

Programmation Android

Debugger ses applications

Jérémy VINET

IE-Concept 2017



Debugger ses applications

- Le concept de débogage
- DDMS
- Utilisation d'un logeur
- Utilisation du débogueur (ADB)
- Ressources



Le concept débogage

- Le débogage d'une application permet au développeur de tester et analyser les réactions de l'application en fonction de son utilisation.
- Il est couramment admis qu'un développeur va passer au moins 50% de son temps de développement à débogger son application.
- Depuis l'avènement d'internet et le partage massif d'application, il est devenu inconcevable pour un éditeur d'application de ne pas corriger/améliorer son application dans le temps.



Le concept débogage

- **Avec le temps le concept de débogage a évolué afin de correspondre aux nouveaux usages des applications.**
 - Alpha et beta test
 - Sortie de version preview public
 - Envoi de reporting vers un serveur
 - Bug tracker
- **Si le concept a évolué, la finalité reste la même :**
 - Recherche de bug.
 - Amélioration ergonomique.
 - Amélioration technique/stabilité.
 - Test de nouveau modèle.



Le concept débogage

- **Dans le cadre d'une application Android le débogage peut se faire de plusieurs méthodes :**
 - Utilisation d'un logeur pour avoir une traçabilité de l'application.
 - Utilisation du débogueur intégré (ADB).
 - Retour des utilisations de l'application via les commentaires sur le marché.
 - Test Unitaire et test d'intégration.
 - Android Profiler
- **Dans le cadre de ce cours nous verrons seulement les deux premières méthodes.**
 - Logeur
 - ADB



DDMS

- **DDMS : Dalvik Debug Monitor Server.**
- **C'est une application graphique de profilage d'applications Android.**
- **DDMS est présente dans le SDK Android.**
- **Android studio intègre parfaitement DDMS pour permettre le débogage des applications plus facilement.**
- **Possibilité de le lancer en mode « lignes de commandes ».**



DDMS

- **Parmi les outils de DDMS, nous pouvons citer :**
 - La vue « Devices » : elle permet de sélectionner le device (virtuel ou non) à monitorer.
 - La vue « LogCat » : elle présente les logs du device sélectionné dans la vue « Devices ».
 - La vue « Console » : elle présente des informations sur l'application en cours d'exécution pour le device sélectionné.
 - La vue « FileExplorer » : elle permet de naviguer dans le système de fichiers du device sélectionné.
 - La vue « Emulator » : elle permet de contrôler votre émulateur.
 - Par exemple, elle permet de contrôler les données de localisation, ...
- **DDMS est un composant d'un outil qui s'appelle Android Device Monitor qui lui-même permet le débogage d'application en profondeur.**



Utilisation d'un logeur

- Qu'est ce qu'un logeur ?
- Le logeur d'Android Logcat



Qu'est ce qu'un logeur ?

- Un logeur est soit une classe soit un service qui a pour but de lister et mémoriser des évènements.
- Il permet de voir le déroulement de l'application et le cas échéant les problèmes lors de l'exécution de celle-ci.
- Il n'a pas pour but d'apporter une fonction en plus à l'utilisateur final.
- L'application doit pouvoir fonctionner sans ressentir de différences que le logeur soit activé ou éteint.



Qu'est ce qu'un logeur ?

- En règle générale le logeur est vu comme un tableau ou chaque ligne représente un évènement ou une action que l'application a exécuté.
- Chaque ligne doit contenir des informations utiles permettant d'analyser l'application.
 - Date et heure précise de l'évènement
 - Un nom humainement compréhensible, par le développeur, de l'évènement
 - Des informations concernant l'évènement (variables, choix, ..)
- Le logeur doit avoir un flux de sortie (console, fichier, mail, ...) afin que l'on puisse accéder à ces données.



