



Un robot parleur sous Android

Parmi les nombreuses fonctionnalités du SDK d'Android il existe un service de synthèse vocale qui vous permet de faire dire à votre téléphone ce que vous voulez dans plusieurs langues. Les applications Android peuvent facilement tirer parti de cette API afin d'aider les personnes mal ou non voyantes, ou permettre de lire à voix haute un SMS qui arrive, lorsque vous conduisez votre voiture par exemple. Dans cet article nous allons nous intéresser à la mise en œuvre d'une petite application de robot parleur qui prononce dans la langue choisie un message saisi au clavier.

L'application est composée d'une activité principale proposant une IHM pour saisir le texte, et d'une activité secondaire accessible depuis un menu d'options, permettant de modifier les réglages de notre application. Les constantes de l'application sont centralisées dans l'interface Constants, tandis que les fonctions de synthèse vocale sont encapsulées dans la classe Tts. Des fichiers de ressources sont utilisés pour définir l'IHM de l'application (layout main.xml), le menu d'options (options_menu.xml) et les différentes options de réglages (settings.xml). Tous les textes de l'application sont externalisés dans values/strings.xml, c'est la ressource utilisée par défaut ; pour une version bilingue il suffit de créer une ressource values-fr/strings.xml avec les textes en français [Fig.1]. Manifeste de l'application :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/
android" android:versionCode="1" android:versionName="1.0"
package="com.programmez.android.speechbot">
  <application android:icon="@drawable/icon" android:label
="@string/app_name">
    <activity android:name=".App" android:label="@string/app_name">
```

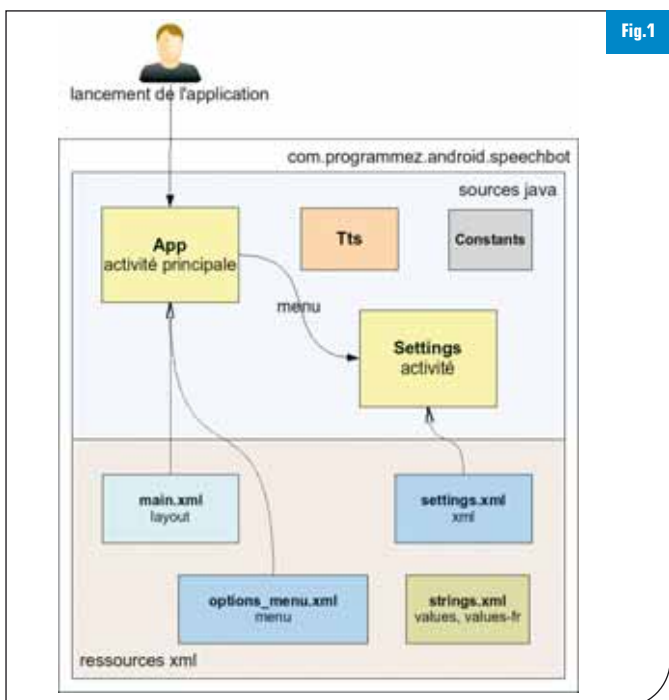


Fig.1

