strEndroit = sb.toString();
   } catch (IOException e) {
      Log.e("Geodecoder", e.getMessage());
   }
   Log.i("Geodecoder", "Emplacement : \n" + strEndroit);
}

// Utilisation de la méthode décrite plus haut.
rechercheEndroit(48.8453972, 2.2783926);
```

Une fois une instance de `Geocoder` créée avec la bonne locale ou la locale par défaut (1), on obtient la liste (ici limitée à 2) des sites correspondant à une localisation (2).

Pour chaque emplacement récupéré (3), on parcourt l'ensemble des lignes de la description qu'on stocke dans un tampon. Le résultat final est affiché sans interface graphique, en utilisant le système de journalisation proposé par la plate-forme. Vous pouvez voir toutes ces traces grâce à la vue *LogCat* du complétement ADT pour Eclipse.

```
Geocoder              Recherche du nom de l'emplacement
LocationMasfClient    getAddressFromProtoBuf(): Ignore feature 9,4-12
LocationMasfClient    getAddressFromProtoBuf(): Ignore feature 0,15Ã¨me Arrondissement Paris
LocationMasfClient    getAddressFromProtoBuf(): Ignore feature 0,15Ã¨me Arrondissement Paris
Geodecoder            Emplacement :
Geodecoder            4-12 Rue de la Convention
Geodecoder            75015 Paris
Geodecoder            15Ã¨me Arrondissement Paris
Geodecoder            Paris
```

Figure 13–10 Résultat de la fonction de conversion des coordonnées en emplacement

Pour toutes les recherches qui demandent un certain temps de résolution, nous vous recommandons de réaliser les opérations concernées dans des tâches concurrentes (threads) afin de laisser l'interface graphique disponible le temps des échanges de données (pour plus d'informations, référez-vous au chapitre 11).

Création d'activités utilisant Google Maps

La manipulation d'adresses ou de lieux, que ce soit pour un affichage ou une saisie, peut se faire via des affichages texte ou des formulaires mais l'utilisateur est habitué à tout visualiser sur des cartes. C'est ce que nous allons apprendre à réaliser dans cette partie du chapitre.

Pour réaliser des applications utilisant ces vues à base de cartes, nous allons employer, entre autres, les différentes classes suivantes :

- `MapActivity` : la classe de base à étendre pour créer une activité qui contiendra une carte ;
- `MapView` : une vue affichant les cartes sous forme de tuiles ;
- `MapController` : qui permet de contrôler la carte (centrage, niveau de zoom, etc.) ;
- `Overlay` : une sorte de calque pour annoter les cartes.

Étendre la classe MapActivity

La classe `MapActivity` est étroitement liée à la classe `MapView` que nous allons étudier ensuite. Elle permet, grâce à ses différentes méthodes, de prendre en charge les tâches - notamment de connexion réseau - pour simplifier l'utilisation d'une vue orientée carte.

> **RECOMMANDATION** **Mise en garde sur l'utilisation de la classe MapActivity**
>
> Si vous consultez la javadoc concernant cette classe, vous noterez que Google conseille de n'utiliser qu'une activité de type carte à la fois et que de multiples instances peuvent interférer *in unexpected and undesired ways*, ce qui est assez dissuadant !

En pratique, l'emploi de la classe `MapActivity` dans une application nécessite plusieurs étapes.

Son utilisation doit être déclarée dans la section `manifest` du fichier de configuration de l'application, plus précisément à l'intérieur du nœud `application` :

```
<uses-library android:name="com.google.android.maps"/>
```

Pourquoi ? Parce qu'il s'agit d'une API optionnelle. Il pourra en être de même avec d'autres bibliothèques de fonctions que vous voudrez utiliser. Étudiez bien les fonctions que vous manipulez pour être capable de dire si votre application a besoin de ces paramétrages.

Ensuite, puisque les tuiles du plan ainsi que toutes les informations comme le trafic routier sont téléchargées au fur et à mesure (vous devez avoir remarqué qu'à chaque déplacement de la carte, les zones sont affichées progressivement), il faut également prévoir la permission d'accéder aux données Internet comme nous l'avons déjà fait à plusieurs reprises :

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

Reste donc à créer notre nouvelle activité qui étendra la classe `MapActivity`. Cette activité contiendra une vue de type plan et un contrôleur de carte que nous manipulerons par la suite. Le squelette de l'application aura donc cette allure.

Code 13-5 : Squelette d'une activité qui manipule des cartes

```
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;
import android.os.Bundle;

public class MyMapActivity extends MapActivity { (1)
   private MapView mapView; (2)
   private MapController mapController; (3)

   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.map); (4)
   }

   protected boolean isRouteDisplayed() { (5)
      return false;
   }
}
```

Comme nous le disions, le plus important est d'étendre la classe `MapActivity` (1). Ceci nous oblige à donner corps à la méthode `isRouteDisplayed` (5), qui est abstraite dans la classe `MapActivity`. Cette méthode renvoie `true` ou `false`, pour savoir si l'on fournit aussi des instructions de navigation.

Le squelette comporte des références à des instances de `MapView` (2) et de `MapController` (3) que nous manipulerons par la suite pour configurer la carte et son affichage.

La mise en forme fixée (4) est abordée dans l'utilisation de la classe `MapView` puisque, comme nous le disions il y a quelques instants, une activité de type carte est composée d'une vue sur la carte.

Manipuler la classe MapView

La classe `MapView` gère l'affichage de la carte. Pour l'employer dans votre application, voici la présentation qui complète l'activité entamée dans la partie précédente.

Code 13-6 : Fichier de présentation pour une activité qui gère des cartes

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
<com.google.android.maps.MapView (1)
   android:id="@+id/mapView" (2)
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:enabled="true"
   android:clickable="true"
   android:apiKey="cle_maps_obtenue" (3)
/>
</LinearLayout>
```

On retrouve sans surprise la vue (1) avec son identifiant (2). On constate la présence d'un paramètre supplémentaire essentiel : la clé Google Maps (3). Elle peut soit être saisie directement dans la description de la vue, comme le montre le code précédent, soit donnée sous forme de ressource.

```
android:apiKey="@string/cleMaps"
```

Il faudra alors ajouter une ressource chaîne de caractères qui permet de mieux organiser votre application :

```
<resources>
   ...
   <string name="cleMaps">XXXXXXXXXX</string>
   ...
</resources>
```

Pour obtenir l'instance de la vue et réaliser des opérations, on utilisera la ligne de code suivante :

```
mapView = (MapView)findViewById(R.id.mapView);
```

À partir de cette instance, plusieurs configurations sont possibles selon le type de vue que vous voulez afficher (`Satellite`, `StreetView`, `Traffic`) :

```
mapView.setSatellite(true/false);
mapView.setStreetView(true/false);
mapView.setTraffic(true/false);
```

Ensuite, il faut regarder plus précisément la documentation de la classe `MapView` pour la personnaliser. On trouvera notamment comment ajouter les contrôles de zoom.

```
mapView.setBuiltInZoomControls(true);
```

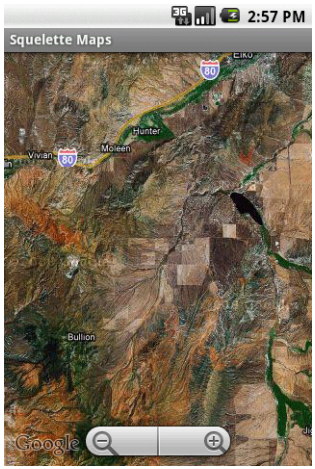


Figure 13–11
Rendu du squelette de l'application

Manipuler la classe MapController

On utilise la classe `MapController` pour manipuler la carte en termes de position (recentrer) et de zoom. La récupération de l'instance de `MapController` associée à une vue de type carte se fait à l'aide d'un accesseur.

```
mapController = mapView.getController();
```

Déplacer la carte

On utilise la classe `GeoPoint` pour déterminer un endroit sur une carte. La classe `GeoPoint` se construit à l'aide de valeurs en micro degrés, à savoir `degrés*1E6` (10 puissance 6). Ensuite, la méthode `setCenter` de l'instance de `MapController` liée à la vue permet de centrer la carte sur le point voulu.

```
Double latitude = 48.8453972*1E6;
Double longitude = 2.2783926*1E6;
GeoPoint point = new GeoPoint(latitude.intValue(), longitude.intValue());

mapController.setCenter(point);
```

On obtient la carte centrée sur Paris, mais avec le niveau de zoom par défaut. Vous noterez la présence de la méthode `animateTo` (et de son pendant `stopAnimation`)

dans la classe `MapController` qui centre la carte sur le point donné en paramètre en réalisant une animation de transition.

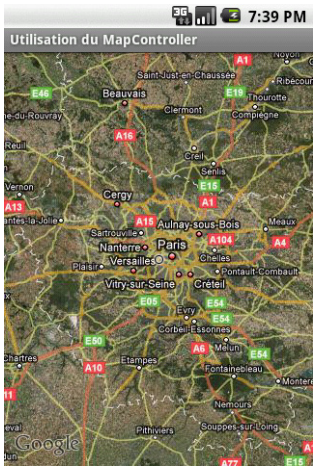


Figure 13–12
Résultat du code pour centrer une carte sur un point

Niveaux de zoom

Comme vous avez pu constater dans l'exemple précédent, il peut être nécessaire de changer le niveau de zoom pour montrer une partie de la carte grâce à la méthode `setZoom` de la classe `MapController`.

```
mapController.setZoom(18);
```

Si vous utilisez Google Maps, vous remarquerez que le fond cartographique se charge selon un découpage (dit *tuilage*) comprenant un ensemble de morceaux (tuiles) visibles à l'écran. Quand vous vous déplacez sur la carte, les tuiles sont chargées au fur et à mesure. Ce tuilage est stocké, pour chaque niveau de précision (et donc de zoom pour nous), sur les serveurs de Google.

Quand on change le niveau de précision (de zoom), on charge un ensemble de tuiles différent. Suivant la zone ciblée, les images sont disponibles à des précisions variées. Le niveau de zoom est réglable de 1 (plus petit niveau) à 21 inclus, sachant qu'il n'existe pas toujours de tuiles du lieu choisi aux niveaux de zoom les plus élevés (la zone n'a pas été photographiée avec cette précision ou elle fait partie d'un ensemble sensible qui doit être masqué par exemple).

La documentation précise également, pour donner un ordre d'idée, qu'au niveau de zoom 1, l'Équateur terrestre mesure 256 pixels de long et que chaque niveau de zoom successif grossit d'un facteur 2.

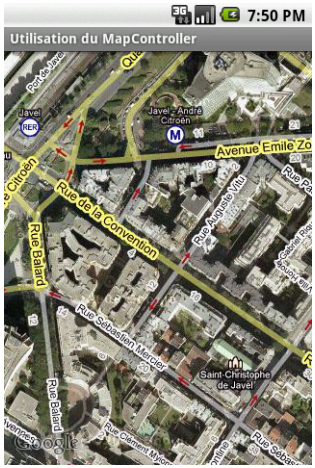


Figure 13–13
Résultat d'un zoom de niveau 18 sur le point utilisé précédemment

La classe `MapController` donne accès à plusieurs méthodes pour manipuler le niveau de zoom. Citons par exemple `zoomIn` et `zoomOut` qui permettent de zoomer/dézoomer d'un niveau avec une animation.

Placer des données sur une carte

Les calques permettent d'ajouter des formes, des images ou du texte sur une vue de type carte (`MapView`). Il est possible de superposer plusieurs calques qui masqueront potentiellement des zones correspondant à des couches plus anciennes.

Créer et utiliser des calques

Pour créer un calque, il faut créer une classe qui hérite de la classe `Overlay`. On dispose alors d'une sorte de canevas pour dessiner. Les instructions de dessin sont placées dans la méthode `draw` qui est à redéfinir. Pour réagir aux actions utilisateur sur ces annotations, il faut redéfinir la méthode `onTap`.

Nous nous retrouvons donc avec un squelette de calque construit ainsi.

Code 13-7 : Squelette d'utilisation des calques

```
package com.eyrolles.android.geolocalisation;

import android.graphics.Canvas;

import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapView;
```

```
import com.google.android.maps.Overlay;

public class MonNouveauCalque extends Overlay {

   public void draw(Canvas canvas, MapView mapView, boolean shadow) {
   }

   public boolean onTap(GeoPoint point, MapView mapView) {
      return false;
   }
}
```

L'utilisation de ces deux méthodes et la signification de leurs signatures seront abordées dans les sections suivantes « Dessiner sur un calque » et « Réagir à une sélection ».

Ajouter et retirer des calques de la vue

Chaque `MapView` contient la liste de ses calques que l'on récupère sous forme de liste d'instances d'`Overlay`.

```
List<Overlay> listeDesCalques = mapView.getOverlays();
```

L'ajout ou la suppression de calques se fait de façon transparente et sans préoccupation de threading ou de synchronisation étant donné que la liste a été passée à la méthode `Collections.synchronizedList(java.util.List)` avant de nous être retournée.

```
MonNouveauCalque nouveauCalque = new MonNouveauCalque();
// Par exemple, pour l'ajout d'un calque.
listeDesCalques.add(nouveauCalque);
```

Tous les calques de la liste seront dessinés par ordre croissant et recevront des événements progressivement en ordre inverse tant qu'un calque ne les traitera pas.

Les changements seront visibles au prochain dessin de la vue, il est donc recommandé d'invoquer la méthode `postInvalidate` sur la vue.

```
mapView.postInvalidate();
```

À noter que la javadoc recommande de réaliser les itérations sur la liste dans un bloc synchronisé sur la liste.

Dessiner sur un calque

Comme vous l'avez vu précédemment, le dessin sur un calque est possible en redéfinissant la méthode `draw` du calque, un peu comme si nous redéfinissions la méthode `paint` d'un composant Swing pour ceux qui connaissent cette API de Java SE.

Reprenons donc la méthode `public void draw(Canvas canvas, MapView mapView, boolean shadow)` :

- la référence de type `Canvas` pour dessiner. Cet objet correspond à la surface de dessin visible ;
- la référence de type `MapView` est la vue qui affiche la carte concernée par notre dessin ;
- le booléen `shadow` indique dans quel appel nous nous trouvons. En effet, cette méthode est appelée deux fois pour permettre, quand `shadow` vaut `true`, de dessiner des ombres aux formes créées.

Pour savoir où dessiner sur l'écran visible, il faut convertir les coordonnées géographiques latitude/longitude en coordonnées sur le `Canvas`. Pour réaliser cette conversion (dans les deux sens), nous disposons de la classe `Projection` qui convertit un `GeoPoint` en `Point` et vice versa.

La classe `Projection` est obtenue grâce à la méthode d'instance `getProjection` de la classe `MapView`. Cette classe obtenue dépend de l'état courant de l'affichage, pensez donc à en obtenir une nouvelle instance à chaque dessin.

```
Projection projection = mapView.getProjection();
```

Ensuite, les méthodes d'instance `toPixels` et `fromPixels` de la classe `Projection` font le travail :

```
// Création du point contenant les coordonnées géographiques.
Double latitude = 48.8453972*1E6;
Double longitude = 2.2783926*1E6;
GeoPoint geoPoint = new GeoPoint(latitude.intValue(), longitude.intValue());
// Point contenant le résultat de la conversion en coordonnées écran.
Point pointEcran = new Point();

projection.toPixels(geoPoint, pointEcran);
```

Dessiner avec style

Nous disposons à présent de coordonnées pour dessiner. Il reste à choisir le style du pinceau que nous allons utiliser pour réaliser une forme, et c'est là qu'intervient la classe `Paint`. Elle va nous permettre de déterminer le style et la couleur des formes et des textes que nous voulons dessiner sur le calque. Créons une instance de cette classe :

```
Paint paint = new Paint();
```

Consultez la javadoc de la classe pour découvrir ses différents paramétrages (texte en gras, souligné, sélection d'une couleur et surtout anti-aliasing pour diminuer l'effet d'escalier des traits) :

```
paint.setARGB(255, 255, 255, 0);
```

La boîte à outils du dessinateur

Une fois les coordonnées de la forme et le style choisis, on utilise toutes les méthodes de l'instance de `Canvas` pour réaliser une figure. On retrouve alors toutes les formes de dessin dans les méthodes disponibles : `drawLine`, `drawCircle` et `drawRect`. Prenons l'exemple d'un texte que l'on veut écrire. La méthode associée est `drawText`.

```
canvas.drawText("C'est ici !", pointEcran.x, pointEcran.y, paint);
```

Là encore, consultez la documentation de la classe pour trouver l'outil adapté à vos besoins sachant que toutes les méthodes de dessin prennent en paramètre le style de la forme via une instance de la classe `Paint` vue plus haut.

Réagir à une sélection

Pour réagir aux clics de l'utilisateur, il faut redéfinir la méthode `public boolean onTap(GeoPoint point, MapView mapView)` de notre squelette de calque :

- l'instance de `GeoPoint` contient les coordonnées du point sélectionné par l'utilisateur ;
- l'instance de `MapView` correspond à la source de l'événement ;
- retourner `true` ou `false` selon que l'on traite l'événement ou pas.

Exemple de prise en compte de l'interaction utilisateur dans un calque :

```
protected boolean onTap(int i) {
   Toast.makeText(...).show();
   return true;
}
```

Exemple de calque

Voici un exemple de calque qui dessine un cercle autour d'un point défini par une latitude et une longitude :

Code 13-8 : Dessin d'un cercle sur un calque

```
import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.Point;
```

```
import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapView;
import com.google.android.maps.Overlay;
import com.google.android.maps.Projection;

public class MonNouveauCalque extends Overlay {

   public void draw(Canvas canvas, MapView mapView, boolean shadow) {

      // Dessin si l'on n'est pas dans l'appel qui correspond à l'ombre.
      if (!shadow) {
         // Le point (latitude/longitude) à convertir en position écran.
         Double lat = 48.8453972*1E6;
         Double lng = 2.2783926*1E6;
         GeoPoint geoPoint = new GeoPoint(lat.intValue(), lng.intValue());

         // Conversion du point.
         Point pointEcran = new Point();
         Projection projection = mapView.getProjection();
         projection.toPixels(geoPoint, pointEcran);

         // On choisit uniquement une couleur (vert).
         Paint paint = new Paint();
         paint.setARGB(255, 0, 255, 0);

         // Dessin du cercle.
         int rayonDuCercle = 10;
         canvas.drawCircle(pointEcran.x, pointEcran.y, rayonDuCercle, paint);
      }
   }
}
```

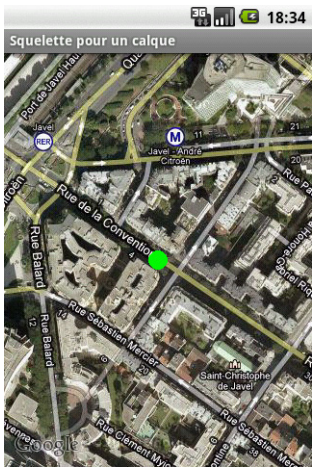


Figure 13–14
Affichage du calque sur la carte

Calques et objets particuliers

Calque de type MyLocationOverlay

L'API Android propose un calque spécifique qui peut répondre à deux problématiques standards :

- montrer où vous êtes sur la carte à partir de coordonnées GPS ou d'un autre fournisseur de position ;
- montrer votre orientation grâce à la boussole électronique si elle est disponible sur votre matériel.

La première action à réaliser nous amène à créer une instance de la classe `MyLocationOverlay` et à l'ajouter aux calques de la vue :

```
MyLocationOverlay locationOverlay = new MyLocationOverlay(this, mapView);
mapView.getOverlays().add(locationOverlay);
```

Pour activer les fonctions spécifiques à ce calque, utilisez les méthodes `enableCompass` et `enableMyLocation` sur l'instance que vous souhaitez impacter.