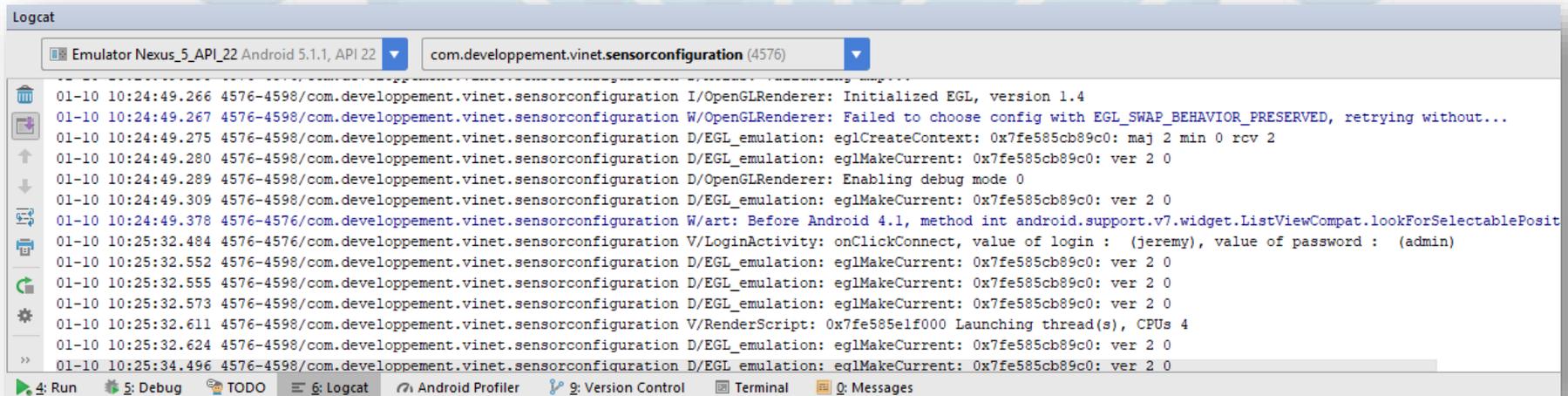
Qu'est ce qu'un logeur ?

- Le logeur étant un outil indispensable pour un développeur d'application, une classe Log a été intégrée dans le SDK Android.
- Cette classe de log permet la remonté des informations de logeur dans une console dédiée (DDMS -> LogCat).
- Elle fonctionne aussi bien avec le simulateur qu'avec un périphérique réel.



Le logeur d'Android Logcat

- Le SDK Android intègre la classe « Log » qui permet la remonté d'informations dans la console « Logcat ».
- La console Logcat affiche toutes les informations provenant du périphérique, pas juste celle de l'application en cours.