```
<intent-filter>
  <action android:name="android.intent.action.MAIN"/>
  <category android:name="android.intent.category.LAUNCHER"/>
</intent-filter>
</activity>
<activity android:name=".Settings"/>
</application>
<uses-sdk android:minSdkVersion="4" android:targetSdkVersion
="7"/>
</manifest>
```



Fig.2

Activité principale et IHM

L'interface de l'application est constituée d'un titre, une invite à saisir, une zone d'édition et un bouton pour lancer la synthèse vocale.

[Fig.2]. L'activité principale est prise en charge par la classe App, qui affiche le layout et

gère les interactions avec l'utilisateur :

- La première fois que l'utilisateur lance l'application, un texte par défaut lui est proposé (« Bonjour ! »)
- Chaque fois que le texte est modifié, il est sauvegardé et remplacera le texte par défaut pour les prochains lancements de l'application.
- Lorsque l'utilisateur clique sur le bouton « Parler », l'activité lance la synthèse vocale.
- Un menu d'options est accessible via la touche « menu » du téléphone, avec la possibilité d'accéder aux réglages ou de quitter l'application.

Source de App.java :

toutes les sources sont disponibles sur le site de Programmez !. La méthode onCreate est invoquée à la création de l'activité lorsque l'application est lancée depuis le bureau; elle se contente de récupérer une instance du service TTS, de définir le layout puis de construire la zone de texte et le bouton.

Ces deux widgets sont associés à des écouteurs :

- pour la zone de texte : la méthode onTextChanged est invoquée à chaque modification de texte; c'est l'endroit idéal pour effectuer la sauvegarde du texte.
- pour le bouton : à chaque clic on lance la synthèse vocale, qui est déléguée à la méthode speech,

Les méthodes `onOptionsItemSelected` et `onOptionsItemSelected` prennent en charge le menu d'options :

- `onOptionsItemSelected` crée le menu : on a choisi ici de définir le menu dans un fichier XML plutôt qu'en java ; d'où l'utilisation de `MenuInflater`.
- `onOptionsItemSelected` gère la sélection parmi les deux options :
 - option « Réglages » : lancement d'une activité fille (la deuxième activité du projet) via une intention.
 - option « Quitter » : on quitte l'application en terminant simplement l'activité.

Pour identifier chaque option, on utilise la fameuse classe « magique » `R` générée à la volée à partir des fichiers de ressources. Le lancement de l'activité fille (`Settings`) s'effectue avec `startActivityForResult` car on souhaite être informé de sa fin (retour à l'activité principale une fois les réglages terminés).

L'événement de retour se matérialise par l'invocation de la méthode `onActivityResult`, quelle que soit la provenance.

S'il était utile de connaître la provenance, il suffirait de vérifier la valeur du paramètre `requestCode` (ici notre constante `MENU_SETTINGS_REQUEST_CODE`); on se contente ici de rafraîchir la zone de texte afin de gérer un éventuel changement effectué par l'activité fille. Enfin, on redéfinit la méthode `onDestroy` pour libérer proprement les ressources du service TTS avant de détruire l'activité.

Constantes du projet :

```
package com.programmez.android.speechbot;
public interface Constants {
    String APP_NAME = «SpeechBot»;
    String LOG_TAG = APP_NAME;
    int MENU_SETTINGS_REQUEST_CODE = 1;
    int TTS_CHECK_REQUEST_CODE = 2;
    String PREF_KEY_CHECK_TTS = «PREF_KEY_CHECK_TTS»;
    String PREF_KEY_EDIT_TEXT = «PREF_KEY_EDIT_TEXT»;
    String PREF_KEY_LANGUAGE = «PREF_KEY_LANGUAGE»;
    String PREF_KEY_RESET = «PREF_KEY_RESET»;
    String PREF_KEY_SPEECH_RATE = «PREF_KEY_SPEECH_RATE»;
    int TTS_QUEUE_FLUSH = 0;
    int TTS_QUEUE_ADD = 1;
}
```

Le menu d'options de l'application est défini dans la ressource `res/menu/options_menu.xml` :

```
<?xml version=»1.0« encoding=»utf-8»?>
<menu xmlns:android=»http://schemas.android.com/apk/res/android«>
    <item android:title=»@string/settings« android:icon=»@android:drawable/ic_menu_preferences« android:id=»@+id/options_menu_settings«/>
    <item android:title=»@string/quit« android:id=»@+id/options_menu_quit« android:icon=»@android:drawable/ic_menu_close_clear_cancel«/>
</menu>
```

[Fig.3]



Le menu d'options s'affiche en appuyant sur la touche « Menu » du téléphone.

Activité secondaire : réglages de l'application

L'activité secondaire est implémentée par la classe `Settings`.

Étant donné sa vocation, elle étend la classe `PreferenceActivity` pour bénéficier de comportements standard de gestion de préférences; nul besoin de définir une IHM, tout est automatique.

L'ensemble des réglages (préférences) est défini dans un fichier de ressources XML, avec :

- Le choix de la langue
- Le choix de la vitesse de diction
- La possibilité de vérifier la disponibilité du service TTS
- La possibilité de réinitialiser tous les réglages

Source de la ressource `res/xml/settings.xml` :