Ensuite, pour optimiser la durée de vie de la batterie, vous pouvez activer et désactiver les fonctions concernées au moment opportun :

```
public void onResume() {
   super.onResume();
   locationOverlay.enableCompass();
}

public void onPause() {
   super.onPause();
   locationOverlay.disableCompass();
}
```

Par exemple, mettre à jour les informations - ici la boussole - n'a pas de sens quand l'application est en pause.

Utiliser les classes ItemizedOverlay et OverlayItem

Comme son nom l'indique, la classe `ItemizedOverlay` affiche sur une carte une liste de points d'intérêts. Ces points sont des instances de la classe `OverlayItem`. Voici les différentes étapes de la manipulation de ce type de calque :

1. En premier lieu, une classe doit hériter de `ItemizedOverlay<OverlayItem>`. Nous créerons également un constructeur qui prendra en paramètre un objet de type `Drawable`. Cet objet est un marqueur qui mettra en valeur les points que nous souhaitons montrer (comme les bulles rouges de Google Maps).

2 Dans le constructeur, préparer tous les éléments `OverlayItem` et appeler la méthode `populate` une fois qu'ils sont prêts.
3 Implémenter la méthode `size` qui renvoie le nombre d'éléments du calque.
4 Redéfinir la méthode `createItem` pour renvoyer les éléments `OverlayItem` un par un en fonction de l'index donné.
5 Instancier la classe `ItemizedOverlay` et l'ajouter aux calques de la vue.

Suivons ces étapes en réalisant une carte des lieux à visiter à Paris. Créons un marqueur pour pointer ces lieux sur la carte. Pour cela, nous avons besoin d'une simple image créée avec n'importe quel éditeur graphique ou téléchargée sur Internet et pourquoi pas sur Google Maps directement.

Figure 13–15
Marqueur qui désignera les points à visiter sur la carte

Côté présentation, il s'agira d'une activité ne contenant qu'une carte, soit une mise en page épurée avec uniquement un élément `MapView`.

Code 13-9 : Fichier de mise en page pour l'utilisation de la classe OverlayItem

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">
   <com.google.android.maps.MapView
      android:id="@+id/mapView"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:enabled="true"
      android:clickable="true"
      android:apiKey="xxxxxxxxxxxxxx" /> (1)
</LinearLayout>
```

N'oubliez pas de renseigner votre clé directement dans ce fichier (1) ou via une ressource de type chaîne de caractères.

Passons, à présent, à la construction de l'activité.

Code 13-10 : Activité du programme Sites à visiter

```
import android.graphics.drawable.Drawable;
import android.os.Bundle;
import android.view.KeyEvent;

import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;

public class SitesAVisiter extends MapActivity { (1)

   private MapView mapView = null;
   private MapController mapController = null;

   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.map); (2)

      mapView =(MapView)findViewById(R.id.mapView); (3)
      mapController = mapView.getController();

      mapController.setCenter(GeoTools.makeGeoPoint(48.855279,2.287939)); (4)
      mapController.setZoom(15);

      Drawable marqueur =
         getResources().getDrawable(R.drawable.maps_position_marker);
      marqueur.setBounds(0, 0, marqueur.getIntrinsicWidth(),
            marqueur.getIntrinsicHeight()); (5)

      mapView.getOverlays().add(new CalqueSites(marqueur, this)); (6)
   }

   /**
    * Gestion des touches pressées sur la carte.
    */
   public boolean onKeyDown(int keyCode, KeyEvent event) { (7)
      if (keyCode == KeyEvent.KEYCODE_S) {
         // Touche S pressée.
         mapView.setSatellite(!mapView.isSatellite());
         return(true);
      } else if (keyCode == KeyEvent.KEYCODE_A) {
         // Touche A pressée.
         mapController.zoomIn();
         return(true);
      } else if (keyCode == KeyEvent.KEYCODE_Z) {
         // Touche Z pressée.
         mapView.displayZoomControls(true);
         mapController.zoomOut();
```

```
            return(true);
        }

        return(super.onKeyDown(keyCode, event));
    }

    /**
     * L'application donne-t-elle des indications routières ?
     */
    protected boolean isRouteDisplayed() {
        return(false); // Non
    }
}
```

Passons rapidement les premières lignes où nous étendons la classe `MapActivity` (1), fixons la mise en page (2) et récupérons les instances de la vue et du contrôleur (3).

Pour l'exemple, nous centrons la carte (4) et choisissons le niveau de zoom pour voir les sites à visiter que nous avons sélectionnés. À noter l'utilisation de la méthode de classe `makeGeoPoint` qui sera utilisée plusieurs fois par la suite. Cette méthode permet de construire une instance de `GeoPoint` à partir des coordonnées latitude et longitude données sous forme de doubles. Le code de cette méthode est disponible plus loin dans l'extrait de code intitulé « Boîte à outils ».

Nous créons ensuite une instance du marqueur (5). Celui-ci est construit avec l'image présentée un peu plus haut. On passe l'objet instancié en paramètre du nouveau calque qui est directement ajouté à la liste des calques de la vue (6). Le code qui sert à la fabrication de ce marqueur est l'extrait de code suivant.

Vous noterez la possibilité d'intercepter les pressions sur les touches du téléphone (7), fonctionnalité que nous n'avons pas encore vue. Il faut redéfinir la méthode `onKeyDown` ou `onKeyUp` et tester l'égalité entre le code touche reçu et une constante.

Voici maintenant le code du calque à proprement parler. Il est chargé d'afficher les différents sites (quatre sélectionnés et écrits en « dur » dans le code). Si l'utilisateur sélectionne l'un des éléments, un descriptif s'affichera.

Code 13-11 : Calque de type ItemizedOverlay pour l'application Sites à visiter

```
import java.util.ArrayList;
import java.util.List;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.drawable.Drawable;
import android.widget.Toast;
```

```
import com.google.android.maps.ItemizedOverlay;
import com.google.android.maps.MapView;
import com.google.android.maps.OverlayItem;

public class CalqueSites extends ItemizedOverlay<OverlayItem> { 1
   private List<OverlayItem> items = new ArrayList<OverlayItem>(); 2
   private Drawable marker = null;
   private Context context = null;

   public CalqueSites(Drawable marker, Context context) {
      super(marker);
      this.marker = marker; 3
      this.context = context;

      items.add(new OverlayItem(GeoTools.makeGeoPoint(48.858304, 2.294360),
            "Tour Eiffel",
            "Tour Eiffel - Parc du Champ-de-Mars 75007 Paris")); 4
      items.add(new OverlayItem(GeoTools.makeGeoPoint(48.852544, 2.278149),
                  "Maison de Radio France",
                  "Maison de la Radio - 116 avenue du Président Kennedy 75016
Paris"));
      items.add(new OverlayItem(GeoTools.makeGeoPoint(48.850017, 2.279704),
            "Statue de la Liberté",
            "Statue de la Liberté - Pont de Grenelle 75015 Paris"));
      items.add(new OverlayItem(GeoTools.makeGeoPoint(48.862236, 2.288341),
            "Trocadéro",
            "Parvis des droits de l'homme 75016 Paris"));
      populate(); 5
   }

   protected OverlayItem createItem(int i) { 6
      return (items.get(i));
   }

   public void draw(Canvas canvas, MapView mapView, boolean shadow) { 7
      super.draw(canvas, mapView, shadow);
      boundCenterBottom(marker);
   }

   protected boolean onTap(int i) { 8
      Toast.makeText(context, items.get(i).getSnippet(),
         Toast.LENGTH_SHORT).show();
      return (true);
   }

   public int size() { 9
      return (items.size());
   }
}
```

Il faut tout d'abord utiliser le mécanisme d'héritage (1). Ensuite, on crée le constructeur (3) qui prend en paramètres le marqueur (stocké pour être réutilisé) et le contexte (également stocké, notamment pour l'affichage de messages après sélection des éléments).

On crée une collection d'`OverlayItem` (2) qui va contenir tous les éléments à afficher avec un marqueur sur la carte. Peu après, on ajoute un par un les éléments à cette collection (4) en utilisant de nouveau la méthode `makeGeoPoint` détaillée dans le code suivant. Parallèlement, on implémente les méthodes `createItem` (6) et `size` (9). La première renvoie un élément situé à un index donné dans la collection, la seconde donne la taille de la collection. Ces deux méthodes permettent au calque de parcourir les éléments et de les afficher avec l'appel de la méthode `populate` (5).

En redéfinissant la méthode `draw` (7), on délègue une grosse partie du travail de dessin du marqueur au calque, mais on l'aide tout de même à déterminer l'emplacement du marqueur et surtout le bas afin que l'ombre soit correctement dessinée.

L'affichage des informations complémentaires sur chaque élément est réalisé via la méthode `onTap`. On affiche sous forme de *toast* l'élément concerné en rapprochant l'index passé en paramètre de l'élément de rang équivalent dans la collection globale (2).

Enfin, le dernier extrait de code est pour la « boîte à outils », sous forme de fichier utilitaire (souvent appelé *helper*) qui ne contient qu'une méthode pour la transformation de coordonnées sous forme de doubles en une instance de `GeoPoint`.

Code 13-12 : Boîte à outils pour les méthodes géographiques

```
import com.google.android.maps.GeoPoint;

public class GeoTools {

   public static GeoPoint makeGeoPoint(double latitudeParam,
      double longitudeParam) {

      Double dLatitude = new Double(latitudeParam * 1E6);
      Double dLongitude = new Double(longitudeParam * 1E6);

      return new GeoPoint(dLatitude.intValue(), dLongitude.intValue());
   }
}
```

Figure 13–16
Résultat de l'application Sites à visiter

En résumé

À travers les différents ateliers décrits dans ce chapitre, vous avez découvert des fonctionnalités de géolocalisation à la fois simples à utiliser et très puissantes.

En quelques lignes de code, votre application est capable de situer l'utilisateur ou d'obtenir une adresse correspondant à des coordonnées. En peu de temps, vous pouvez mettre en place des alertes pour être averti quand on entre ou on sort d'une zone ou utiliser des calques pour dessiner sur une carte.

14

Ressources matérielles : Wi-Fi, Bluetooth, capteurs et accéléromètre

Les dernières générations de téléphones tentent d'embarquer de plus en plus de fonctionnalités matérielles : GPS, boussole, gyroscope, Wi-Fi, Bluetooth, capteur magnétique, capteur de température, etc. Bien évidemment, en tant que développeur, vous comprenez immédiatement que ce sont autant d'opportunités de créer des applications toujours plus immersives, communicantes et interactives !

Des appareils de plus en plus divers fonctionnent avec la plate-forme Android. Son caractère gratuit et ouvert et sa relative simplicité d'emploi en font le système idéal pour les fabricants de toutes marques. Néanmoins, cette ouverture comporte certains risques notamment en termes de compatibilité entre les différents modèles et versions. Cette considération est à prendre au sérieux par les développeurs et c'est encore plus vrai quand on exploite les ressources matérielles dans son application. Vous devrez donc vérifier vous-même les ressources disponibles sur le téléphone de l'utilisateur et réagir en conséquence.

Un développeur doit ainsi bien connaître son environnement et pour nous aider dans cette tâche, nous allons faire le point sur les ressources matérielles disponibles.

Ce chapitre est l'occasion de vous présenter les fonctionnalités matérielles suivantes, exploitables sur la plate-forme Android : Wi-Fi, Bluetooth et diverses capteurs. Chacune de ces possibilités représente une opportunité certaine pour créer des applications innovantes !

Les ressources disponibles sur un appareil Android

La plate-forme Android gère de façon native bon nombre de ressources matérielles telles que le clavier et le *trackball* qui ne nécessitent pas vraiment d'adaptation lorsque ceux-ci ne sont pas disponibles sur le terminal. Par exemple, le clavier physique sera remplacé par un clavier virtuel sans avoir besoin de reprogrammer l'interface utilisateur.

Vous trouverez ci-dessous la liste des principales ressources utilisables depuis Android.

- Affichage :
  - appareil photo
  - 3D
- Entrées :
  - clavier
  - tactile
  - boule de commande (*trackball*)
- Son :
  - haut parleur / microphone
  - reconnaissance vocale
  - vibration
- Réseau :
  - 3G
  - Wi-Fi
  - Bluetooth
  - GPS
- Capteurs :
  - accéléromètre
  - orientation
  - champs magnétique
  - température

Gérer le réseau Wi-Fi

Le Wi-Fi est une fonctionnalité importante car elle offre aux appareils qui ne sont pas dotés de fonctions de téléphonie l'accès à l'Internet.

Accéder à un réseau Wi-Fi

Pour manipuler les données relatives aux réseaux Wi-Fi, vous devez faire appel à l'objet `WifiManager`. Cet objet sert d'interface entre le service système qui gère le Wi-Fi et votre application. Il permet d'effectuer les opérations suivantes :

- fournir des informations sur la connexion actuelle ;
- renvoyer la liste des points d'accès à portée ;
- manipuler les configurations enregistrées par l'utilisateur ;
- allumer ou éteindre la connexion Wi-Fi.

Instancier un objet `WifiManager` se fait de la même manière que n'importe quel autre service en utilisant la fonction `getSystemService` de l'objet `Activity` comme indiqué ci-dessous :

Code 14-1 : Instancier l'objet WifiManager

```
WifiManager wifi = (WifiManager)getSystemService(Context.WIFI_SERVICE);
```

Voyons à présent comment tirer profit des nombreuses fonctionnalités de ce service.

Activer et désactiver le Wi-Fi

Le premier point à aborder avec le Wi-Fi, c'est bien évidement de pouvoir obtenir son état (activé ou non) et au besoin de le modifier.

Code 14-2 : Obtenir l'état du Wi-Fi

```
private boolean estWifiActif(){
   WifiManager wifiManager = (WifiManager)getSystemService(Context.WIFI_SERVICE);
   if (!wifiManager .isWifiEnabled()){
      if (wifiManager.getWifiState() != WifiManager.WIFI_STATE_ENABLING) {
         return false;
      }
   }
   return true;
}
```

Avec cette fonction nous savons si l'adaptateur Wi-Fi est en train de se connecter ou l'est déjà. Voyons à présent comment l'activer ou le désactiver :

Code 14-3 : Changer l'état du Wi-Fi

```
WifiManager w = (WifiManager)getSystemService(Context.WIFI_SERVICE);
if(wifi.isWifiEnabled()){
   wifi.setWifiEnabled(false);
}else{
   wifi.setWifiEnabled(true);
}
```

Le Wi-Fi peut être activé mais ne pas être le réseau actif pour votre matériel, c'est-à-dire le réseau utilisé par Android à cet instant si d'autres réseaux sont disponibles (3G, EDGE, etc.). Pour obtenir des informations sur le réseau actuellement en cours d'utilisation par Android, nous utiliserons le service de gestion des connexions.

Code 14-4 : Savoir quel est le réseau actif

```
private void getNetworkInfo(){
   ConnectivityManager m =
      ConnectivityManager)getSystemService(Context.CONNECTIVITY_SERVICE);
   NetworkInfo info = m.getActiveNetworkInfo();
   if(info!=null){
      ((TextView)findViewById(R.id.txt_type)).setText(info.getTypeName());
   }else{
      ((TextView)findViewById(R.id.txt_type)).setText("Pas de connexion");
   }
}
```

Le système Android choisit lui-même l'ordre dans lequel il utilisera les connexions. Il peut alors privilégier le Wi-Fi plutôt que le réseau mobile ou l'inverse. Vous pouvez agir sur cette préférence au travers des options du téléphone ou bien via l'utilisation des APIs dans votre application.

Voyons ci-dessous comment faire pour récupérer cette information :

Code 14-5 : Obtenir l'ordre de préférence des réseaux

```
private void getNetworkPreference(){
   ConnectivityManager m =
         (ConnectivityManager)getSystemService(Context.CONNECTIVITY_SERVICE);
   int pref = m.getNetworkPreference();
   ((RadioButton)findViewById(R.id.radio_wifi))
      .setChecked((pref == ConnectivityManager.TYPE_WIFI));
   ((RadioButton)findViewById(R.id.radio_mobile))
      .setChecked((pref == ConnectivityManager.TYPE_MOBILE));
}
```

Pour modifier cette préférence, il faudra au préalable avoir la permission `WRITE_SETTINGS` ou `WRITE_SECURE_SETTINGS`.

> **À SAVOIR Permissions pour modifier des paramètres système**
>
> La permission `WRITE_SETTINGS` permet de lire et d'écrire les paramètres système « non sécurisés ». Par paramètre sécurisé, Android entend tous les paramètres qui doivent être explicitement modifiés par une interface système et non directement par les applications. Pour ces paramètres, par exemple pour changer le réseau actif, vous devrez être autorisé à effectuer ces changements en ayant la permission `WRITE_SECURE_SETTINGS`.

Les valeurs que vous pouvez spécifier pour cette préférence sont celles de la classe `ConnectivityManager` (`TYPE_MOBILE` ou `TYPE_WIFI`).

Code 14-6 : Changer la préférence des réseaux

```
private void setNetworkPreference(int preference){
   // Besoin de android.permission.WRITE_SETTINGS
   // Besoin de android.permission.WRITE_SECURE_SETTINGS
   if(isPermission(permission.WRITE_SECURE_SETTINGS) == 0){
      ConnectivityManager m =
   (ConnectivityManager)getSystemService(Context.CONNECTIVITY_SERVICE);
      m.setNetworkPreference(preference);
   }else{
      Toast.makeText(this,
         "Vous n'avez pas la permisison de modifier ce paramètre"
         , Toast.LENGTH_SHORT).show();
   }
}

private int isPermission(String permission){
   // Renvoi 0 si ok sinon -1
   int pid = android.os.Process.myPid();
   int uid = android.os.Process.myUid();
   return checkPermission(permission,pid,uid);
}
```

Lorsque la connexion Wi-Fi est active, nous avons la possibilité d'obtenir les informations concernant entre autres la force du signal, ainsi que le nom et la vitesse du réseau. Pour ce faire, nous nous appuierons sur les méthodes `WifiManager.calculateSignalLevel`, `WifiInfo.getSSID` et `WifiInfo.getLinkSpeed`.

Code 14-7 : Obtenir la force du signal et la vitesse de la connexion

```
if (info.getBSSID() != null) {
   int force = WifiManager.calculateSignalLevel(info.getRssi(), 5);
   Toast.makeText(this, "Connecté à " + info.getSSID() +
   " à " + info.getLinkSpeed() + WifiInfo.LINK_SPEED_UNITS +
   " avec une force " + strength + "/5", Toast.LENGTH_LONG).show();
}
```

Au-delà de ces fonctions de base, l'API propose des fonctionnalités avancées pour gérer les points d'accès et la sauvegarde des préférences.

Gérer les points d'accès

Avant de pouvoir rejoindre un réseau Wi-Fi, vous devez d'abord détecter les points d'accès vous environnant. Pour demander à l'appareil quels sont les points d'accès actifs, vous devez utiliser la méthode `startScan`. Le retour de cette méthode est immédiat : elle exécute une recherche en tâche de fond et les résultats seront connus plus tard grâce à la réception d'un événement asynchrone. La valeur booléenne retournée par la méthode `startScan` indique uniquement si la détection des points d'accès a été initialisée.

Une fois la détection des réseaux Wi-Fi lancée, vous pourrez récupérer le résultat en appelant la méthode `getScanResults` qui vous renverra une liste de résultats contenant les points d'accès disponibles.

Code 14-8 : Détection des réseaux Wi-Fi