The screenshot shows the Logcat window in Android Studio. The top bar indicates the device is an Emulator Nexus_5_API_22 (Android 5.1.1, API 22) and the package is com.developpement.vinet.sensorconfiguration (4576). The log entries show various system messages, including OpenGL initialization, EGL emulation, and application-specific logs like LoginActivity and RenderScript.

```
Logcat
Emulator Nexus_5_API_22 Android 5.1.1, API 22 com.developpement.vinet.sensorconfiguration (4576)
01-10 10:24:49.266 4576-4598/com.developpement.vinet.sensorconfiguration I/OpenGLRenderer: Initialized EGL, version 1.4
01-10 10:24:49.267 4576-4598/com.developpement.vinet.sensorconfiguration W/OpenGLRenderer: Failed to choose config with EGL_SWAP_BEHAVIOR_PRESERVED, retrying without...
01-10 10:24:49.275 4576-4598/com.developpement.vinet.sensorconfiguration D/EGL_emulation: eglCreateContext: 0x7fe585cb89c0: maj 2 min 0 rcv 2
01-10 10:24:49.280 4576-4598/com.developpement.vinet.sensorconfiguration D/EGL_emulation: eglMakeCurrent: 0x7fe585cb89c0: ver 2 0
01-10 10:24:49.289 4576-4598/com.developpement.vinet.sensorconfiguration D/OpenGLRenderer: Enabling debug mode 0
01-10 10:24:49.309 4576-4598/com.developpement.vinet.sensorconfiguration D/EGL_emulation: eglMakeCurrent: 0x7fe585cb89c0: ver 2 0
01-10 10:24:49.378 4576-4576/com.developpement.vinet.sensorconfiguration W/art: Before Android 4.1, method int android.support.v7.widget.ListViewCompat.lookForSelectablePosit
01-10 10:25:32.484 4576-4576/com.developpement.vinet.sensorconfiguration V/LoginActivity: onClickConnect, value of login : (jeremy), value of password : (admin)
01-10 10:25:32.552 4576-4598/com.developpement.vinet.sensorconfiguration D/EGL_emulation: eglMakeCurrent: 0x7fe585cb89c0: ver 2 0
01-10 10:25:32.555 4576-4598/com.developpement.vinet.sensorconfiguration D/EGL_emulation: eglMakeCurrent: 0x7fe585cb89c0: ver 2 0
01-10 10:25:32.573 4576-4598/com.developpement.vinet.sensorconfiguration D/EGL_emulation: eglMakeCurrent: 0x7fe585cb89c0: ver 2 0
01-10 10:25:32.611 4576-4598/com.developpement.vinet.sensorconfiguration V/RenderScript: 0x7fe585elf000 Launching thread(s), CPUs 4
01-10 10:25:32.624 4576-4598/com.developpement.vinet.sensorconfiguration D/EGL_emulation: eglMakeCurrent: 0x7fe585cb89c0: ver 2 0
01-10 10:25:34.496 4576-4598/com.developpement.vinet.sensorconfiguration D/EGL_emulation: eglMakeCurrent: 0x7fe585cb89c0: ver 2 0
```

Le logeur d'Android Logcat

- Le logcat permet donc d'avoir énormément d'informations sur l'état du périphérique sans même rajouter des informations de log.
- Pour rajouter des informations de log il faut utiliser la classe « Log ».
- Elle n'a pas besoin d'être instanciée, les méthodes permettant d'afficher un message log étant static.

```
Log.i(TAG_DEBUG, msg: "onCreate");
```



Le logeur d'Android Logcat

- Afin de pouvoir classer les messages par ordre de priorité , il existe plusieurs méthodes à la classe « Log » pour afficher des messages.
- Il est possible, dans le Logcat, de filtrer les messages en fonction du niveau de priorité voulu.
- **Liste des commandes les plus courantes :**
 - Log.v -> niveau verbose
 - Log.d -> niveau debug
 - Log.i -> niveau information
 - Log.w -> niveau warning
 - Log.e -> niveau error
 - Log.wtf -> niveau fatal (What a Terrible Failure)



Le logeur d'Android Logcat

- Afin d'avoir un bon niveau de traçabilité, lors d'une utilisation d'une commande de log, il est fortement conseillé d'utiliser un TAG.
- Un TAG peut, par exemple, être le nom de la classe dans laquelle s'exécute l'évènement de log.

```
private final String TAG_DEBUG = "LoginActivity";

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

    Log.i(TAG_DEBUG, msg: "onCreate");
}
```