```
<?xml version=»1.0« encoding=»utf-8»?>
<PreferenceScreen xmlns:android=»http://schemas.android.com/apk/res/android«>
    <ListPreference android:key=»PREF_KEY_LANGUAGE« android:title=»@string/language« android:summary=»@string/language_summary« android:persistent=»true«/>
    <ListPreference android:summary=»@string/speech_rate_summary« android:title=»@string/speech_rate« android:key=»PREF_KEY_SPEECH_RATE« android:entryValues=»@array/speech_rate_values« android:entries=»@array/speech_rates« android:defaultValue=»100«/>
    <Preference android:title=»@string/check_tts« android:key=»PREF_KEY_CHECK_TTS« android:summary=»@string/check_tts_summary«/>
    <Preference android:key=»PREF_KEY_RESET« android:summary=»@string/reset_summary« android:title=»@string/reset«/>
</PreferenceScreen>
```

Chaque préférence est associée à une clef qui sera utilisée pour la persister et qui servira de donnée discriminante aux traitements java.

Source de `Settings.java`, explication :

Les trois méthodes statiques sont de simples helpers, qui auraient pu classiquement appartenir à une classe « Util » (on économise une classe). La méthode `onCreate` récupère une instance du service TTS, ajoute les préférences depuis le fichier de ressources, et construit la liste des langues disponibles. Les deux dernières lignes associent des écouteurs aux options simplement cliquables : la vérification du service TTS et la réinitialisation. La méthode `buildAvailableLanguages` définit la liste des langues qui sera proposée, avec comme premier choix la langue par défaut. C'est la classe `Tts` qui fournit les langues effectivement supportées.

Dans la méthode `onPreferenceClick`, on effectue le traitement correspondant à la clé de l'option sélectionnée :

- Vérification du service TTS : on invoque la méthode `checkTtsData` qui lance une activité de vérification des données propre au service Tts, on récupère ensuite le code résultat dans la méthode `onActivityResult` que l'on transmet à nouveau au service Tts pour déléguer la vérification. Si tout est opérationnel on affiche un message.
- Réinitialisation : on récupère les préférences de l'application, on efface tout puis on termine l'activité (les préférences ne sont plus à jour).

[Fig.4, 5, 6]

L'activité `PreferenceActivity` du SDK d'Android présente des avantages certains de simplicité et rapidité de mise en œuvre ; elle



répond à la majorité des besoins mais fournit un look & feel basique, et ne conviendra donc pas toujours à des besoins plus exotiques.

La synthèse vocale : service TTS (Text-to-speech)

Pour effectuer la synthèse vocale, le SDK d'Android fournit en standard un module de Text-to-speech à partir de la version 1.6 (Donut). Notre classe Tts sert de façade dans le projet pour accéder à ce service technique du SDK. La classe Tts implémente le pattern singleton (methode getInstance), pour faciliter le partage du service dans les différentes activités quel que soit leur cycle de vie. La constitution de la liste des langues supportées est assurée par la méthode obtainAvailableLocales qui se contente de renvoyer « en dur » deux langues. Pour obtenir dynamiquement la liste des langues réellement supportées on pourrait faire évoluer cette méthode en respectant le principe suivant :

- Récupération des langues du système Android,
- Pour chaque Locale, vérification de sa disponibilité au sein du module TTS (méthode isLanguageAvailable),
- Tri et nettoyage des doublons dans la liste.

La méthode checkTtsData permet de vérifier les données du module TTS en lançant une activité dédiée (ACTION_CHECK_TTS_DATA).

La méthode onCheckTtsDataResult interprète le résultat de l'activité et lance l'installation du module TTS si le test a échoué. Les autres méthodes jouent leur rôle de façade et utilisent l'API du module TTS pour définir la langue (setLanguage), définir la vitesse (setSpeechRate), libérer les ressources (shutdown), et parler (speak). La méthode speak peut être invoquée pour ajouter un texte à la file d'attente (mode QUEUE_ADD) ou le dire immédiatement et vider la file (mode QUEUE_FLUSH). En cas d'erreur du module TTS, une exception spécifique est levée :