```
public void detectionPointAcces()
{
   if(isPermission("android.permission.ACCESS_WIFI_STATE",
ctx.getPackageName()) == 0)
   {
     // Nous créons un filtre d'Intent afin de l'associer à un BroadcastReceiver
qui recevra la notification que les résultats de la détection de points d'accès
sont disponibles.
      IntentFilter intf= new IntentFilter();
      intf.addAction(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION);

     // Nous enregistrons un composant BroadcatReceiver pour être prévenu de la
disponibilité des informations
      registerReceiver(
         new BroadcastReceiver() {
            @Override
```

```
            public void onReceive(Context context, Intent intent) {
               WifiManager wifiManager = (WifiManager)
                                    getSystemService (Context.WIFI_SERVICE);

               // Récupération de la liste des points d'accès
               List<ScanResult> hotspot= wifiManager.getScanResults();
               System.out.println("Nombre de point d'accès trouvés : "
                                    + hotspot.size());
            }
         }
      , intf);
   }
}
```

Pour configurer ou récupérer d'autres informations, vous disposerez des autres méthodes du `WifiManager`. La méthode `getDhcpinfo` permet de récupérer la dernière adresse renvoyée par le DHCP du point d'accès.

Les méthodes `addNetwork`, `removeNetwork` et `updateNetwork` vous permettront respectivement d'ajouter un réseau, de le supprimer et de mettre à jour la configuration du réseau en utilisant un objet `WifiConfiguration` comme paramètre.

La gestion et l'utilisation d'un réseau Wi-Fi est grandement simplifiée par les APIs Android. La classe `WifiManager` sera d'ailleurs votre point d'entrée pour gérer tous les réseaux Wi-Fi. Si vous créez des applications qui nécessitent une connexion à Internet, gérez spécifiquement l'accès Wi-Fi dans votre application. Vous pourrez ainsi prévenir l'utilisateur, lorsque votre application nécessite une connexion gourmande (pour laquelle le débit du Wi-Fi risque d'être insuffisant), que sa connexion n'est pas suffisante et l'avertir que les performances seront dégradées ou que la qualité du rendu s'effectuera en basse résolution.

Gérer le Bluetooth

Le Bluetooth est une technologie de communication radio de courte distance, créée pour simplifier les connexions entre appareils. Ce système a été conçu dans le but de remplacer les câbles des imprimantes, des kits « main libre », des souris/claviers, téléphones portables/PDA, etc.

La plate-forme Android supporte le Bluetooth afin de permettre à des appareils d'échanger des données entre elles. Pour pouvoir profiter de ce système de connexion, Android apporte un ensemble d'API que nous allons détailler dans cette partie.

L'espace de nom `android.bluetooth` contient les classes suivantes :

- `BluetoothAdapter` : similaire au `WifiManager`, il s'agit de la classe centrale pour les interactions avec le Bluetooth, elle est utilisée pour récupérer les appareils environnants, effectuer toutes les requêtes et communiquer avec les autres appareils ;
- `BluetoothDevice` : représente un appareil Bluetooth distant, et est utilisé pour récupérer les informations sur l'appareil et créer une connexion ;
- `BluetoothSocket` et `BluetoothServerSocket` : représentent les points de connexion permettant aux appareils d'échanger des données entre eux.

Ces API supportent la connexion à un appareil dans une configuration point à point (l'appareil est connecté à un seul autre appareil) ou alors en multi-points (l'appareil peut être connecté à plusieurs appareils simultanément).

Les permissions

À l'instar de toutes les fonctionnalités de communication, pour qu'une application puisse utiliser les capacités Bluetooth, vous devez avoir défini la permission `android.permission.BLUETOOTH` dans le fichier de configuration de l'application. Cette permission autorise votre application à accepter une connexion entrante, à demander une connexion distante et à transférer des données.

Code 14-9 : Utilisation de la permission pour le Bluetooth

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

Pour pouvoir activer la recherche des autres appareils Bluetooth ou pour manipuler les paramètres Bluetooth du téléphone, vous devrez spécifier une autre permission nommée `android.permission.BLUETOOTH_ADMIN.`

Code 14-10 : Utilisation de la permission pour modifier les paramètres Bluetooth

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

Préparer votre application

Avant de pouvoir utiliser le Bluetooth et de pouvoir échanger avec d'autres appareils, vous devez vérifier que l'appareil de l'utilisateur possède bien la fonctionnalité du Bluetooth et si celle-ci est activée.

L'objet `BluetoothAdapter` est incontournable pour l'utilisation du Bluetooth sur la plate-forme Android. La première chose que vous aurez donc à réaliser dans votre code sera de récupérer une instance de cette classe.

Vous réaliserez cette opération via la méthode statique `getDefaultAdapter` qui vous renverra l'objet en question et qui vous servira à vérifier l'existence et l'activation du Bluetooth sur le téléphone de l'utilisateur. Si la méthode `BluetoothAdapter.getDefaultAdapter` renvoie un objet `null` alors le téléphone de l'utilisateur ne possède pas de Bluetooth et vous pourrez désactiver toutes les fonctionnalités utilisant ce service de communication.

Si le Bluetooth est disponible sur l'appareil, vous devez également vérifier que celui-ci est activé avant d'essayer de l'utiliser. Pour cela, vous bénéficiez de la méthode `isEnabled` de la classe `BluetoothAdapter`. Si la méthode retourne `false`, alors le Bluetooth n'est pas activé. Vous pouvez demander à l'utilisateur de l'activer en démarrant une activité spécialement conçue à cet effet en spécifiant un `Intent` avec une action `ACTION_REQUEST_ENABLE`.

Code 14-11 : Vérifier l'existence du Bluetooth et son activation

```
private static final int REQUEST_ENABLE_BT = 1;

...

BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
if (mBluetoothAdapter != null) {
   if (!mBluetoothAdapter.isEnabled()) {
      Intent enableBtIntent =
                     new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
      startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
   }
}
else
{
   // Si la méthode renvoie null alors le Bluetooth n'est pas supporté par
l'appareil.
}
```

En fonction de la réponse de l'utilisateur, le Bluetooth sera activé ou non. En fonction de la réponse, la méthode `startActivityForResult` renverra `RESULT_OK` ou `RESULT_CANCELED`.

Rechercher d'autres appareils Bluetooth

Vous pouvez rechercher les appareils Bluetooth disponibles autour de l'utilisateur. Cependant, pour qu'un appareil puisse être découvert, il faut que celui-ci autorise sa découverte. Si un appareil autorise sa découverte, une association sera alors établie et les appareils échangeront leurs informations d'identification (nom de l'appareil, son adresse MAC, etc). Une association signifie que les deux appareils sont conscients de leur existence respective, qu'ils sont capables de s'authentifier l'un l'autre et d'initier une connexion sécurisée entre eux.

> **B.A.-BA** **Adresse MAC**
>
> En réseau informatique, une adresse MAC (*Media Access Control*) est un identifiant physique stocké dans une carte réseau, attribué de façon unique à chaque carte à l'échelle mondiale : cet identifiant est en partie constitué par un champs identifiant le fabricant de la carte et l'autre partie attribuée par le constructeur lui-même.

Une fois l'association établie avec un appareil distant, vous pouvez vous connecter à celui-ci en utilisant son adresse MAC pour transférer des données via un flux RFCOMM. Android obligeant un appareil à être associé avant de pouvoir s'y connecter, tous les paramètres des appareils qui ont été associés sont enregistrés pour un usage ultérieur et ainsi éviter de réaliser systématiquement une détection des appareils.

> **Aller plus loin** **Protocole RFCOMM**
>
> Le protocole RFCOMM émule les paramètres de la ligne série câblée ainsi que le statut d'un port série RS-232. Il est utilisé pour permettre le transfert des données série. Ce protocole est notamment utilisé par les modems, les imprimantes et les ordinateurs. Le Bluetooth utilise ce protocole pour échanger des données.

Activer la découverte par Bluetooth

Si vous souhaitez rendre accessible l'appareil de l'utilisateur lors d'une recherche d'appareils Bluetooth, vous devez demander l'autorisation à l'utilisateur. Pour cela, lancez l'activité adéquate en spécifiant un objet `Intent` associé à l'action `ACTION_REQUEST_DISCOVERABLE` et contenant une donnée supplémentaire représentant le délai maximum pendant lequel l'appareil effectue la recherche (en secondes) :

Code 14-12 : Lancer l'activité d'activation de la découverte du Bluetooth

```
Intent intent = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 120);
startActivity(intent);
```

Récupérer les appareils associés

Au lieu de lancer immédiatement une nouvelle recherche d'appareils Bluetooth, il convient dans un premier temps de rechercher si l'appareil de l'utilisateur ne connaît pas déjà l'appareil ciblé. Pour cela, vous allez demander à la méthode `getBondedDevices` d'effectuer une requête sur les appareils déjà connus.

Code 14-13 : Parcourir les appareils associés

```
Set<BluetoothDevice> appareilsAssocies = bluetoothAdapter.getBondedDevices();

if (appareilsAssocies.size() > 0) {
   // Nous parcourons tous les appareils associés.
   for (BluetoothDevice appareil : appareilsAssocies) {
      String msg = appareil.getName() + " - MAC : " + appareil.getAddress());
      ...
   }
}
```

Avant de pouvoir communiquer avec un appareil, vous devez obtenir l'adresse de l'appareil distant en récupérant les informations des appareils associés.

Rechercher les appareils environnant

Pour lancer une découverte des appareils, utilisez la méthode `startDiscovery` de la classe `BluetoothAdapter`. Cette méthode retourne immédiatement une valeur booléenne indiquant si la découverte s'est correctement lancée. Pour récupérer le résultat de la recherche (qui peut prendre une dizaine de secondes), vous utiliserez un récepteur de diffusion :

Code 14-14 : Rechercher les appareils Bluetooth

```
...

BroadcastReceiver receiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();

        // Si un appareil est découvert
        if (BluetoothDevice.ACTION_FOUND.equals(action)) {
            BluetoothDevice appareil =
                      intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            ...
        }
    }
};
```

```
// Enregistre le composant
IntentFilter filtre = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(receiver, filtre);

...
```

Une fois l'appareil distant identifié, vous pouvez lui demander d'établir une connexion afin d'échanger des données.

Communication entre appareils Bluetooth

Pour établir une connexion entre deux appareils, vous devez implémenter chaque côté de votre logique de communication : le côté serveur et le côté client. Le client et le serveur sont considérés connectés lorsque chaque appareil possède un `BluetoothSocket` sur le même flux RFCOMM. C'est à ce moment que les appareils peuvent transmettre des données dans un sens ou dans l'autre.

Qu'il s'agisse du client ou du serveur, l'objet `BluetoothSocket` sera récupéré de façon différente. Pour le serveur, cet objet lui sera renvoyé lorsqu'il aura accepté la demande de connexion alors que le client l'obtiendra après acceptation de la demande par le serveur et l'établissement du flux RFCOMM. En général votre application pourra se comporter aussi bien en tant que serveur que client.

Créer un serveur

Pour connecter deux appareils, l'un doit faire office de serveur en ouvrant une socket `BluetoothServerSocket` qui écoutera les requêtes entrantes. Si la requête est acceptée alors un objet `BluetoothSocket` connecté à l'autre appareil sera renvoyé.

Pour créer une socket serveur et accepter la connexion de l'appareil distant :

1. Obtenez un objet `BluetoothServerSocket` en appelant la méthode `listenUsingRfcommWithServiceRecord(String, UUID)`. Le premier paramètre est la chaîne représentant le nom de votre service. Le second paramètre est un identifiant unique qui identifie sans ambiguïté votre service. Lors de la demande de connexion, le client enverra également un identifiant unique qui devra être le même que celui spécifié ici pour que la connexion s'établisse.
2. Débutez l'écoute des demandes de connexions en appelant la méthode `accept`. Cette méthode ne retourne pas immédiatement et bloque l'exécution de votre application. La méthode retournera un objet `BluetoothSocket` dès qu'une demande de connexion sera acceptée avec un UUID correct.

Vous pouvez annuler l'écoute des demandes de connexions en appelant la méthode `close` du `BluetoothServerSocket`. Dans tous les cas, appelez cette méthode pour libérer les ressources dès que vous n'aurez plus besoin de cet objet.

Du fait de la nature bloquante de la méthode `accept`, vous devez toujours exécuter cette méthode hors du thread principal de l'interface au risque de bloquer complètement votre application. Dans le code suivant, nous créons un fil d'exécution à part qui se chargera de l'exécution de la demande de connexion afin de ne pas bloquer l'exécution de l'application.

> **À RETENIR UUID**
>
> Un identifiant UUID est un identifiant universel unique (*Universally Unique Identifier*) représenté par une chaine de 128 bits qui identifie l'information de façon unique. L'intérêt de l'UUID est qu'il est suffisamment grand pour que vous puissiez en générer un aléatoirement et avoir peu de chance d'entrer en collision avec le même identifiant.

Code 14-15 : Création d'un serveur Bluetooth attendant les demandes de connexion

```
BluetoothAdapter adaptateurBluetooth = ...

...

class AcceptThread extends Thread {

   // La socket qui sera utilisée pour le transfert de données, une fois que
   // la connexion aura été acceptée.
   private BluetoothSocket socket = null;
   // La socket en attente d'une demande de connexion. Une fois acceptée,
   // la méthode accept renverra un objet BluetoothSocket.
   private BluetoothServerSocket socketServeur = null;

   public AcceptThread(String nomService, UUID serviceUUID) {
      try {
         // Spécifiez le nom du service et l'UUID qui sera le même envoyé
         // par le client.
         socketServeur = adaptateurBluetooth.listenUsingRfcommWithServiceRecord(
                              nomService, serviceUUID);
      } catch (IOException e) { }
   }

   public void run() {
      // Tant qu'une connexion n'a pas été acceptée ou qu'une erreur ne s'est
      // produite, la boucle et l'appel à la méthode accept bloque le code.
      while (true) {
         try {
            socket = socketServeur.accept();
            } catch (IOException e) {
                break;
            }
            break;
```

```
            }
        }
    }

    // Cette méthode ferme la socket pour mettre fin à la connexion et libérer
    // les ressources.
    public void annuler() {
        try {
            if (serverSocket != null)
                serverSocket.close();
        } catch (IOException e) { }
    }
}
```

Créer un client

Avant de pouvoir demander une connexion à un appareil distant, vous devez posséder l'objet `BluetoothDevice` de ce dernier (soit en parcourant les appareils liés, soit en effectuant une nouvelle détection d'appareils). La création d'un client suit la logique suivante :

1. Appelez la méthode `createRfcommSocketToServiceRecord` de votre objet `BluetoothDevice` en spécifiant l'UUID du service désiré (le même que celui que vous aurez spécifié au `BluetoothServerSocket`). Cette méthode vous retourne un objet `BluetoothSocket` qui vous permettra de vous connecter à l'appareil plus tard.
2. Appelez la méthode `connect` de votre instance `BluetoothSocket`. Cet appel créera la connexion de communication RFCOMM avec l'autre appareil si le service distant accepte votre demande. Cette méthode est bloquante ; par conséquence placez toujours votre code dans un fil d'exécution indépendant.

Une fois connecté, vous pouvez vous déconnecter en appelant la méthode `close` de l'objet `BluetoothSocket`. Dans tous les cas, appelez cette méthode pour libérer les ressources.

Si la méthode `connect` est trop longue (par exemple lorsque l'appareil n'est pas suffisamment près ou si une recherche d'appareils Bluetooth est en train de s'exécuter en même temps), une exception sera lancée.

Code 14-16 : Création d'un client Bluetooth effectuant une demande de connexion

```
BluetoothAdapter adaptateurBluetooth = ...

...

class ConnectThread extends Thread {

    private BluetoothSocket socket;
```

```
    public ConnectThread(BluetoothDevice appareil, UUID uuid) {
        try {
            // Obtention de la socket client. Spécifiez le même identifiant UUID
            // que le service écoutant.
            socket = device.createRfcommSocketToServiceRecord(uuid);
        } catch (IOException e) { }
        mmSocket = tmp;
    }

    public void run() {
        // Si une détection d'appareil est en cours, nous
        // l'annulons afin de nous connecter à l'appareil
        // dans de bonnes conditions. Sans cela, la connexion
        // risque de prendre plus de temps à s'effectuer et
        // se solder par une exception.
        if (adaptateurBluetooth.isDiscovering())
            adaptateurBluetooth.cancelDiscovery();

        try {
            // Nous effectuons la connexion à l'appareil distant.
            socket.connect();
        } catch (IOException connectException) {
            try {
                socket.close();
            } catch (IOException closeException) { }
            return;
        }
    }

    // Cette méthode ferme la socket pour mettre fin à la
    // connexion et libérer les ressources.
    public void annule() {
        try {
            socket.close();
        } catch (IOException e) { }
    }
}
```

Échanger des données

Une fois le serveur et le client connectés, chacun va pouvoir échanger avec l'autre au travers de son objet `BluetoothSocket`. Comme pour toute communication par les sockets, les méthodes et techniques sont les mêmes. Vous pouvez utiliser les méthodes `getInputStream` et `getOuputStream` pour récupérer les flux entrant et sortant.

Une fois les flux récupérés, vous pourrez utiliser les méthodes `read` et `write` respectivement pour lire et écrire des données. Ces méthodes étant bloquantes, vous devrez toujours les exécuter dans un fil d'exécution séparé.

> **ERGONOMIE Utiliser le Bluetooth dans vos applications**
>
> Le Bluetooth est un puissant outil pour rendre vos applications communicantes (réseau social, service de transfert de données, partage de données ou de connexion, etc). Mais comme tout moyen de communication, il consomme de la batterie et diminue d'autant l'autonomie du téléphone de l'utilisateur. Sachez utiliser ce moyen de communication avec parcimonie et n'activez le Bluetooth que si nécessaire. Sachez également que la détection des appareils à proximité consomme plus que la moyenne : n'effectuez donc des détections qu'en cas de besoin et après avoir averti l'utilisateur de cet écueil.

Les capteurs

Une bonne partie du succès des terminaux modernes (comme les appareils Android, l'iPhone d'Apple ou la console Wii de Nintendo) est la gestion du mouvement. Cela ne serait pas possible sans l'extraordinaire démocratisation des capteurs embarqués. Désormais, votre appareil détecte le mouvement et peut se situer dans l'espace. Voyons comment tout cela fonctionne sur Android, en créant un projet qui utilise les capteurs.

Identification des capteurs

Le panel des capteurs s'est étoffé avec les versions successives d'Android : accéléromètre, capteur de pression, capteur de proximité.... Cela dit, il ne faut pas croire que tous les capteurs sont disponibles sur tous les terminaux, loin de là !

L'API d'Android 1.6 a introduit un nouvel objet *sensor* responsable de la récupération des données, toujours sous la houlette du `SensorManager` qui comme son nom l'indique chapeaute le groupe de capteurs au travers d'un service commun à tout le système.

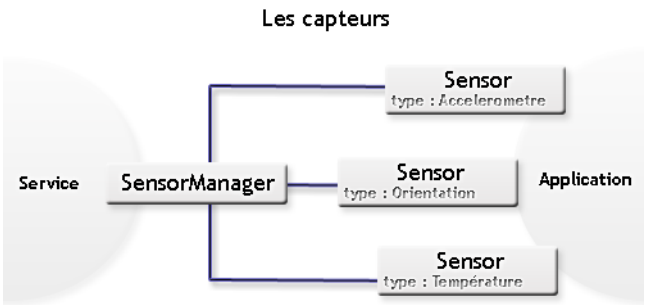


Figure 14–1
Représentation schématique des capteurs

La liste des différents types de capteurs est la suivante :

- TYPE_ACCELEROMETER : permet d'évaluer les mouvements du terminal ou tout simplement la gravité ;
- TYPE_GYROSCOPE : permet de connaître la position angulaire du terminal en fonction de trois axes x, y, z ;
- TYPE_LIGHT : le capteur de lumière permet d'évaluer l'exposition lumineuse subie par le terminal ;
- TYPE_MAGNETIC_FIELD : permet de détecter les modifications des champs magnétiques environnants ;
- TYPE_ORIENTATION : permet de déterminer la position du terminal en fonction de trois axes x, y, z ;
- TYPE_PRESSURE : indique la pression atmosphérique ;
- TYPE_PROXIMITY : indique la distance du terminal par rapport à un objet ;
- TYPE_TEMPERATURE : indique la température à proximité du capteur.

Utilisation des capteurs

Pour accéder aux capteurs nous allons dans un premier temps récupérer une instance du service (1) puis nous abonner aux événements du capteur qui nous intéressent (2).

Si le capteur n'est pas disponible sur le terminal, cette dernière étape renverra une valeur négative nous permettant d'agir en conséquence. Voyons cela en pratique.

Code 14-17 : Instanciation du service