Le logeur d'Android Logcat

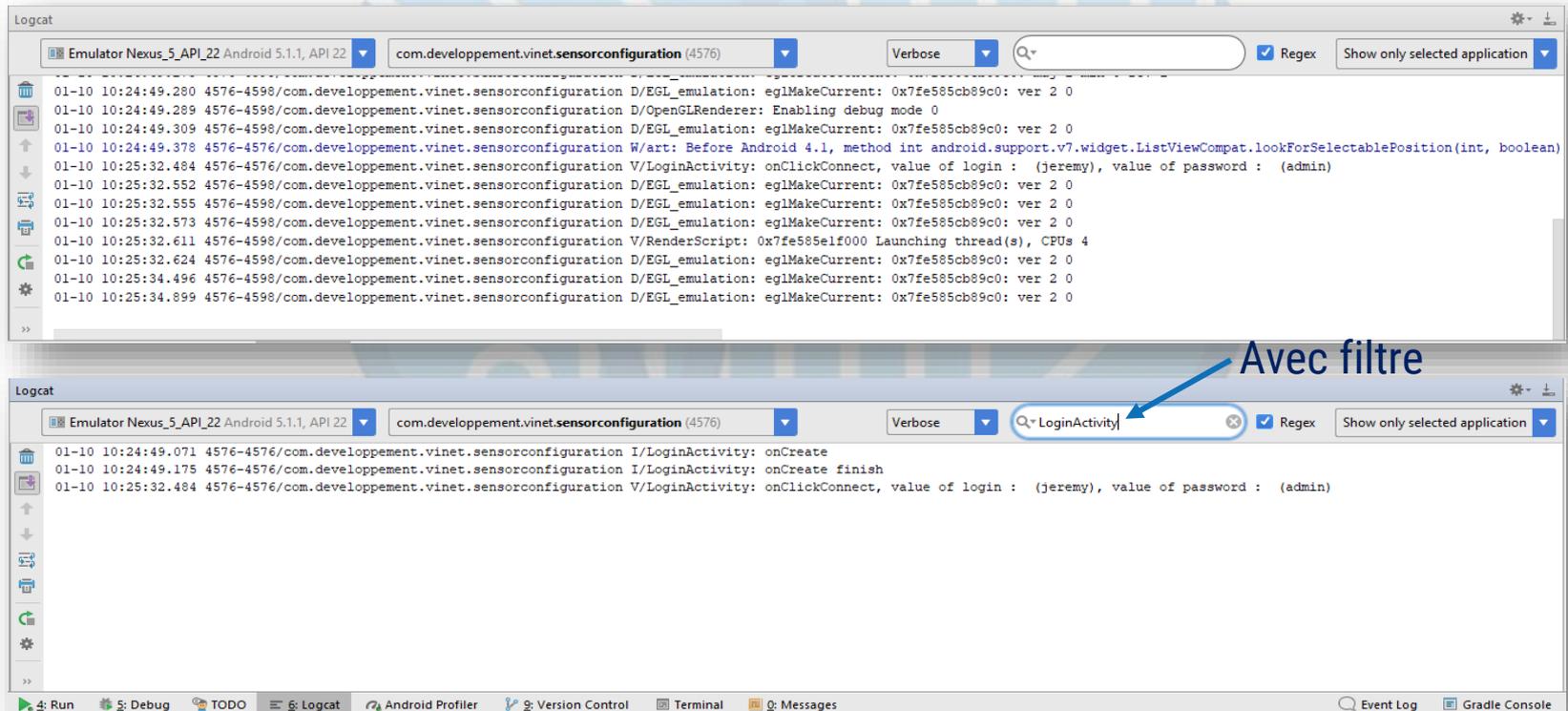
- Le logeur prend en paramètre des string, il est donc tout à fait possible d'afficher le contenu d'une variable à condition de l'avoir converti en string juste avant.

```
//--- Load ressources
m_editTextLogin      = (EditText)    findViewById(R.id.editTextLogin);
m_editTextPassword   = (EditText)    findViewById(R.id.editTextPassword);
m_btnConnect         = (Button)      findViewById(R.id.btnConnect);
//---
m_btnConnect.setOnClickListener((v) -> {
    Log.v(TAG_DEBUG, msg:"onClickConnect, value of login : "
        +m_editTextLogin.getText().toString()+" (+ADMIN_LOGIN+)", value of password : "
        +m_editTextPassword.getText().toString()+" (+ADMIN_PASSWORD+)");
});
```