```
package com.programmez.android.speechbot.tts;

public class TtsNotInitializedException extends Exception {
    private static final long serialVersionUID = -8307341316965013306L;
}
```

Problématique de fragmentation : TTS avec Cupcake ?

Le service mis en œuvre s'appuie sur le module intégré à Android depuis la v1.6 (Donut). Tous les androphones récents propulsés par Donut ou Eclair (v2.0-2.1) seront donc compatibles.

Mais ceux qui sont propulsés par une version antérieure comme Cupcake (v1.5), encore bien présente sur le marché malheureusement, poseront problème (Samsung Galaxy par exemple).

[Fig.7 et 8]

Comment faire fonctionner l'application avec Cupcake ?

Il se trouve qu'un module tiers existe via le projet Google « eyes-free » : <http://code.google.com/p/eyes-free/>

En fait, le module intégré au SDK est postérieur à ce projet qui continue d'évoluer.

Les livrables se présentent sous deux formes :

- Une application pour les utilisateurs : « Text-To-Speech Extended »
- Une librairie (.jar) pour les développeurs : http://eyesfree.googlecode.com/files/TTS_library_stub_1.7_market.jar



Fig.9

[Fig.9]

L'API fournie par la librairie est relativement similaire au module standard Donut, au nom de package près et à quelques petites différences non bloquantes. En revanche, le rendu vocal n'est pas du tout le même, c'est un homme qui parle au lieu d'une femme et la voix est beaucoup plus « robotisée »... Les réglages intrinsèques du module sont accessibles via l'application Text-To-Speech Extended, au lieu du panneau de configuration standard.

Intégration de Text-To-Speech Extended

L'intégration de ce module à notre application est un moindre mal puisque nous bénéficions d'une façade; il reste néanmoins à savoir si l'application doit régresser en 1.5 ou rester au-dessus, et savoir déterminer dynamiquement la version du téléphone hôte pour choisir le bon module.

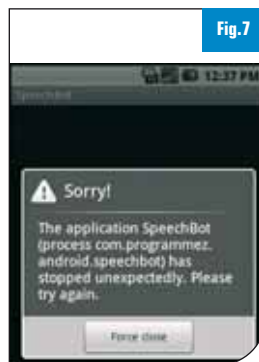


Fig.7

Le passage en v1.5 du projet dans Eclipse génère des erreurs liées à la perte des API supérieures.

Une première solution consiste à utiliser la réflexion pour instancier dynamiquement



Fig.8

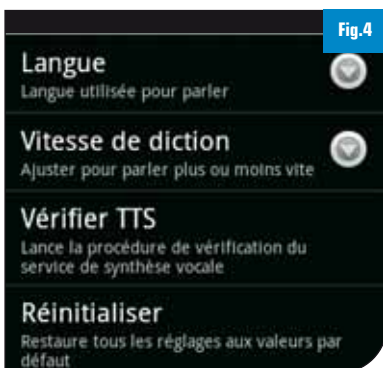


Fig.4



Fig.5

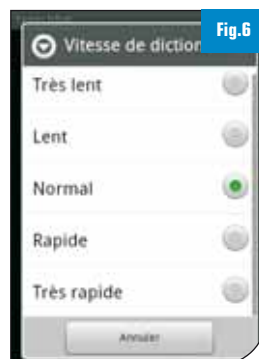


Fig.6

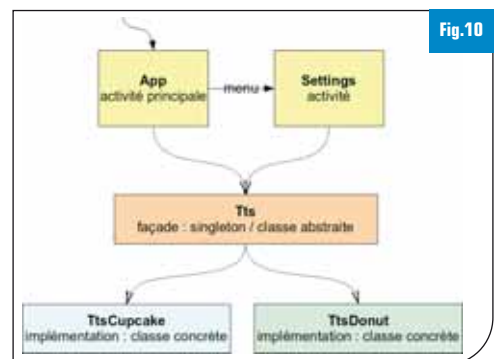


Fig.10

quement les classes nécessaires, invoquer leurs méthodes, accéder aux champs.

C'est techniquement tout à fait envisageable, mais le code source résultant y perdrait en qualité (compilateur impuissant), et serait pénible à maintenir, c'est un euphémisme.

Une autre solution consiste à laisser le projet en version haute (v2.1 par exemple), et changer dans le manifeste l'attribut `minSdkVersion` (3 pour Cupcake).