```
SensorManager sensorMgr =
(SensorManager)getSystemService(SENSOR_SERVICE);(1)boolean accelSupported =
sensorMgr.registerListener(
   this,sensorMgr.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
   SensorManager.SENSOR_DELAY_UI); (2)
if (!accelSupported) {
   sensorMgr.unregisterListener(this,
   sensorMgr.getDefaultSensor(Sensor.TYPE_ACCELEROMETER));
   ((TextView)findViewById(R.id.acc)).setText("Pas d'accelerometre");
}
```

Comme vous pouvez le constater la récupération d'une instance de capteur s'associe à un taux de rafraîchissement des données qui peut être plus ou moins précis. Ce taux influence directement les performances de l'application ainsi que l'usage de la batterie. Plus le taux est élevé, plus les données sont rafraîchies et précises, et plus la consommation d'énergie est élevée. Les taux sont définis par des constantes dont voici la liste ci-dessous :

- SENSOR_DELAY_FASTEST : le taux de rafraîchissement le plus élevé. Les données sont récupérées aussi vite que possible par le système ;
- SENSOR_DELAY_GAME : taux utilisable pour des applications nécessitant des données précises telles que les jeux ;
- SENSOR_DELAY_NORMAL : taux normal utilisé par exemple par le système pour détecter les changements d'orientation ;
- SENSOR_DELAY_UI : le taux le plus bas utilisé dans les applications ne nécessitant qu'une faible précision ou ne sollicitant que très peu de données, comme les composants graphiques d'une interface utilisateur.

Notez également que pour obtenir des données, nous nous sommes enregistrés auprès du service comme écouteur d'événement grâce à la méthode registerListener. Après utilisation de ces données ou avant de quitter l'application, il est indispensable de se désenregistrer en utilisant la méthode opposée unregisterListener.

Code 14-18 : Dés-enregistrement de l'écouteur

```
if (accelSupported)
   sensorMgr.unregisterListener(this,
   sensorMgr.getDefaultSensor(Sensor.TYPE_ACCELEROMETER));
```

Le système se chargera d'appeler notre écouteur le moment venu. La notification se fera via deux méthodes onAccuracyChanged et onSensorChanged. Ces fonctions sont utilisées respectivement pour notifier d'un changement de sensibilité et fournir les informations sur les données des capteurs. Voyons cela avec du code :

Code 14-19 : Utilisation de l'écouteur

```
public class MainActivity
   extends Activity
   implements SensorEventListener {

   public void onAccuracyChanged(Sensor sensor, int accuracy) {
      switch(accuracy){
      case SensorManager.SENSOR_STATUS_ACCURACY_HIGH:
      case SensorManager.SENSOR_STATUS_ACCURACY_MEDIUM:
      case SensorManager.SENSOR_STATUS_ACCURACY_LOW:
      case SensorManager.SENSOR_STATUS_UNRELIABLE:
      }
      Log.d("Sensor",sensor.getType()+":"+accuracy);
   }

   public void onSensorChanged(SensorEvent event) {
      switch(event.sensor.getType()){
```

```
        case Sensor.TYPE_ACCELEROMETER:
            onAccelerometerChanged(event);
            break;
        case Sensor.TYPE_ORIENTATION:
            onOrientationChanged(event);
            break;
        case Sensor.TYPE_MAGNETIC_FIELD:
            onMagneticFieldChanged(event);
            break;
        case Sensor.TYPE_TEMPERATURE:
            onTemperatureChanged(event);
            break;
        case Sensor.TYPE_PROXIMITY:
            onProximityChanged(event);
            break;
        case Sensor.TYPE_LIGHT:
            onLightChanged(event);
            break;
        case Sensor.TYPE_PRESSURE:
            onPressureChanged(event);
            break;
        case Sensor.TYPE_GYROSCOPE:
            onGyroscopeChanged(event);
            break;
        }
    }
    //...
```

Chaque capteur renvoie un tableau de valeurs contenant les données qui lui sont propres. Il nous faut donc concevoir une méthode par type de capteur.

Réception des données

Nous avons vu plus haut comment instancier un capteur et comment être notifié des changements d'état ou de sensibilité. Voyons à présent comment interpréter le tableau de valeurs envoyé par chaque capteur.

Code 14-20 : Utilisation de l'accéléromètre

```
private void onAccelerometerChanged(SensorEvent event){
    float x,y,z;
    x = event.values[0];
    y = event.values[1];
    z = event.values[2];
    ((TextView)findViewById(R.id.axex)).setText("Axe X: "+x+"m/s^2");
    ((TextView)findViewById(R.id.axey)).setText("Axe Y: "+y+"m/s^2");
    ((TextView)findViewById(R.id.axez)).setText("Axe Z: "+z+"m/s^2");
}
```

Le système est relativement simple à mettre en œuvre : le capteur détecte un changement de vitesse et renvoie l'accélération subie par le terminal selon les axes x,y,z. Chaque axe se retrouve à un index précis du tableau.

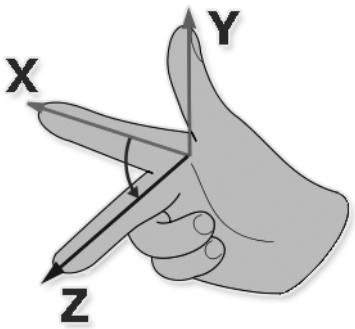


Figure 14–2
Repère main droite

Code 14-21 : Utilisation du capteur d'orientation

```
private void onOrientationChanged(SensorEvent event){
   float azimuth,pitch,roll;
   azimuth = event.values[0];
   pitch   = event.values[1];
   roll    = event.values[2];
   ((TextView)findViewById(R.id.azimuth)).setText("Azimuth: "+azimuth+"°");
   ((TextView)findViewById(R.id.pitch)).setText("Pitch: "+pitch+"°");
   ((TextView)findViewById(R.id.roll)).setText("Roll: "+roll+"°");
}
```

Il en va de même pour le capteur d'orientation : une fois que l'on connaît l'indexation des données, il nous suffit de les récupérer depuis le tableau puis de les utiliser comme bon nous semble.

Attention toutefois, certains capteurs peuvent induire en erreur si les données sont mal interprétées. Par exemple, le capteur de température est fortement soumis à la température des composants. Ainsi lorsque le terminal est en charge ou fortement sollicité, la variation entre la température ambiante et la température du capteur accuse plusieurs degrés.

Code 14-22 :Utilisation du capteur de température

```
private void onTemperatureChanged(SensorEvent event){
   float temperature;
   temperature = event.values[0];
   ((TextView)findViewById(R.id.temp)).setText("Temperature: "+temperature+"°F");
}
```

Code 14-23 : Utilisation du capteur de champ magnétique

```
private void onMagneticFieldChanged(SensorEvent event){
   float uTx,uTy,uTz;
   uTx = event.values[0];
   uTy = event.values[1];
   uTz = event.values[2];
   ((TextView)findViewById(R.id.uTx)).setText("Axe X: " + uTx + "µT");
   ((TextView)findViewById(R.id.uTy)).setText("Axe Y: " + uTy + "µT");
   ((TextView)findViewById(R.id.uTz)).setText("Axe Z: " + uTz + "µT");
}
```

Les autres capteurs ne sont, pour le moment, pas disponibles sur les terminaux en vente sur le marché. Néanmoins, rien ne nous empêche d'étudier leur principe de fonctionnement pour une utilisation future.

Code 14-24 : Utilisation du capteur de lumière

```
private void onLightChanged(SensorEvent event){
   float illuminance;
   illuminance = event.values[0];
   ((TextView)findViewById(R.id.litex)).setText("Light: "+ illuminance + "lux");
}
```

Code 14-25 : Utilisation du capteur de proximité

```
private void onProximityChanged(SensorEvent event){
   float distance;
   distance = event.values[0];
   ((TextView)findViewById(R.id.prox)).setText("Proximity: " + distance + "cm");
}
```

Code 14-26 : Utilisation du capteur de pression

```
private void onPressureChanged(SensorEvent event){
   float pressure;
   pressure = event.values[0];
   ((TextView)findViewById(R.id.pressex)).setText("Pressure: " + pressure + "psi");
}
```

Code 14-27 : Utilisation du gyroscope

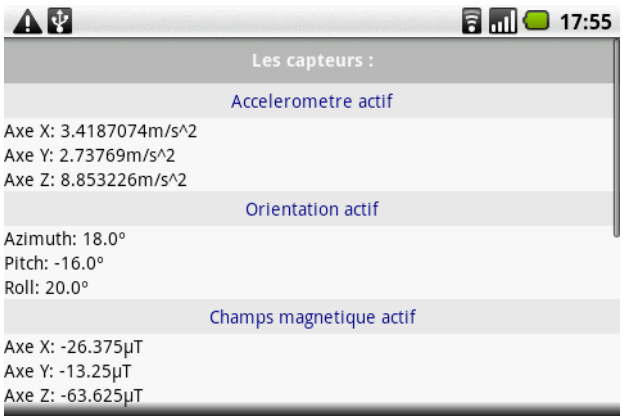
```
private void onGyroscopeChanged(SensorEvent event){
   float x, y, z;
   x = event.values[0];
   y = event.values[1];
```

```
    z = event.values[2];
    ((TextView)findViewById(R.id.gyrox)).setText("Axe X: " + x + "°");
    ((TextView)findViewById(R.id.gyroy)).setText("Axe Y: " + y + "°");
    ((TextView)findViewById(R.id.gyroz)).setText("Axe Z: " + z + "°");
}
```

L'émulateur d'Android ne permet pas d'obtenir les données de capteurs virtuels : il est donc nécessaire de travailler sur un vrai terminal.

Une fois l'application déployée et démarrée, nous obtenons la valeur de chaque capteur directement depuis le formulaire principal comme le montre la capture d'écran ci-après.

Figure 14–3
Le projet lancé sur un terminal



Grâce à ce système, nous sommes en mesure de réaliser une petite application simulant une boule sur un comptoir. En bougeant le terminal, la boule se déplacera comme si elle était réellement soumise à la gravité.

Code 14-28 : Démonstration de l'accélérometre

```
public class MainActivity
    extends Activity
    implements SensorEventListener {

    private SensorManager manager;
    private SurfaceHolder holder;
    private SurfaceView surface;
    private Bitmap boule,fond;
    private float bx,by,vx,vy;

    // Creation de l'activité.
    @Override
```

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Resources res = getResources();
    boule   = BitmapFactory.decodeResource(res, R.drawable.boule);
    fond    = BitmapFactory.decodeResource(res, R.drawable.background);
    surface = new SurfaceView(this);
    holder = surface.getHolder();
    manager = (SensorManager) getSystemService(Service.SENSOR_SERVICE);
    setContentView(surface);
}

// Démarrage ou reprise de l'activité.
@Override
protected void onStart() {
   super.onStart();
   Sensor sensor = manager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
   manager.registerListener(this, sensor, SensorManager.SENSOR_DELAY_GAME);
 }

 // Arrêt de l'activité.
@Override
protected void onStop() {
   super.onStop();
   Sensor sensor = manager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
   manager.unregisterListener(this,sensor);
}

// La précision du capteur a changé.
@Override
public void onAccuracyChanged(Sensor arg0, int arg1) {}

// Récupération des données.
@Override
public void onSensorChanged(SensorEvent event) {
   switch(event.sensor.getType()){
   case Sensor.TYPE_ACCELEROMETER:

      // On ajoute les valeurs du capteur aux variables.
      vx -= event.values[0];
      vy += event.values[1];

      // On déplace la boule.
      bx += vx;
      by += vy;

      // On récupère la taille de l'écran.
      float tx = surface.getWidth() - boule.getWidth();
      float ty = surface.getHeight()- boule.getHeight();
```

```
            // On empêche la boule de sortir.
            if (bx<0) {bx = 0; vx = 0;};
            if (by<0) {by = 0; vy = 0;};
            if (bx>tx){bx = tx; vx = 0;};
            if (by>ty){by = ty; vy = 0;};

            // On redessine.
             Canvas c = holder.lockCanvas();
             if(c==null)return;
             c.drawBitmap(fond, 0, 0, null);
             c.drawBitmap(boule, bx, by, null);
             holder.unlockCanvasAndPost(c);
        }
    }
}
```

La fonction la plus importante de cet exemple est sans nul doute `onSensorChanged()`. On utilise les valeurs du capteur dès leur réception et on profite du taux de rafraîchissement élevé pour redessiner l'écran dans la foulée.

Figure 14–4
Capture d'écran de la boule en déplacement

En résumé

Quelquefois vous aurez besoin d'une connexion et d'une bande passante élevées. Vous ne pouvez pas compter sur la connexion téléphonique (EDGE, 3G, etc) pour répondre à tous vos besoins. Les capacités des appareils Android à exploiter les réseaux Wi-Fi et la connectivité Bluetooth doivent être prises comme de véritables opportunités pour créer des applications communicantes. Sachez cependant que l'activation de ces fonctionnalités est consommatrice d'énergie et que l'autonomie de

l'utilisateur n'en sera que réduite. Informez-en l'utilisateur, surtout si le niveau de la batterie est faible.

Android possède un nombre de capteurs permettant de prendre en compte l'environnement de l'utilisateur : orientation et localisation, température, champs magnétique, etc. Tous ces capteurs peuvent être utilisés par vos applications.

Sachez néanmoins avant tout détecter si les capteurs sont disponibles et en état de marche (réseau accessible ? localisation GPS activée ? etc.). Chaque appareil pouvant posséder ou non certains capteurs, vos applications doivent donc être préparées à tous les cas de figure.

15

Publier ses applications

Le développement d'une application est une étape en soi. Cela prend du temps, requiert des compétences et de la rigueur. Que vous vouliez tirer parti de votre travail en vendant votre application ou bien gagner en notoriété en la fournissant gratuitement à la communauté, vous avez dans les deux cas une excellente raison de partager votre application avec les autres utilisateurs !

La publication d'une application n'est que le commencement du cycle de vie de cette dernière car une fois en ligne, vous aurez besoin de la mettre à jour régulièrement, soit pour corriger des bogues notifiés par vos utilisateurs, soit pour proposer de nouvelles fonctionnalités ou adapter votre création aux différentes versions du SDK. Ce chapitre donne quelques conseils pour mettre en ligne votre application et ne pas rater sa sortie.

L'Android Market

L'Android Market – le marché d'applications Android – permet aux développeurs du monde entier de mettre leurs applications à disposition des utilisateurs d'Android. Ceux-ci pourront ainsi télécharger, gratuitement ou moyennant une rétribution, n'importe quelle application déposée dans cet espace.

Comme nous le verrons, certaines applications ne sont disponibles que dans un ou plusieurs pays spécifiques. Le choix des pays dont les utilisateurs seront autorisés à télécharger votre application sera à faire lors de la publication.

Avant de publier votre application sur le marché Android, il vous faut :

1. vous inscrire et créer un compte développeur sur l'Android Market ;
2. payer un enregistrement – très abordable – de 25 $ (environ 20 € au moment de l'écriture de cet ouvrage) ;
3. signer la charte de distribution du développeur du marché Android ;
4. vérifier votre application et signer votre fichier `.apk` ;
5. et enfin publier votre application !

S'inscrire en tant que développeur sur le marché Android

Ouvrir un compte développeur est la première chose à faire (les heureux possesseurs d'un tel compte peuvent passer directement à la partie suivante). Pour vous rendre sur le site officiel du marché Android et vous y inscrire, saisissez l'adresse http://www.android.com/market dans votre navigateur favori. Vous obtiendrez une page du type :

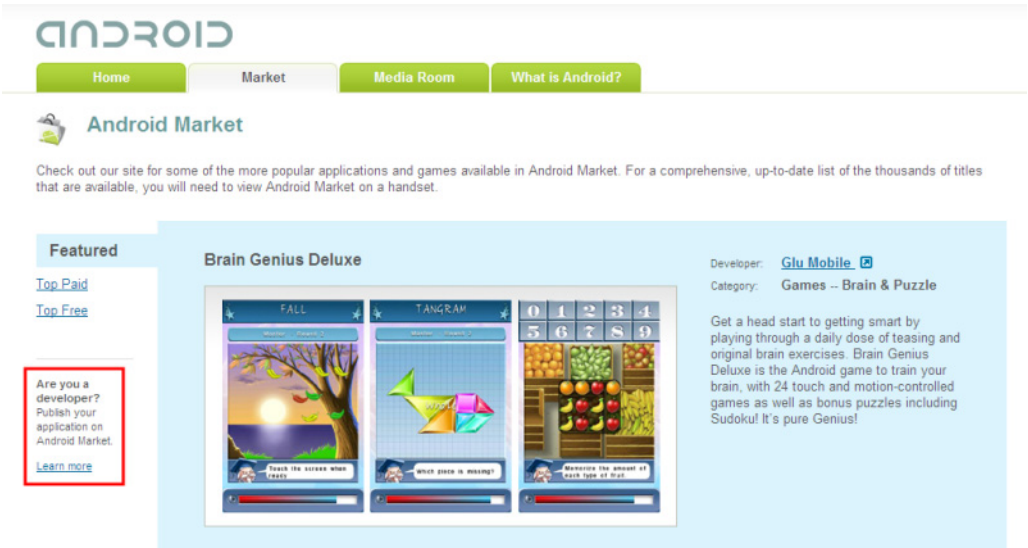


Figure 15–1 La page principale de l'Android Market sur Internet

Pour accéder à l'espace développeur, cliquez sur *Learn More* dans le menu gauche du site ou saisissez directement l'adresse suivante dans votre navigateur : http://market.android.com/publish/Home.

> **À NOTER Mise à jour des procédures d'inscription**
>
> Les procédures décrites dans la suite de ce chapitre peuvent être susceptibles d'évoluer en fonction des mises à jour des sites.

> **À SAVOIR Création obligatoire d'un compte Google pour s'inscrire sur le marché Android**
>
> L'inscription sur le marché Android nécessite d'avoir un compte Google. Si vous n'en avez pas, il vous faudra donc en créer un, soit depuis la page http://market.android.com/publish/Home (en cliquant sur *Create an account now*), soit depuis la page https://www.google.com/accounts/NewAccount (qui a l'avantage d'être dans la langue de votre navigateur).

Une fois muni de votre identifiant Google, vous pouvez vous authentifier sur la page d'inscription du marché Android. Si vous n'êtes pas encore inscrit en tant que développeur, la procédure d'inscription commencera automatiquement.

La page d'inscription débute par la création du profil du développeur : nom, adresse mail, site internet et téléphone de contact. Saisissez bien tous les champs de façon à ce que l'on puisse vous contacter si un souci de paiement avec un autre utilisateur se produisait.