Le logeur d'Android Logcat

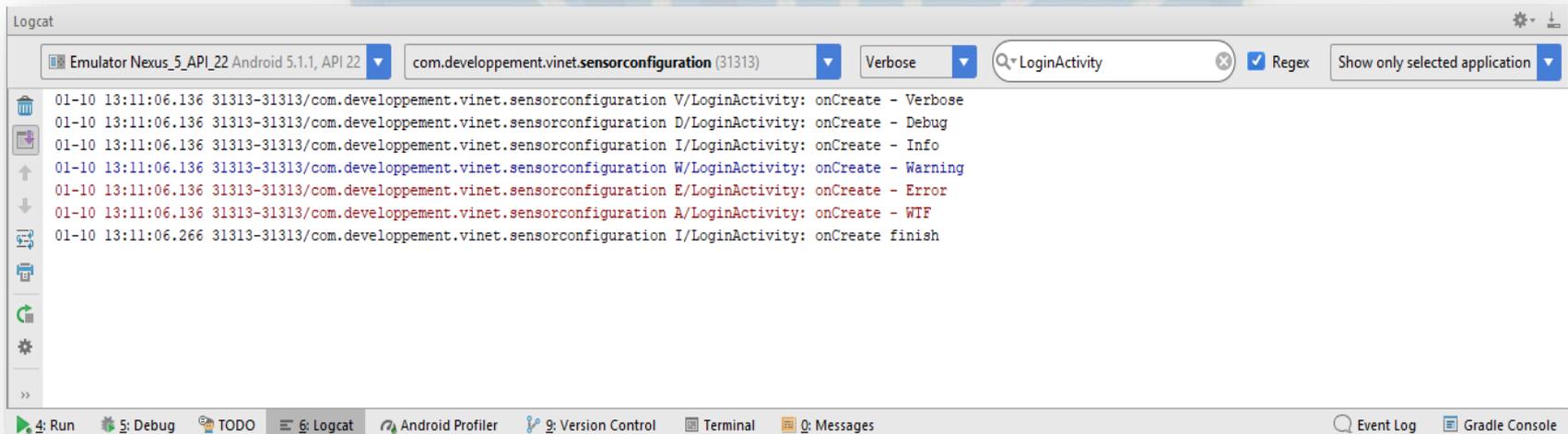
- Une fois dans la console Logcat d'Android studio on peut venir filtrer sur le TAG afin d'avoir seulement les messages comportant ce TAG.



Le logeur d'Android Logcat

- Exemple de rendus en utilisant les 6 niveaux de log possibles

```
Log.v(TAG_DEBUG, msg: "onCreate - Verbose");  
Log.d(TAG_DEBUG, msg: "onCreate - Debug");  
Log.i(TAG_DEBUG, msg: "onCreate - Info");  
Log.w(TAG_DEBUG, msg: "onCreate - Warning");  
Log.e(TAG_DEBUG, msg: "onCreate - Error");  
Log.wtf(TAG_DEBUG, msg: "onCreate - WTF");
```



Utilisation du débogueur (ADB)

- Si le logeur est un outil très utile, il est parfois insuffisant pour déboguer en profondeur son application.
- Android possède parmi tous ces outils un débogueur du nom de ADB
- ADB : Android Debug bridge
- Il permet, en mode debug, de se connecter à une application afin de pouvoir gérer des points d'arrêt et l'analyse de la mémoire du device.



Utilisation du débogueur (ADB)

- L'intégration de ADB dans Android studio est complète, pas de ligne de commande à lancer.
- Pour déboguer une application, il faut la lancer en mode debug.