```
<uses-sdk android:minSdkVersion=>3> android:targetSdkVersion=>7 />
```

Il faut ensuite modifier la classe `Tts` pour la dissocier de l'implémentation effective; pour cela une classe abstraite est tout à fait appropriée. [Fig.10]. Voici les modifications à appliquer :

```
package com.programmez.android.speechbot.tts;
import java.util.*;
import android.app.Activity;
import android.content.Context;
import com.programmez.android.speechbot.App;
public abstract class Tts {
    protected static Tts instance;
    public static Tts getInstance(final Context context, final boolean initialize) {
        if (instance == null || initialize) {
            try {
                instance = new TtsDonut(context);
            } catch (final VerifyError e) {
                instance = new TtsCupcake(context);
            }
        }
        App.log("tts instance : « + instance.getClass().getSimpleName();
        return instance;
    }
    protected boolean initialized;
    protected final Context context;
    protected Tts(final Context context) {
        this.context = context;
        initTtsImpl();
    }
    public abstract void checkTtsData(final Activity activity, int requestCode);
    public abstract boolean onCheckTtsDataResult(final Activity activity, final int resultCode);
    public void setLanguage(final String language) throws TtsNotInitializedException {
        checkTtsImpl();
        setLanguageImpl(language == null || «».equals(language) ? Locale.getDefault() : new Locale(language));
    }
    public abstract void setSpeechRate(final float speechRate);
    public void shutdown() {
        App.log("tts shutdown");
        shutdownTtsImpl();
    }
}
```

```
    initialized = false;
    instance = null;
}
public void speak(final String text, final int queueMode) throws TtsNotInitializedException {
    checkTtsImpl();
    speakImpl(text, queueMode);
}
protected void checkTtsImpl() throws TtsNotInitializedException {
    if (getTtsImpl() == null || !initialized)
        throw new TtsNotInitializedException();
}
protected abstract Object getTtsImpl();
protected abstract void initTtsImpl();
protected abstract Map<String, Set<Locale>> obtainAllTtsSupportedLocales();
protected abstract void setLanguageImpl(Locale locale);
protected abstract void shutdownTtsImpl();
protected abstract void speakImpl(String text, int queueMode);
}
```

A noter dans la méthode `getInstance` : l'utilisation de l'exception `VerifyError` pour choisir l'implémentation en fonction de la version d'Android. Deux autres classes concrètes hériteront de la classe `Tts` et prendront en charge les deux différents modules implémentés en fonction de la version d'Android. Le contrat du module TTS vu par l'application est fixé par notre classe abstraite : le découplage est bien assuré. La classe `TtsDonut` est équivalente à notre ancienne implémentation, c'est quasiment un déplacement de code. La classe `TtsCupCake` implémente les fonctionnalités de synthèse vocale avec le module tiers Text-To-Speech.

Évidemment, les API utilisées apparaissent comme dépréciées dans le source, mais dans cette classe particulière on s'autorise exceptionnellement l'utilisation de l'annotation `SuppressWarnings`. On retrouve des situations analogues qui nécessitent l'emploi de cette technique pour gérer proprement les différentes versions d'Android : gestion des contacts, lecture de SMS ...

Conclusion

La façade `Tts` peut facilement évoluer pour proposer plus de possibilités, par exemple intégrer vos propres samples vocaux ou encore choisir le moteur de synthèse. Ce système pourra très simplement être intégré à n'importe quelle application Android pour apporter un plus aux utilisateurs. En outre, la façade permet de gérer correctement la problématique de fragmentation en s'appuyant sur un service de synthèse vocale tiers (eyes-free) dans le cas d'un androphone Cupcake qui n'intègre pas l'API `TextToSpeech` standard de Donut.

Pour les plus courageux, une application Android de traduction simultanée pourrait être imaginée moyennant l'ajout de fonctions de reconnaissances vocales et l'intégration d'une API du type `Google Traduction` ;-)

■ Olivier Penhoat

Consultant & Formateur
Valtech Training