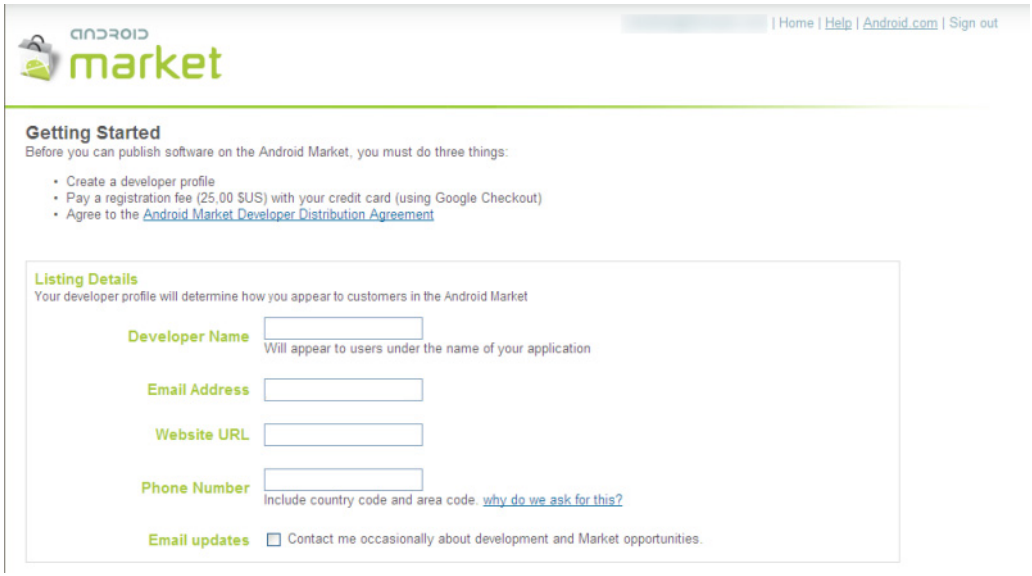


Figure 15–2 Création du profil du développeur

Une fois les informations de profil renseignées, cliquez sur *Continue* en bas de la page. L'assistant d'inscription vous proposera de payer les droits d'utilisation du service (fixé à 25 $ lors la rédaction de cet ouvrage).

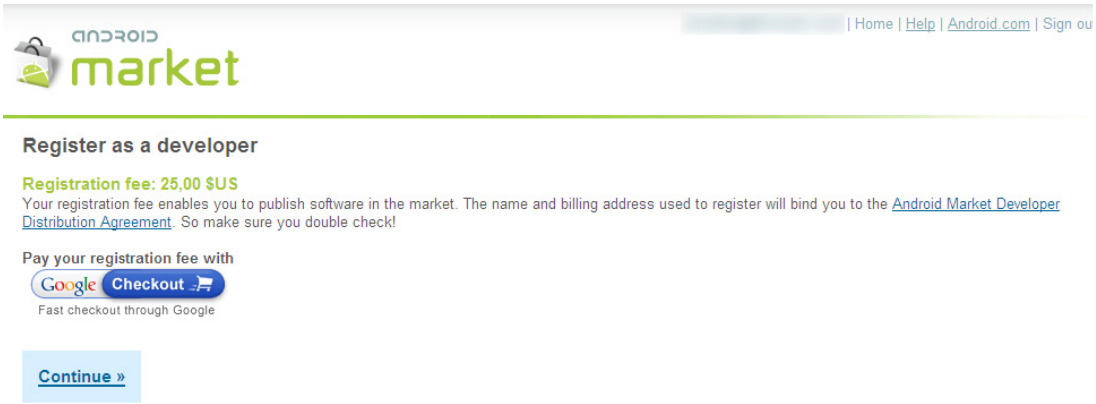


Figure 15–3 Paiement des droits d'enregistrement

Cliquez sur *Continue* à nouveau pour arriver sur la page d'ajout d'une carte bancaire à votre compte Google Checkout et l'approbation de la charte de distribution du développeur du marché Android.

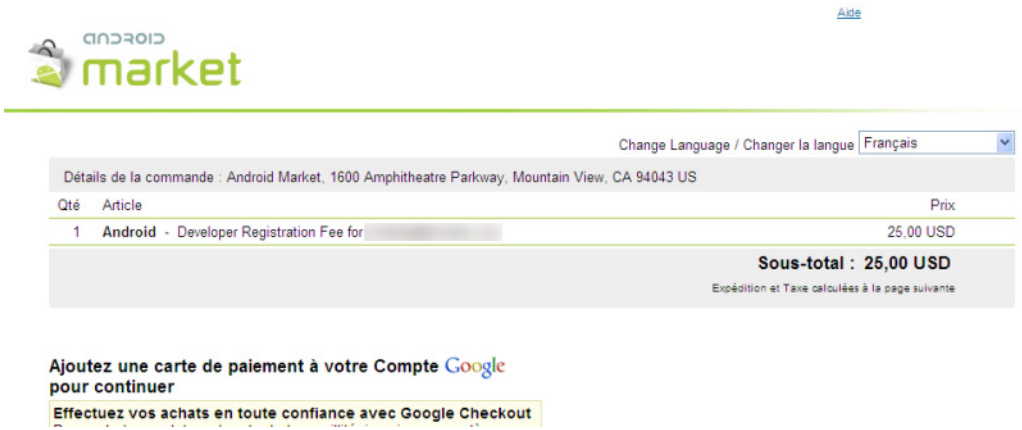


Figure 15–4 L'enregistrement d'un compte développeur coûte 25 $ géré avec Google Checkout.

Une fois que vous avez rempli les informations de paiement, cliquez sur *Accepter et continuer* pour terminer le processus. Vous devriez recevoir les informations récapitulant l'ensemble du processus et votre facture dans votre boîte de messagerie.

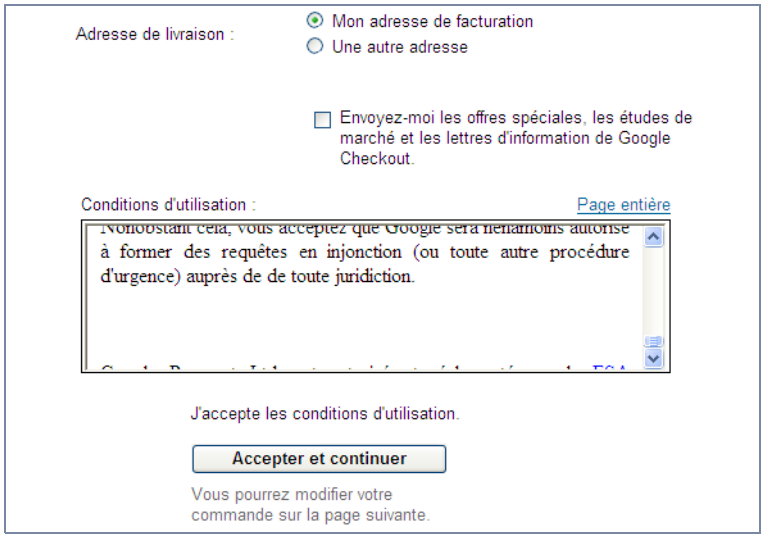


Figure 15–5
Acceptation de la licence d'utilisation pour devenir développeur

> **REMARQUE** **Interface anglophone du marché Android**
>
> À l'heure de l'écriture de cet ouvrage, l'interface proposée depuis le page d'accueil du marché Android n'est pas intégralement francisée. Vous devrez donc évoluer dans un interface écrite en anglais jusqu'aux premières étapes de votre inscription.

Félicitations, vous faites maintenant partie des développeurs Android et allez bientôt pouvoir rejoindre cette grande communauté et mêler vos talents de programmateur et de graphiste à ceux des autres.

Préparer son application pour la publication

La publication d'une application Android nécessite la création d'un paquetage – un fichier d'extension `.apk` contenant tout le code et les ressources de l'application – qui servira au système Android pour lire les propriétés et installer l'application sur le système.

Pour garantir à vos utilisateurs une bonne installation et utilisation de votre application, ne succombez pas tout de suite à la précipitation en vous ruant sur le marché Android pour y publier votre application. Prenez le temps qu'il faut pour vérifier celle-ci et ne pas rater votre lancement !

Vérifier son application

En terme de méthodologie, publier une application Android à destination des autres utilisateurs n'est pas très différent de la publication d'autres types d'applications à destination des ordinateurs de bureau ou des serveurs. L'application devra d'abord être testée, et fournie sous la forme d'un paquetage avant d'être livrée sur le marché Android.

Voici une liste des étapes les plus importantes à valider avant d'empaqueter votre application :

1. Testez votre application de façon exhaustive (tests unitaires, tests fonctionnels, tests de charge, etc).
2. Ajoutez, si vous le pensez nécessaire, une licence d'utilisation de votre application.
3. Prenez le temps de créer une icône et d'internationaliser le titre (si votre application est disponible en plusieurs langues).
4. Retirez toutes les routines et informations de débogage (alertes, fenêtres de messages, sortie console, etc.).
5. Vérifiez tous les fichiers de ressources de l'application (fichiers d'internationalisation présents, images en résolution adéquate et de taille correcte pour un affichage optimal, etc).
6. Indiquez les numéros de version de l'application et du code (voir plus loin).
7. Obtenez une clé privée adéquate pour signer l'application.

Après avoir passé toutes ces étapes, vous pouvez enfin empaqueter l'application :

8. Signez l'application avec la clé privée précédemment obtenue.
9. Testez votre application signée dans l'émulateur.
10. Testez l'application sur un appareil réel !

N'oubliez surtout pas cette dernière étape indispensable avant de livrer votre application sur le marché Android ! Sitôt toutes ces étapes passées, vous serez fin prêt à publier votre application et peut-être à gagner un peu d'argent avec.

Ajouter des informations de version à une application

Une application Android est, comme toutes les applications informatiques, une création évolutive. À ce titre, elle possède un cycle de vie commençant par une première version, qui se transforme ensuite au fil du temps et des évolutions fonctionnelles et technologiques, créant ainsi de multiples versions.

Que ce soit pour des raisons marketing, pour notifier l'ajout de corrections et/ou de nouvelles fonctionnalités ou pour suivre l'évolution de l'application par l'équipe de développement, vous devez spécifier la version de votre application. Un numéro de version d'une application permet notamment de faire le lien avec de potentielles

dépendances vers d'autres briques logicielles et de permettre de gérer la compatibilité avec les différentes versions de la plate-forme Android.

Une application Android possède donc, au même titre que d'autres plates-formes, une gestion des versions. Vous pouvez spécifier deux valeurs de version (`versionCode` et `versionName`) dans le manifeste de l'application : ces deux attributs doivent être renseignés avant de compiler votre code :