Lancement de l'application en mode normal

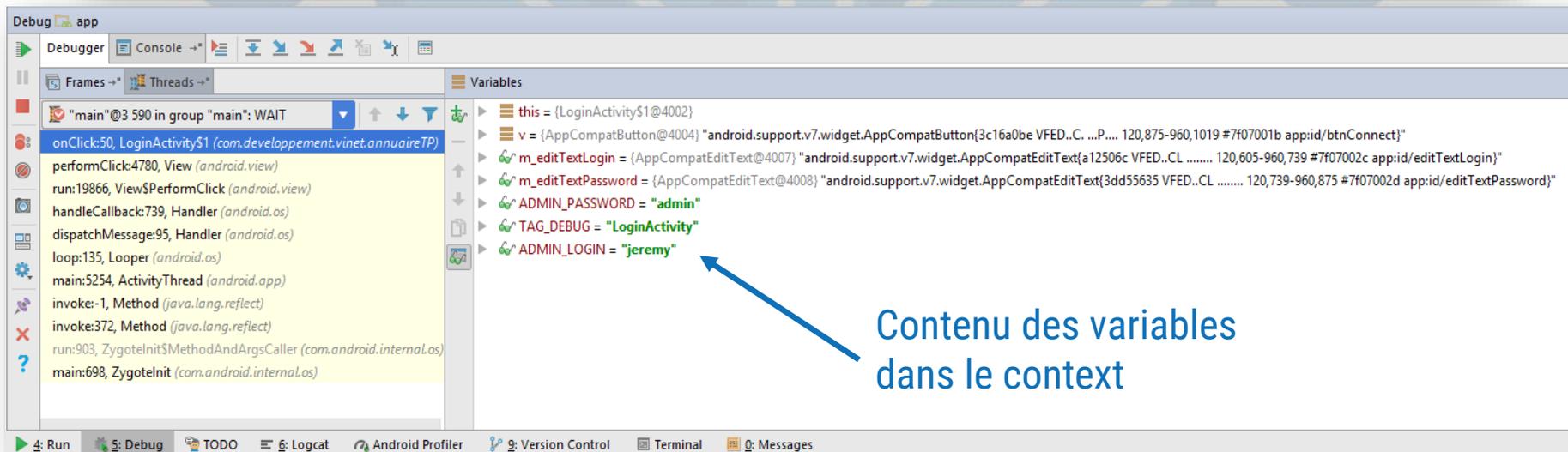
Lancement de l'application en mode debug



Utilisation du débogueur (ADB)

- L'utilisation de point d'arrêt permet de bloquer l'exécution du programme le temps d'analyser le contenu des variables et autres données utiles.

```
49 //--- check login
50 if(m_editTextLogin.getText().toString().equals(ADMIN_LOGIN))
51 {
```



The screenshot shows the Android Studio debugger interface. The Variables panel on the right displays the following variables:

- `this` = {LoginActivity\$1@4002}
- `v` = {AppCompatActivity@4004} "android.support.v7.widget.AppCompatActivity{3c16a0be VFED..C. ...P.... 120,875-960,1019 #7f07001b app:btnConnect}"
- `m_editTextLogin` = {AppCompatActivity@4007} "android.support.v7.widget.AppCompatActivity{a12506c VFED..CL 120,605-960,739 #7f07002c app:editTextLogin}"
- `m_editTextPassword` = {AppCompatActivity@4008} "android.support.v7.widget.AppCompatActivity{3dd55635 VFED..CL 120,739-960,875 #7f07002d app:editTextPassword}"
- `ADMIN_PASSWORD` = "admin"
- `TAG_DEBUG` = "LoginActivity"
- `ADMIN_LOGIN` = "jeremy"

A blue arrow points to the `ADMIN_LOGIN` variable, which is highlighted in green. A text box next to the arrow contains the text: "Contenu des variables dans le contexte".

Utilisation du débogueur (ADB)

- Le débogueur permet aussi d'exécuter du code en mode pas à pas.
- Ce qui permet de vérifier la manière dont le code s'exécute et donc comment votre application réagit.
- ADB permet une grande souplesse pour déboguer une application
- L'utilisation d'un logeur et d'ADB permet de répondre à la grande majorité des cas de débogage d'application.



Ressources

- <https://developer.android.com/studio/debug/index.html>
- <https://developer.android.com/studio/debug/am-logcat.html>
- <https://developer.android.com/>