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   paquetage="com.eyrolles.android.demo" android:versionCode="3"
   android:versionName="2.5">
```

L'attribut `versionCode` est un nombre entier spécifiant la version de votre code. De façon générale, vous publierez votre application avec une version de code égale à 1, puis vous incrémenterez ce chiffre de 1 à chaque évolution, qu'elle soit majeure ou mineure. Ce numéro de version peut être consulté par d'autres applications de façon à vérifier la compatibilité, sans pour autant être affiché à l'utilisateur. Notez que la version du code n'a pas de lien direct avec la version de l'application – attribut `versionName` – qui est visible quant à elle par l'utilisateur.

L'attribut `versionName` est une chaîne de caractères qui représente le numéro de version de l'application qui sera affiché à l'utilisateur. Cette chaîne permet de spécifier n'importe quel format de version. Généralement, les développeurs utilisent le formalisme `<majeur>.<mineur>.<révision>` pour spécifier un numéro de version. Vous pouvez également compléter celui-ci d'un qualificatif quant à la maturité de l'application, par exemple « 1.0.1 Beta ». Ce numéro de version est purement informatif et à destination des utilisateurs : celui-ci ne sera pas utilisé par le système et/ou les services de publication.

À chaque publication sur le marché Android, n'oubliez pas d'incrémenter les deux valeurs.

Compiler et signer une application avec ADT

Toutes les applications Android doivent être numériquement signées pour être installées sur le système Android. La signature, à l'aide d'un certificat contenant votre clé privée, permet d'identifier le développeur et de créer une relation de confiance entre celui-ci et le système. Le certificat n'a pas besoin d'être signé par une autorité certifiée ; un certificat auto-signé est accepté.

Le système Android n'exécutera pas et n'installera pas une application non signée. Cette règle est tout aussi vraie dans le cadre du développement. En effet, pour pouvoir signer votre application, vous devez utiliser un certificat. Par défaut, lorsque le complément ADT compile un projet, il utilise un certificat généré automatiquement.

Afin de distribuer votre application sur le marché Android, ce certificat de débogage devra être remplacé par celui du développeur ou de l'entreprise. Toutes les applications d'un même développeur ou d'une même entreprise sont en général signées avec le même certificat.

Il existe plusieurs façons de signer vos applications pour les publier sur le marché Android, les deux plus connues sont avec ADT et avec les outils standards *keystore/jarsigner*. C'est avec ADT que nous allons procéder pour la suite de ce chapitre.

La première étape pour signer votre application est de la compiler sans la signer. Ensuite vous devez créer ou récupérer une clé privée adéquate avec laquelle vous signerez l'application.

Les utilisateurs d'Eclipse et d'ADT peuvent utiliser l'assistant d'export pour compiler l'application en la signant ou non avec une clé privée. Vous pouvez aussi générer un nouveau trousseau de clés et une nouvelle clé privée directement depuis cette interface.

Pour compiler votre application et la signer en une seule action, effectuez un clic droit sur votre projet dans la fenêtre d'exploration de paquetage – vue Eclipse nommée *Package Explorer* – puis sélectionnez *Android Tools > Export Signed Application package*.

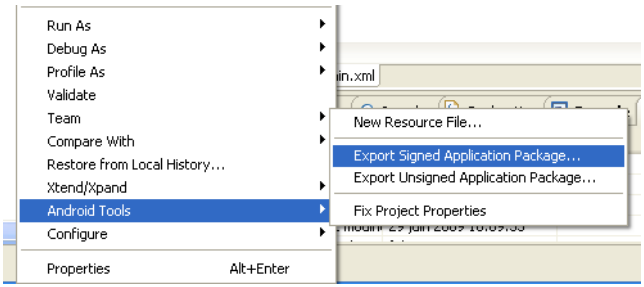


Figure 15–6
Exportez votre application et signez-la en une seule action grâce à ADT

L'assistant d'export et de signature de l'application vous propose alors de choisir une application Android dans votre espace de travail :

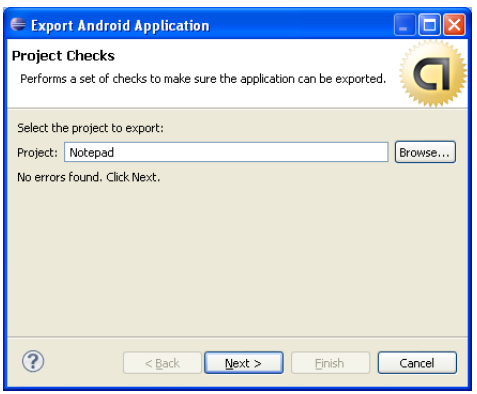


Figure 15–7
Sélectionnez le projet que vous désirez exporter, puis appuyez sur 'Next'.

Si vous possédez déjà un fichier `.keystore` – que vous auriez généré avec l'outil *keystore* du SDK Java – contenant la clé privée avec laquelle vous souhaitez signer votre application, sélectionnez l'option *Use existing keystore*. Sinon créez un nouveau trousseau de clés en spécifiant son emplacement et le mot de passe de protection.

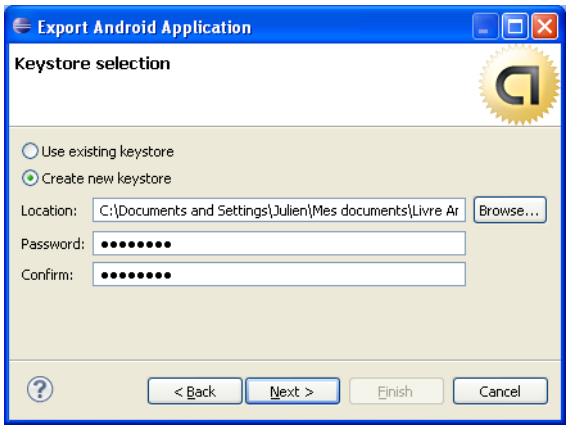


Figure 15–8
Création d'un nouveau trousseau de clés

Spécifiez ensuite l'alias de la clé à générer – de façon à pouvoir l'identifier par la suite – ainsi qu'un mot de passe de façon à protéger votre clé privée.

> **À RETENIR** **Notez votre mot de passe de clé !**
>
> Notez votre mot de passe ! Celui-ci vous sera redemandé à chaque fois que vous souhaiterez signer une application.

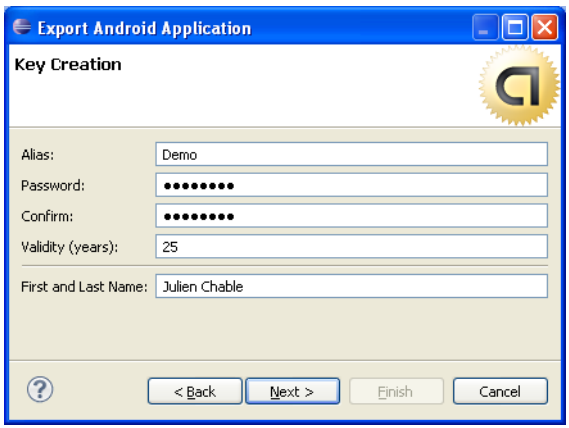


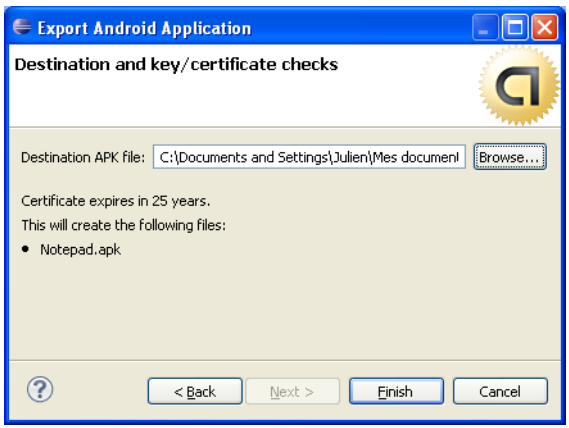
Figure 15–9
Création d'une clé valable 25 ans

Entrez toutes les informations sur vous ou votre entreprise pour cette clé. Ces informations seront stockées et permettront à vos utilisateurs d'identifier l'origine de l'application et de son auteur.

Figure 15–10
Les informations concernant votre clé sont à remplir rigoureusement.

Pour compiler, générer votre clef et signer l'application, cliquez sur *Finish*.

Figure 15–11
Dernier écran de l'assistant permettant de générer une clé, de compiler et de signer l'application



Le fichier `.apk` résultant de cette opération n'est ni plus ni moins que votre application signée numériquement et prête à rejoindre les milliers d'applications du marché Android.

Publier son application sur le marché Android

Vous avez vérifié et signé votre application, tous les feux sont ainsi au vert pour publier votre application sur le marché Android.

Présentation de la console de publication des applications

Connectez-vous sur le marché Android (http://market.android.com/publish/Home). Une fois authentifié avec les identifiants que vous avez saisis lors de l'enregistrement, vous serez redirigé sur la console du développeur.

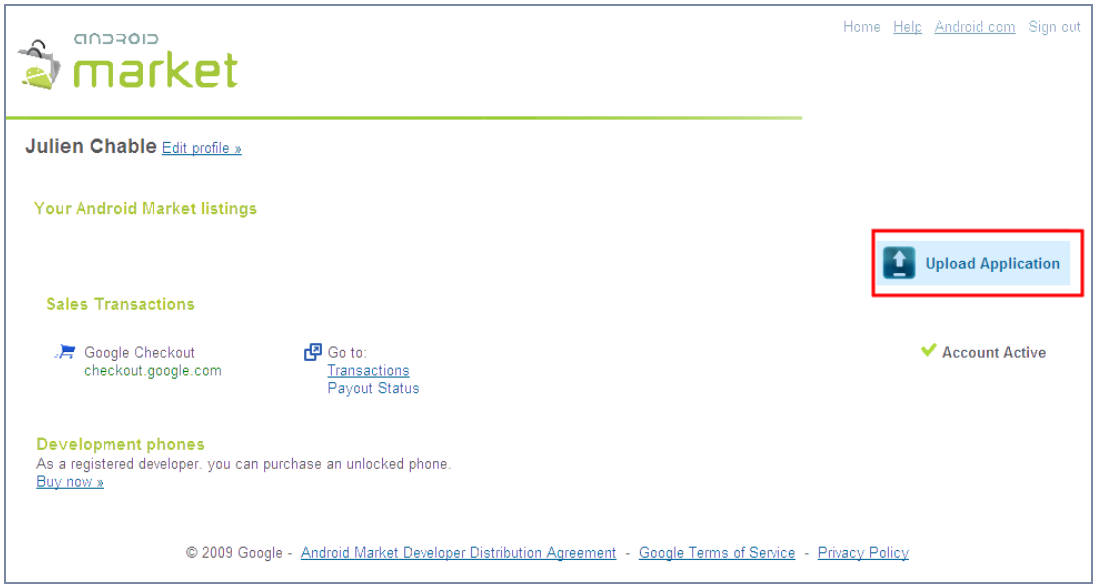
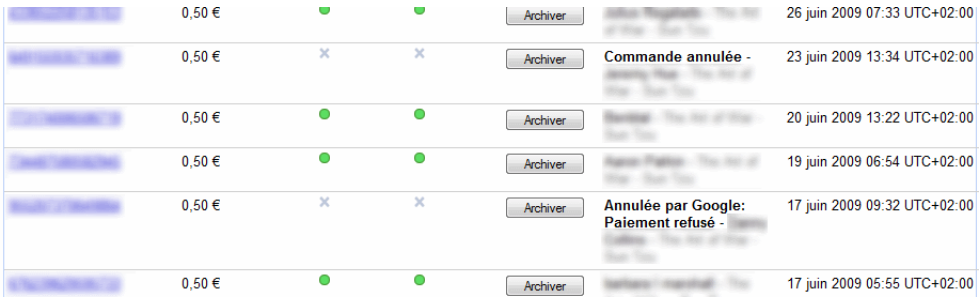


Figure 15–12 La console de développeur Android

Cette dernière offre une vue sur l'ensemble des développements déjà publiés ou en cours de publication. Vous pouvez également modifier votre profil (nom, adresse de messagerie, téléphone, etc.) si celui-ci a évolué depuis votre enregistrement sur le marché Android.

La section *Sales Transactions* permet de consulter et de gérer toutes les transactions effectuées sur votre compte *Google CheckOut* en rapport avec le marché Android. En cliquant sur le lien *Transactions* vous pourrez consulter et gérer les commandes effectuées par les utilisateurs. Quelque fois une action de votre part sera nécessaire, par exemple, si un utilisateur rencontre un souci.



[redacted]	0,50 €	●	●	Archiver	[redacted]	26 juin 2009 07:33 UTC+02:00
[redacted]	0,50 €	×	×	Archiver	Commande annulée - [redacted]	23 juin 2009 13:34 UTC+02:00
[redacted]	0,50 €	●	●	Archiver	[redacted]	20 juin 2009 13:22 UTC+02:00
[redacted]	0,50 €	●	●	Archiver	[redacted]	19 juin 2009 06:54 UTC+02:00
[redacted]	0,50 €	×	×	Archiver	Annulée par Google: Paiement refusé - [redacted]	17 juin 2009 09:32 UTC+02:00
[redacted]	0,50 €	●	●	Archiver	[redacted]	17 juin 2009 05:55 UTC+02:00

Figure 15–13 Les transactions avec vos utilisateurs sont accessibles depuis la console du développeur.

Le lien *Payout Status* permet de consulter le récapitulatif des paiements et les frais de traitement (par exemple, les anomalies pouvant vous être directement imputables) des ventes de vos applications.

Récapitulatif des paiements En savoir plus — 1 - 20 sur 24 dates

Date ▼	Initial Solde	+ Achats	+ Autre Activité	- Paiement	= Clôture Solde
13 juil. 2009	16,93 €	0,35 €	-	-	17,28 €
8 juil. 2009	16,58 €	0,35 €	-	-	16,93 €
7 juil. 2009	16,23 €	0,35 €	-	-	16,58 €
6 juil. 2009	15,88 €	0,35 €	-	-	16,23 €

Frais de traitement (Mois en cours) Historique des frais

Frais de base	(depuis 07/05/2009)	0,00% + 0,00 €

Figure 15–14 Le récapitulatif des paiements et des frais de traitement

En tant que développeur enregistré sur le marché Android, la section *Development Phones* vous proposera une offre pour acheter un téléphone débridé dédié au développement : exécution et débogage de vos applications, possibilité de modifier et de compiler votre propre version du système d'exploitation Android, indépendant des opérateurs de téléphonie, etc. pour un tarif de 399 $ (environ 300 €). Vous trouverez plus de détail en annexe de ce livre.

Pour terminer, le bouton situé en haut à droite de la page nommé *Upload Application* permet de lancer le processus de publication de vos applications sur le marché Android.

Publier son application depuis la console du marché Android

Cliquez sur le lien du bouton *Upload Application* pour accéder à la page de publication. C'est à cet endroit que vous allez enfin pouvoir télécharger le fichier `.apk` de votre application signée et remplir les différentes informations nécessaires à sa distribution auprès de vos futurs utilisateurs.

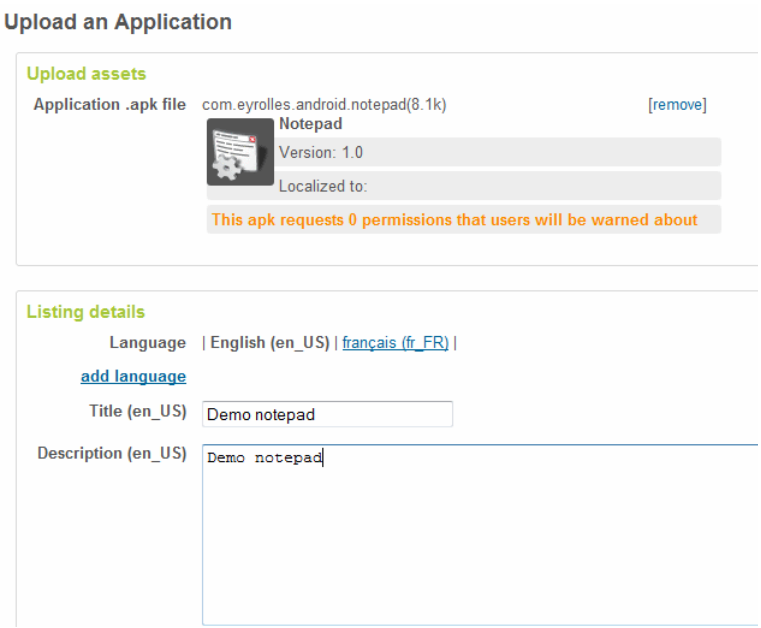


Figure 15–15
Téléchargement et description de l'application

Pour télécharger votre application vers la plate-forme du marché Android, cliquez sur le bouton *Browse* pour sélectionner le fichier `.apk` de votre application sur votre disque dur. Pour terminer et lancer le processus de téléchargement, cliquez sur *Upload*.

La plate-forme devrait mettre quelques secondes à traiter votre application afin de vérifier la version, d'extraire l'icône et les permissions. Ces informations sont ensuite affichées à l'écran. Si votre application porte le même nom qu'une autre de vos applications déjà publiée et que la version est égale ou inférieure à celle déjà existante, un message d'erreur apparaîtra (la partie suivante traitera des mises à jour plus en détails).

Vous pouvez ajouter plusieurs titres et descriptions en plusieurs langues en ajoutant une langue supplémentaire en cliquant sur le lien *add language*. La capture d'écran ci-dessus possède une description en français et en anglais.

Les dernières mises à jour du marché Android permettent aujourd'hui d'ajouter des captures d'écran de votre application (maximum deux). Nous ne saurions que vous conseiller de fournir ces captures d'écran ; une illustration étant bien plus vendeuse qu'une simple description.

Une fois ces informations renseignées, il vous faudra renseigner la catégorie de l'application parmi un choix limité (exemple : Finance, Santé, Divertissement, Achats, etc.) et son prix (gratuit ou payant avec un prix unitaire au minimum de 0,50€).

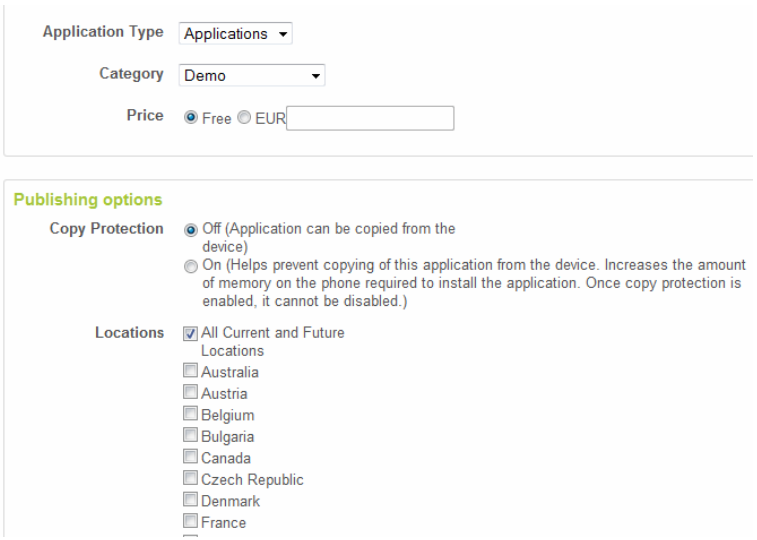


Figure 15–16
Saisie de la catégorie, du prix et des options de publication

La section des options de publication – ici *Publishing options* – permet de spécifier la protection de copie de l'application de l'appareil vers un ordinateur et les pays dans lesquels votre application sera disponible. Si vous souhaitez que votre application soit disponible dans tous les pays, laissez la case *All current and future locations* cochée.

Pour terminer, vous devez indiquer vos informations de contact, afin que l'utilisateur puisse vous contacter s'il rencontre un problème ou s'il souhaite vous communiquer des suggestions.

Figure 15–17 Les informations de contact pour les utilisateurs

Une fois toutes les informations saisies et vérifiées, cochez les deux cases de fin de formulaire – après avoir pris connaissance du contenu des accords – puis cliquez sur le bouton de publication *Publish* pour commencer la publication.

> **JURIDIQUE Lisez bien les documents avant de publier**
>
> Lorsque vous publiez votre application, vous vous engagez en vertu des éléments contractuels cités. Si vous cochez les cases de fin de formulaire, vous engagez votre entreprise ou votre personne. Par conséquent, lisez attentivement les différents clauses en question. Ne prenez pas cette étape à la légère : elle détermine le rapport juridique que vous aurez avec la plate-forme Android et vos utilisateurs !

Le bouton *Delete* sert à annuler la publication et le bouton *Save* à enregistrer les informations mais sans publier l'application sur le marché Android.

La page de la console du développeur réapparaît et vous devriez pouvoir observer votre application publiée :

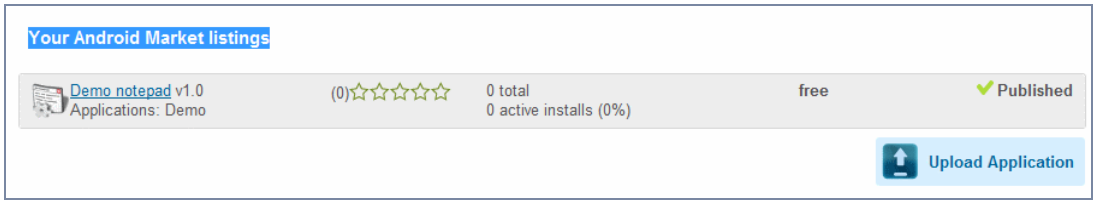


Figure 15–18 Votre application est publiée sur le marché Android !

Vous pouvez publier autant d'applications que vous le souhaitez.

Mettre à jour son application et notifier ses utilisateurs

Félicitations, vous avez déjà publié votre application sur le marché Android et vous avez corrigé ou rajouté de nouvelles fonctionnalités. Il vous reste maintenant à mettre cette nouvelle version sur le marché Android et à notifier vos utilisateurs de la bonne nouvelle.

Mettre à jour une application sur le marché Android

Afin de garantir que votre application sera bien reconnue comme une nouvelle version par le système Android, vous devez effectuer quelques modifications dans le manifeste de l'application :

- changer le numéro de version `android:versionCode` et `android.versionName` ;
- tester votre nouvelle version de l'application signée (avec la même rigueur que pour une nouvelle application).

Une fois ces opérations effectuées, vous êtes prêt pour mettre à jour votre application sur le marché Android en suivant la procédure suivante :

1. Dans la console du développeur, cliquez sur votre application.
2. En haut de la page, cliquez sur *Upload Upgrade*.
3. Cliquez sur *Browse* et sélectionnez le fichier `.apk` de la nouvelle version de votre application puis cliquez sur *Upload* pour envoyer l'application vers le marché Android.
4. Si aucun message d'erreur ne s'affiche, vous pouvez allez en bas de la page pour cliquer sur *Publish* afin de publier la nouvelle version de l'application et la mettre à disposition des utilisateurs du marché Android.

Notez que si vous essayez de mettre à jour une application en utilisant un certificat qui n'est pas identique à celui avec lequel vous aviez précédemment publié votre application, vous aurez le message d'erreur suivant :

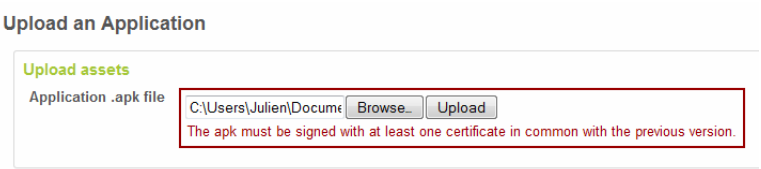


Figure 15–19
Échec de la publication à cause d'un certificat différent de celui utilisé originellement

Vous aurez également un message d'erreur si vous n'avez pas incrémenté le numéro de version du code `android:versionCode`.

En résumé

La publication de votre application sur l'Android Market doit être un événement bien préparé : vous devez tester, en autres, les performances de celle-ci, ainsi que sa compatibilité avec les différents modèles de téléphones (différents écrans) et les différentes versions d'Android. Ne vous ruez pas hâtivement sur le marché d'Android, au risque de passer à côté de points essentiels d'une application et de décevoir vos premiers utilisateurs qui vous attribueront de mauvaises notes certainement non méritées.

La mise à jour doit également faire l'objet du même rituel, peut même nécessiter une attention encore plus soutenue : il vous faudra incrémenter la version du code et de l'application, vérifier que les informations déjà possédées par l'utilisateur seront com-

patibles (ou devront faire l'objet d'une migration), faire attention à l'évolution de l'ergonomie de l'interface utilisateur, que l'application soit compatible avec les nouvelles versions du SDK Android et des nouveaux appareils arrivés dernièrement sur le marché, etc. En fonction des retours des utilisateurs, que ce soit des demandes de nouvelles fonctionnalités ou la correction de bogues, votre réactivité sera mise à rude épreuve pour garder vos utilisateurs.

Vous l'aurez compris, la publication reste une activité à part entière : répondre aux utilisateurs, corriger les bogues, ajouter de nouvelles fonctionnalités, etc. Tout comme pour un site Internet, il faudra aussi apporter un soin particulier au référencement de votre application : est-ce que les utilisateurs trouvent votre application sur le marché Android ? Utilisez-vous les bons mots pour que votre application apparaisse en premier lors d'une recherche sur le marché Android ?

N'oubliez pas non plus d'effectuer votre propre publicité pour promouvoir vos créations. Vous l'aurez compris, la publication d'une application n'est qu'un début, la suite dépendra de vous !

ANNEXE

Développer sur Android

Le développement sur Android n'est pas si difficile, du moins lorsque l'on sait utiliser les outils mis à notre disposition.

Utiliser les téléphones de développement

> **REMARQUE** **Acheter un téléphone de développement ou pas ?**
>
> Pour utiliser un appareil sur n'importe quel opérateur, il suffit désormais d'entrer le code de déverrouillage avant de taper le code PIN. Ce code peut être acheté auprès d'un revendeur spécialisé ou fourni directement par le constructeur. À noter que les appareils GeeksPhone, LG, Motorola et Samsung possèdent pour le moment des claviers. L'archos 5, lui, n'a pas de caméra, ni de GPS. Le Lenovo a un « dock » avec clavier. Pour pouvoir réellement tester une application Android, il faudrait pratiquement posséder tous les appareils, ce qui est rarement possible. Nous vous conseillons donc de bien organiser vos tests et de gérer un maximum de cas de figure. Si l'on devait ne retenir qu'un appareil au moment de la rédaction de ce livre, nous vous conseillerions le Nexus One.

Depuis l'origine d'Android, Google propose sur le marché Android deux téléphones de développement : les Android Dev Phone 1 et 2. Le premier est équivalent à un HTC Dream G1, le second est pour sa part un HTC Magic. Vous ne pourrez vous procurer ces téléphones qu'après vous être inscrit sur le site de l'Android Market en tant que développeur (la procédure d'inscription est décrite dans le chapitre 15 dédié à la publication des applications).

Ces téléphones de développement sont débridés, c'est-à-dire qu'ils sont utilisables avec tous les opérateurs et disposent des droits pour pouvoir être mis à jour avec n'importe quelle version de la plate-forme Android. Cependant concernant le G1, sa faible capacité mémoire ne lui permet pas d'installer Android2.0.

Tableau A–1 Tableau récapitulatif des caractéristiques techniques des deux téléphones de développement, telles qu'indiquées sur l'Android Market

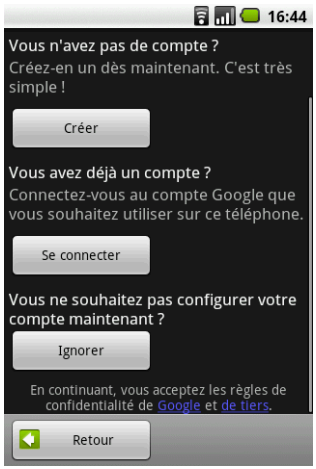
Dev Phone 1	Dev Phone 2
	
• Touch screen • Trackball • Appareil photo 3.2 Megapixel Auto focus • Wi-Fi • Bluetooth v2.0 • 3G WCDMA (1700/2100 MHz) • GSM (850/900/1800/1900 MHz) • GPS • Clavier coulissant QWERTY • Inclut une carte MicroSD 1GB (peut être remplacée par une carte allant jusqu'à 16Go)	• Android 1.6 • Dimension (mm) 113 x 55.56 x 14.65 • Taille écran 3.17 pouces • HVGA Resolution • Touch screen • Mémoire Flash 512MB • 192MB RAM • MSM7200A,528MHz chipset • GSM/GPRS/EDGE • 850/900/1800/1900 MHz • WCDMA 1700/2100 MHz : BC4 • 2100 MHz : BC1 • HSPA Speed HSDPA 7.2 Mbps • HSUPA 2 Mbps • Blutooth 2.0 avec EDR • Wi-Fi 802.11b/g • Appareil photo Auto Focus 3 Mégapixels • 1340 Battery(mAh) • Port mémoire microSD • USB 2.0 • GPS/AGPS • Mailing list
399€	399€

> **À NOTER Téléphones de développement versus téléphones du commerce**
>
> Il y a encore quelques mois, seuls les téléphones de développement pouvaient être débridés et modifiés complètement au niveau logiciel (« rooté » dans le jargon du développeur), y compris au niveau des accès système. Ils ont ainsi permis de promouvoir la plateforme Android et d'en accélérer l'adoption par les développeurs.
> De nos jours, les nouvelles générations d'appareils (tel que le Nexus One) ont dépassé leurs ainés puisqu'en plus de pouvoir être débridés (se renseigner auprès du constructeur ou du vendeur pour cela) et « rootés », ils offrent des caractéristiques techniques plus avancées pour un prix tout aussi intéressant. Un tel téléphone peut ainsi être utilisé comme téléphone de développement en lieu et place des téléphones historiques.
> Dans ce qui suit, nous utiliserons néanmoins un téléphone de développement, au cas où vous en auriez un.

Première utilisation du téléphone de développement

Après avoir déballé le téléphone, allumez-le. Une succession d'écrans vous invite à vous identifier grâce à un compte Google si vous en possédez un. Cette identification permet à Android de configurer la messagerie et le calendrier de votre compte Google. Vous pouvez ignorer cette identification et configurer ultérieurement votre téléphone si vous le souhaitez.

Figure A–1
Associer un compte Google au téléphone



Une fois votre compte Google associé à votre téléphone de développement, vous serez guidé pour choisir vos préférences concernant vos informations personnelles. Ces choix sont modifiables plus tard si vous voulez revenir sur ce que vous avez indiqué.

Ensuite, à vous de définir l'heure et les différentes options de format. Appuyez enfin sur *Terminer l'installation* et votre téléphone est prêt à l'utilisation !

ASTUCE Utiliser un téléphone de développement sans forfait données

Le premier écran, au démarrage de votre téléphone, vous invite à enregistrer un compte Google pour pouvoir utiliser l'appareil. Évidemment cette opération n'est possible que si votre forfait est activé pour transmettre des données.

Pour éviter l'utilisation d'un forfait données, vous pouvez passer l'étape, d'enregistrement (via le bouton *ignorer*), paramétrer une connexion Wi-Fi et reprendre le processus (en lançant le client de messagerie GMail par exemple).

Mais vous pouvez également faire immédiatement apparaître le panneau de configuration Wi-Fi et franchir normalement les différentes étapes. Pour cela vous devez connecter votre téléphone à votre ordinateur, installer les drivers comme expliqué plus loin dans ce chapitre, en clair prendre un peu d'avance sur le déroulement normal de la lecture de ce livre !

Une fois la connexion à l'ordinateur configurée, nous allons utiliser l'outil *adb* qui se trouve dans le répertoire *tools* du kit de développement que vous venez d'installer. Nous reviendrons plus tard sur ses utilisations, l'urgence est de pouvoir utiliser votre téléphone !

```
adb shell
```

Une invite de commande commençant par $ apparaît. Saisir (sans retour à la ligne) :

```
am start -a android.intent.action.MAIN -n com.android.settings/
.Settings
```

Le panneau de configuration Wi-Fi apparaît, à vous de jouer !

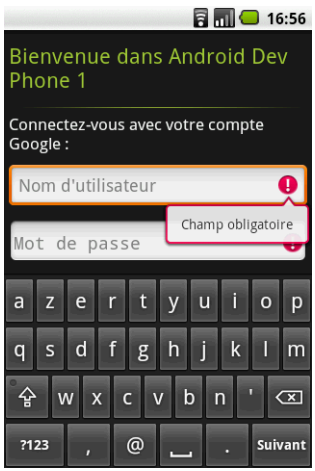


Figure A–2
Saisie des paramètres de votre compte Google

Connecter le téléphone à votre ordinateur

Maintenant que votre téléphone est fonctionnel, le temps est venu de le brancher à votre ordinateur. En effet pour pouvoir déboguer directement sur votre téléphone, vous devez le connecter à votre ordinateur. Cela se fait via le câble USB qui est fourni avec le téléphone. À noter, le téléphone est vu comme un périphérique USB spécifique qui nécessite des drivers particuliers (fournis dans le kit de développement). Nous allons aborder ce point maintenant.

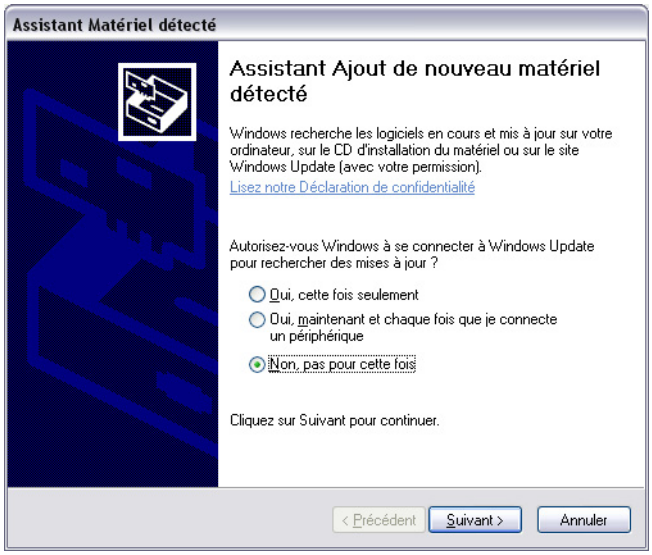
> **À SAVOIR Windows et les autres plates-formes**
>
> Ce chapitre ne traitera que de la plate-forme Windows. Pour déboguer sur les autres systèmes d'exploitation, reportez vous à la documentation Android disponible sur le site officiel.

Branchez le câble USB, fourni avec l'appareil, à votre ordinateur.

> **CONSEIL Choix d'un câble USB**
>
> Votre téléphone est vendu avec un câble USB. Utilisez-le en priorité. Nous avons pu lire sur Internet quelques mésaventures arrivées à des utilisateurs qui ont utilisé d'autres câbles USB de moins bonne qualité (notamment beaucoup plus fins). Le téléphone se mettait en charge mais n'était pas détecté par l'ordinateur. Penser à un problème de câble USB dans ce cas-là n'est pas totalement intuitif.

En prenant le cas de Windows (XP sur les captures d'écran), une fois le téléphone branché, un écran apparaît pour vous avertir qu'un nouveau matériel a été détecté. Il est tout à fait inutile de se connecter à *Windows Update :* choisissez donc *Non, pas pour cette fois* quand Windows vous propose une connexion pour rechercher les mises à jour.

Figure A–3
Nouveau matériel détecté



En sélectionnant *Suivant*, Windows vous demande un CD d'installation ou équivalent. Nous allons le faire pointer vers le répertoire d'installation du kit de développement où nous avons téléchargé la plate-forme 2.0 d'Android avec les drivers USB de l'appareil. Choisissez *Installer à partir d'un emplacement spécifié* dans la fenêtre.

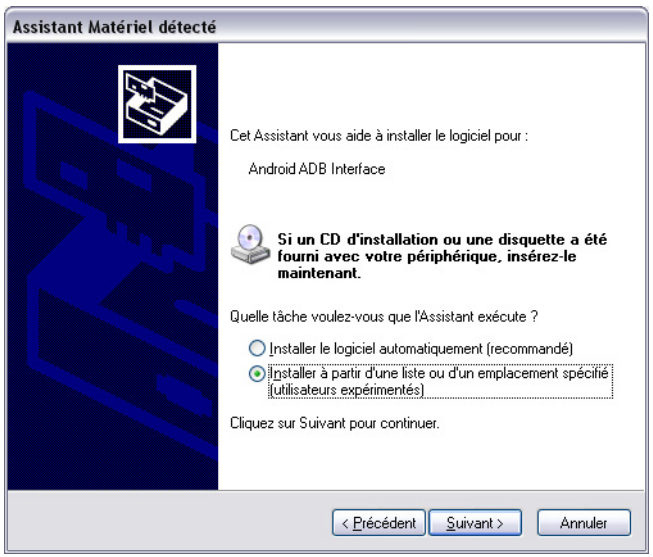


Figure A–4
Choisir un emplacement spécifique

Cliquez sur *Suivant*, l'installation démarre.

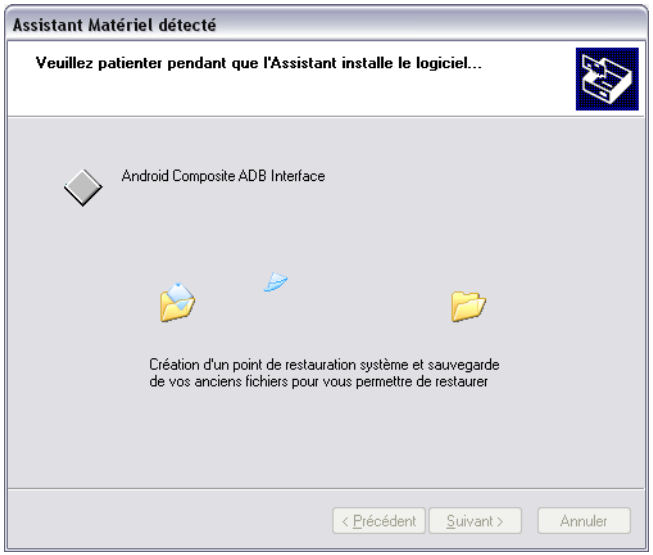


Figure A–5
Progression de l'installation

Une fois l'installation terminée, fermez la fenêtre. Windows reconnaîtra votre téléphone quand vous le brancherez grâce à un câble USB.

Pour pouvoir déboguer des programmes grâce à votre téléphone, il reste un réglage à réaliser. Déplacez-vous dans les interfaces de votre téléphone *Menu>Paramètres>Applications>Développement* et cochez débogage USB.

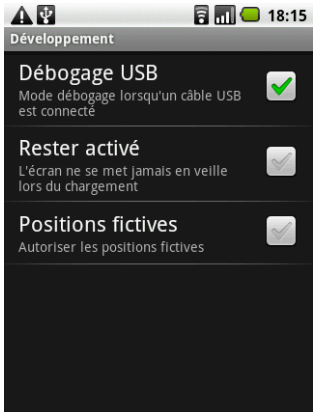


Figure A–6
Mode débogage configuré sur le téléphone

Côté téléphone, la barre de notification, en haut de l'écran, vous confirme la connexion USB et le mode de débogage.

En déroulant cette interface, vous pouvez même choisir de monter la carte mémoire de l'appareil, c'est à dire la transformer en stockage de masse type clé USB, où vous pourrez mettre de nouveaux fichiers (photos, sons, mises-à-jour du système, etc.). Pour cela, cliquez sur *USB connected*, puis sur le bouton *Monter*.

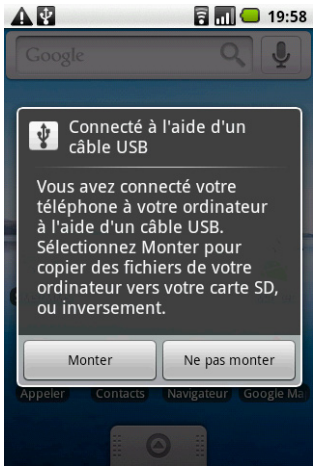


Figure A–7
Notification de connexion USB et utilisation de la carte du téléphone

Tout est prêt !

Utilisation de votre téléphone avec Eclipse

En fait, il n'y a rien de plus à faire pour lancer votre application sur le téléphone de développement. Il suffit que le câble USB soit branché entre votre téléphone et votre ordinateur et de lancer l'application, l'environnement de développement l'enverra automatiquement sur le téléphone.

Une fois plus avancé dans le livre et dans vos développements, vous pourrez modifier ce comportement choisissant l'émulateur à la place du téléphone. Pour cela il faut régler les paramètres dans le menu *Run>Run configurations...* Sélectionnez à gauche le projet concerné et allez dans l'onglet *Target*.

À noter que dans la vue *package explorer* d'Eclipse, un clic droit sur un projet Android vous permet de faire apparaître un *Run as ...>Android Application* sur lequel vous pouvez cliquer. Un écran apparaît alors vous invitant à choisir entre le téléphone si il est branché et l'émulateur (dans une des versions matérielles créées).

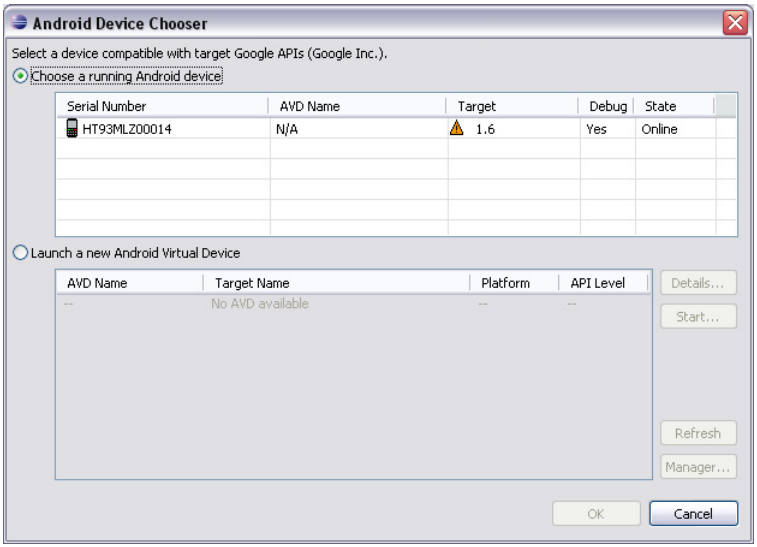


Figure A–8
Écran de sélection du moyen d'exécution à partir du menu Run as...

Si vous choisissez le mode *debug*, un message vous invitera à attendre quelques instants au lancement de l'application, le temps que l'environnement de développement paramètre correctement l'exécution du programme lancé.

Tout se déroule ensuite exactement comme avec l'émulateur et vice versa.

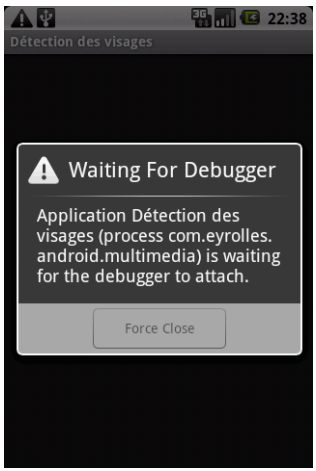


Figure A–9
Écran intermédiaire avant le lancement de l'application en mode debug

Si vous avez un doute, vous pouvez contrôler la présence de votre téléphone avec le menu *Windows>Show View>Other...>Android>Devices*. Dès que vous connecterez votre téléphone, l'affichage se mettra à jour et vous donnera en détails les processus qui s'exécutent sur votre téléphone.

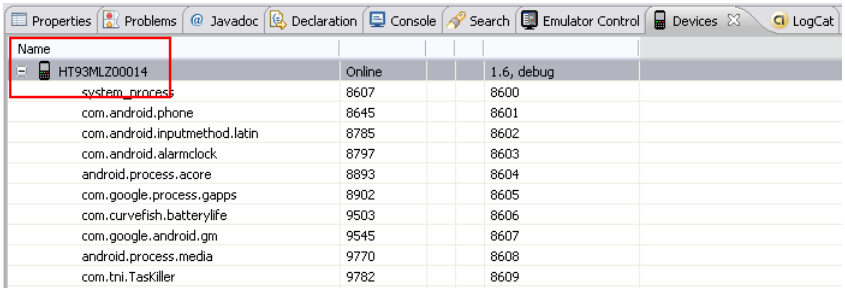


Figure A–10
Vue du plugin ADT concernant les appareils

> **ASTUCE** **Installer une application de source inconnue**
>
> Si vous souhaitez installer des applications dont la source est inconnue, c'est-à-dire ne provenant pas de l'Android Market, vous pouvez paramétrer votre téléphone pour les accepter. Naviguez dans *Menu>Settings>Applications* puis cochez la case *Unknown sources*.

Réglages cachés du navigateur

Afin de permettre aux développeurs d'être correctement outillé pour déboguer leurs applications, Android propose un menu de débogage qui est par défaut invisible des utilisateurs.

Pour activer ce menu caché, entrez l'adresse suivante dans la barre d'adresse du navigateur d'Android : about:debug. Une fois cette opération réalisée, naviguez dans *Menu>More>Settings* dans le navigateur et si vous descendez tout en bas, vous devriez voir de nouveaux réglages ou actions disponibles.

Prenons comme exemple le sous-menu *UAString* (User-Agent), qui correspond à la chaîne qui identifie le type d'appareil utilisant le navigateur. Les sites Internet peuvent récupérer cette information et savoir qu'un appareil mobile se connecte et ainsi adapter les pages qu'ils proposent au téléphone ou à la tablette. Ce paramètre sert beaucoup également en termes de statistiques pour calculer les parts de marché.

Figure A–11
Exemple de réglage caché

Ici, comme vous pouvez le voir sur la capture d'écran, vous pouvez vous faire passer pour un navigateur de système bureautique, c'est à dire un ordinateur de travail (pour éviter d'obtenir des pages adaptées) ou pour un iPhone !

Utiliser le concepteur graphique dans Eclipse pour réaliser des interfaces graphiques

Le module ADT (*Android Developement Tools*) d'Eclipse fournit un grand nombre d'outils, que ce soit au niveau de l'édition, de la structure du projet, de la compilation, du débogage ou des différentes fenêtres de génération. Parmi ces outils, il en existe un vous permettant de visualiser et d'éditer l'interface utilisateur de vos activités.

Bien qu'à l'origine cet outil était destiné aux débutants dans ses premières versions en raison de limites techniques, les dernières versions d'ADT prennent maintenant en compte les composants personnalisés dans le rendu visuel et en font maintenant un réel outil utilisable par les développeurs avancés.

Cet outil, nommé *concepteur* ci-après, peut vous faire gagner un temps non négligeable pour concevoir rapidement vos interfaces sans avoir à tester chaque modification dans l'émulateur ou sur un téléphone.

Le concepteur est disponible lorsque vous éditez une déclaration d'interface utilisateur, c'est à dire un fichier XML localisé dans `/res/layout` de votre projet. Sélectionnez l'onglet *Layout* et non `<nom_layout>.xml` en bas de la vue d'édition pour disposer de l'aperçu en temps réel.

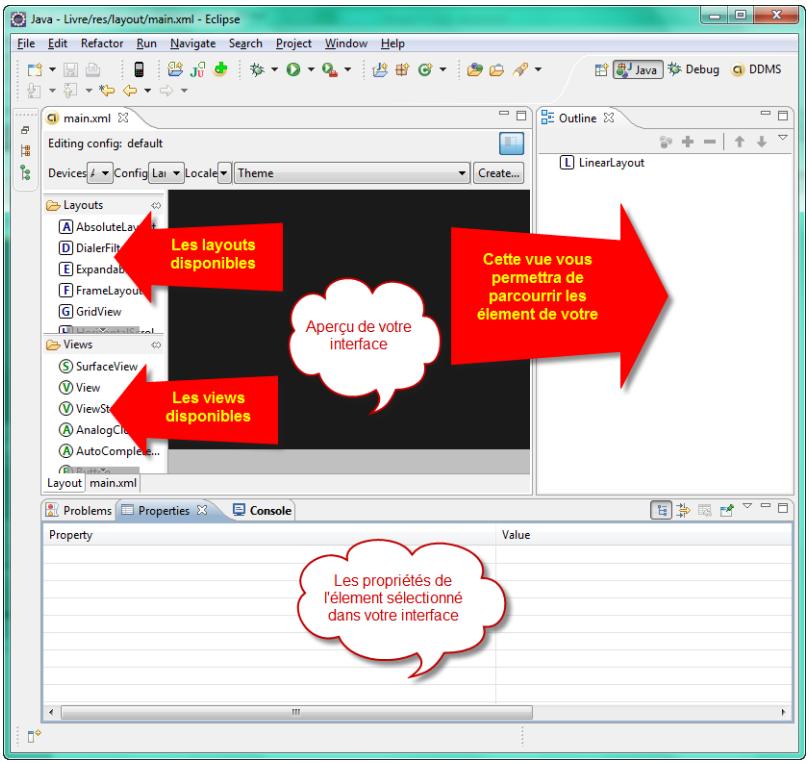


Figure A–12
Vue du concepteur d'interface utilisateur dans Eclipse

Dans la partie supérieure gauche, vous trouverez tous les types de gabarits de mise en page disponibles que vous pourrez intégrer dans votre interface. Dans la partie inférieure gauche, vous trouverez tous les types de vues que vous pouvez intégrer dans votre interface.

Naviguer dans le concepteur d'interfaces

Sur la droite se trouve la fenêtre *Outline* qui est un aperçu de notre schéma XML. Cette fenêtre fournie par Eclipse est très importante car dans certains cas elle vous

permettra d'accéder à des éléments de votre interface qui ne le seront pas via la zone centrale, qui comporte l'aperçu de l'interface. En bas de la fenêtre, vous trouverez la liste des propriétés de l'élément sélectionné dans votre interface. Cet affichage est également très important car il vous permettra de voir toutes les propriétés disponibles pour l'élément sélectionné.

Par exemple, en sélectionnant le `LinearLayout` à droite, nous pouvons voir toutes les propriétés modifiables pour cet élément.

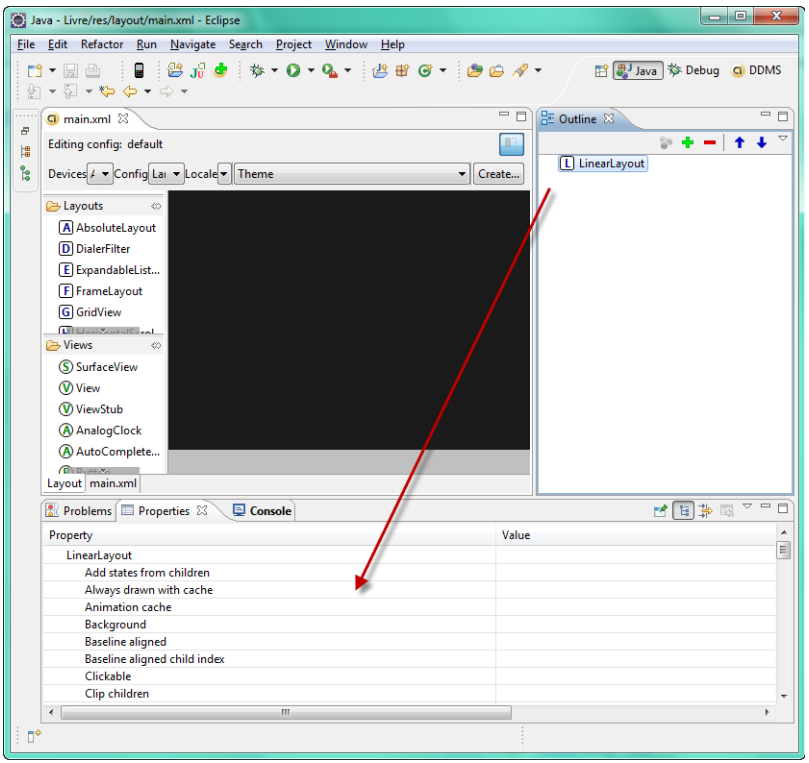


Figure A–13
Propriétés d'une vue dans Eclipse

En faisant défiler les propriétés, vous retrouverez des propriétés que nous avons abordées dans ce livre, comme *Layout width* et *Layout height* qui vous permettent de définir la taille de votre `LinearLayout`.

En passant la souris sur une des propriétés, une infobulle descriptive s'affiche.

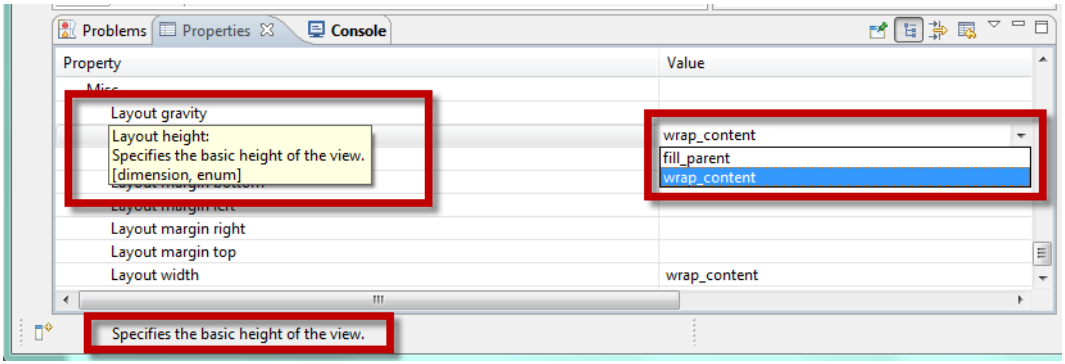


Figure A–14 Info bulle d'une propriété dans l'éditeur de propriétés d'une vue

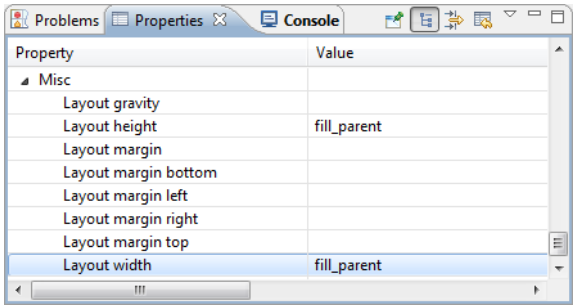
Si vous sélectionnez une des propriétés, la description de cette propriété s'affichera.

Modifier les propriétés des éléments graphiques

Certaines propriétés comme *Layout width* et *Layout height* peuvent avoir des choix prédéfinis et de ce fait proposer une liste de choix. Néanmoins dans ce cas précis, le choix n'est pas exclusif ; vous pourrez préciser une taille en pixel ou dpi. Nous vous recommandons de préférer les tailles en dip (dip) plutôt que px (pixel).

Mettez `fill_parent` sur les propriétés `Layout width` et `Layout height` pour occuper tout l'espace disponible sur l'écran.

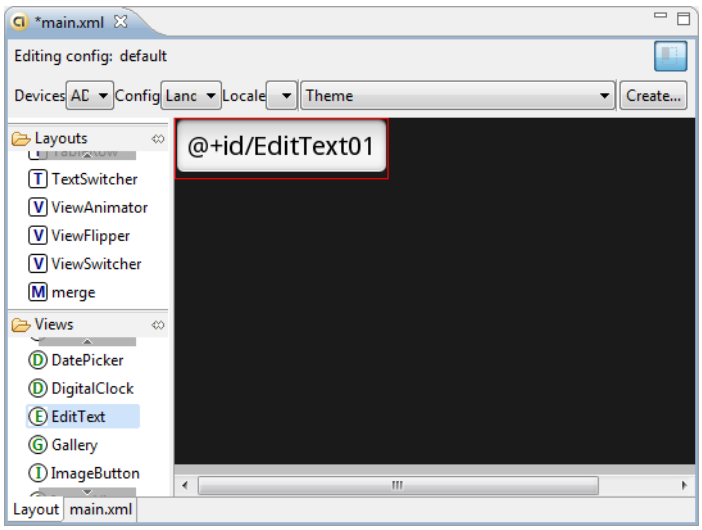
Figure A–15
Changer de propriétés dans l'éditeur de propriétés d'une vue



Créer une interface grâce au concepteur

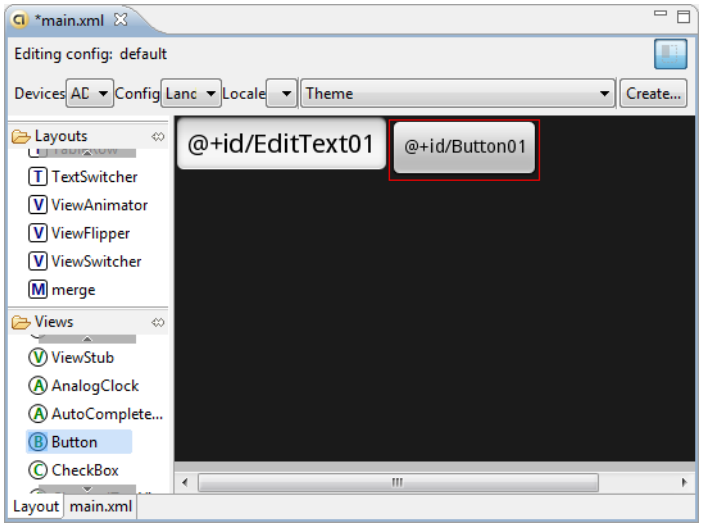
Pour ajouter un `EditText`, sélectionnez-le sur la gauche dans la liste des *Views* et faites le glisser sur la zone noire centrale.

Figure A–16
Ajout d'une vue de type éditeur de texte dans l'interface utilisateur



Ajoutez maintenant un bouton de la même façon. Sélectionnez *Button* dans la liste des vues et faite le glisser sur la zone centrale.

Figure A–17
Ajout d'une vue de type bouton dans l'éditeur d'interface utilisateur



Ces deux éléments s'affichent l'un à côté de l'autre. Si vous souhaitez les afficher l'un au dessus de l'autre, vous devrez changer l'orientation du `LinearLayout` parent de ces deux éléments.

Sélectionnez le fond noir ou bien sur la droite dans la fenêtre *Outline*, sélectionnez le `LinearLayout` qui se trouve à la base de notre arborescence.

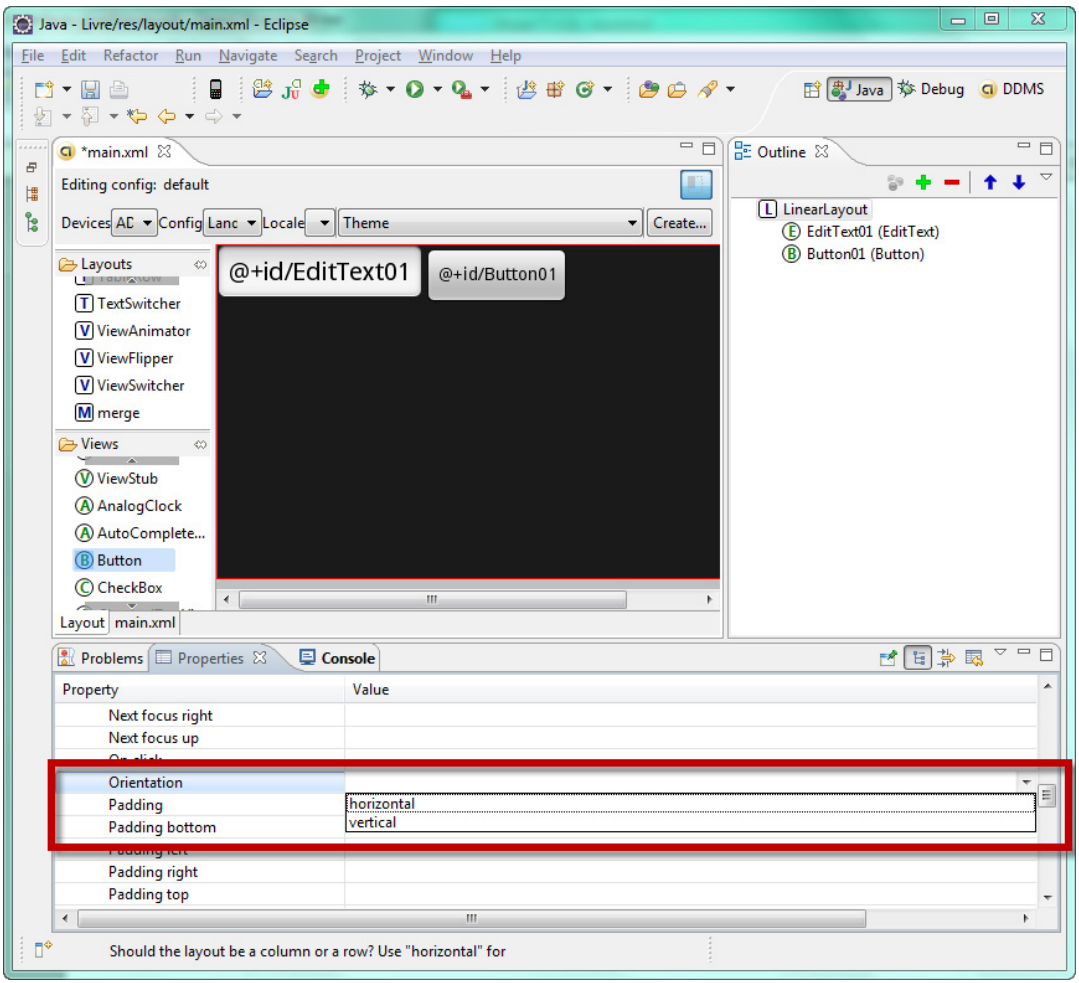


Figure A–18 Changement de l'orientation du LinearLayout

Deux choix sont alors à votre disposition : *Horizontal* et *Vertical*. Par défaut les `LinearLayout` dispose les éléments à l'horizontal, même si la propriété n'est pas spécifiée. Sélectionnez *Vertical* afin d'afficher les vues enfants les unes en dessous des autres. L'aperçu se met à jour immédiatement.

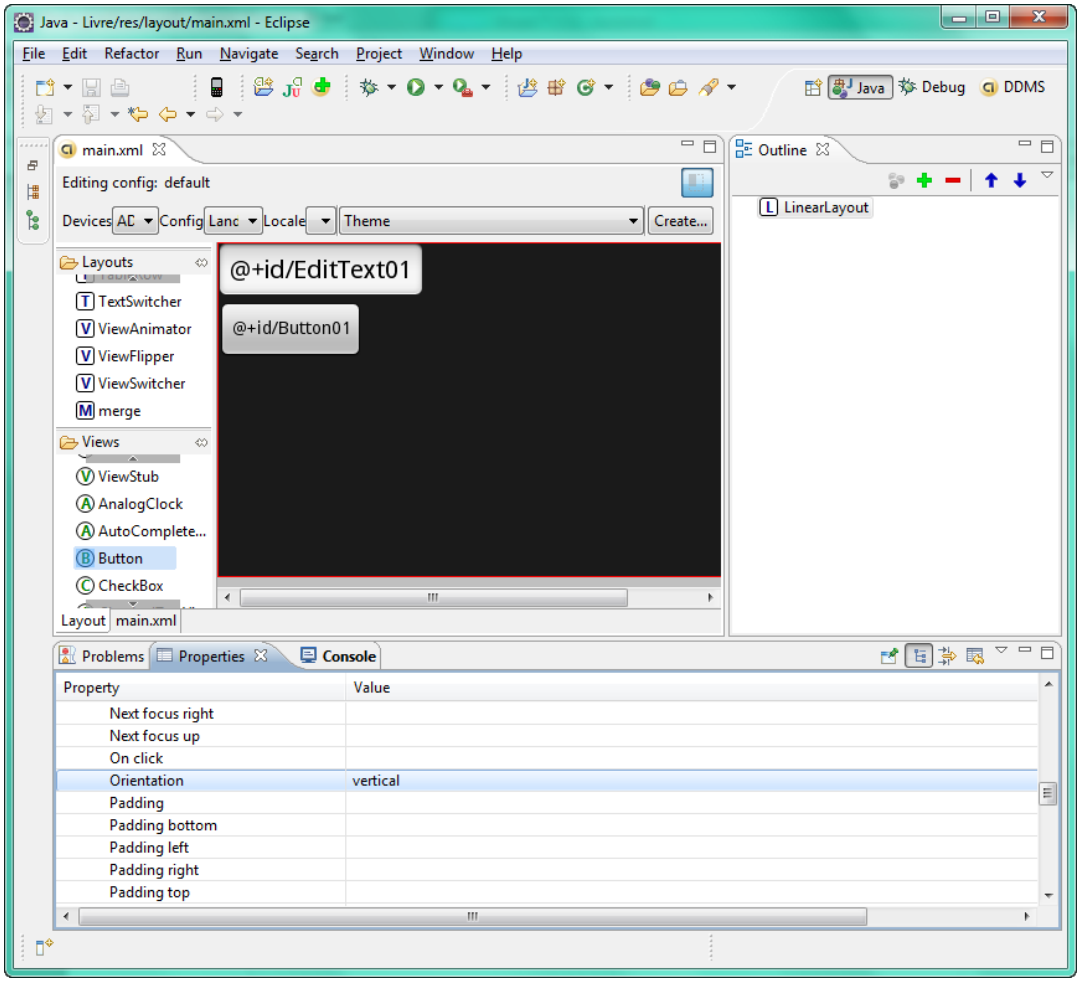


Figure A–19 Alignement vertical des vues du LinearLayout

Pour les interfaces complexes composées d'une multitude de `LinearLayout`, vous pourrez difficilement sélectionner les éléments directement dans l'aperçu au centre. Pour accéder aux éléments de l'interface, vous devrez alors utiliser la fenêtre *Outline* et naviguer dans l'arborescence pour sélectionner l'élément dont vous souhaitez changer les propriétés.

Vous pouvez à tout moment consulter et modifier manuellement le XML en mode texte en cliquant sur l'onglet en dessous de l'aperçu.

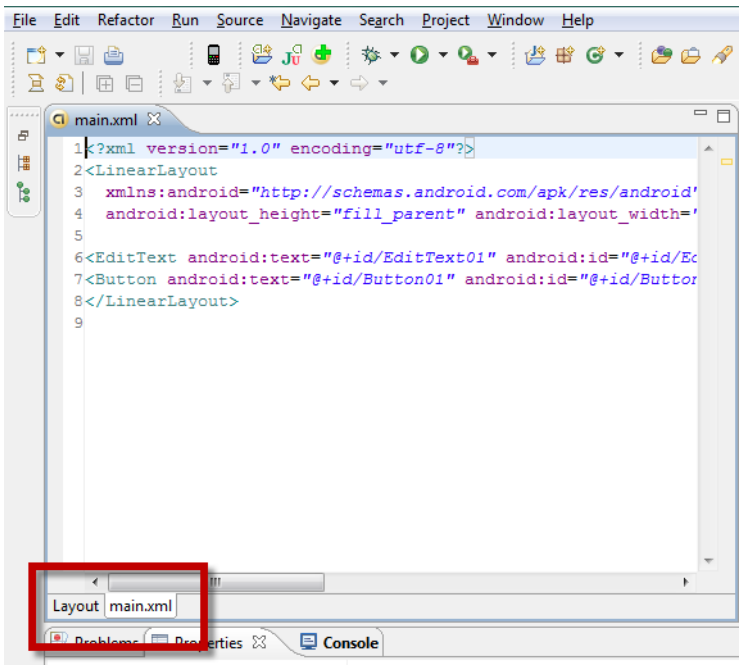


Figure A–20
Passer du concepteur graphique au source XML via les onglets

De cette façon, vous pourrez basculer à tout moment sur le concepteur graphique en cliquant sur *Layout* et voir l'aperçu des changements effectués dans le XML.

Vous voilà prêt à utiliser le concepteur d'interfaces graphique qui vous permettra de gagner encore plus en productivité grâce à ADT et Eclipse.

Index