|  |
| --- |
|  |
| CrypgetSQL試験手順 |
|  |

|  |
| --- |
| 2016年4月 |

目次

[1. はじめに 3](#_Toc417393588)

[1.1. 試験概要 3](#_Toc417393589)

[1.1.1. 機能試験 3](#_Toc417393590)

[1.1.2. メモリリーク試験 3](#_Toc417393591)

[1.1.3. 性能試験 3](#_Toc417393592)

[1.1.4. マルチスレッド試験 3](#_Toc417393593)

[1.1.5. 結合試験 3](#_Toc417393594)

[1.1.6. 移行試験 4](#_Toc417393595)

[1.2. 試験アプリ概要 4](#_Toc417393596)

[1.2.1. 全体像 4](#_Toc417393597)

[1.2.2. 単体試験アプリ群 4](#_Toc417393598)

[1.2.3. 結合試験アプリ群 4](#_Toc417393599)

[1.3. 本書の構成 5](#_Toc417393600)

[1.4. 用語 5](#_Toc417393601)

[2. 試験環境の準備 6](#_Toc417393602)

[2.1. SDKとADTの準備 6](#_Toc417393603)

[2.2. SDK Platformの準備 8](#_Toc417393604)

[2.3. エミュレータの準備 9](#_Toc417393605)

[2.4. 実機の準備 13](#_Toc417393606)

[2.4.1. USBドライバのインストール 13](#_Toc417393607)

[2.4.2. 実機での設定 13](#_Toc417393608)

[2.4.3. 接続の確認 14](#_Toc417393609)

[2.5. 試験アプリの準備 15](#_Toc417393610)

[2.5.1. 試験アプリのインポート 15](#_Toc417393611)

[2.5.2. 試験アプリの設定 15](#_Toc417393612)

[2.5.3. モジュールの配置 16](#_Toc417393613)

[3. 試験手順 17](#_Toc417393614)

[3.1. 共通の手順 17](#_Toc417393615)

[3.1.1. Android JUnitの実行方法 17](#_Toc417393616)

[3.1.2. Android Applicationの実行方法 18](#_Toc417393617)

[3.2. 機能試験の手順 19](#_Toc417393618)

[3.2.1. SQL試験の手順 19](#_Toc417393619)

[3.2.2. Java API試験の手順 22](#_Toc417393620)

[3.3. メモリリーク試験 22](#_Toc417393621)

[3.3.1. メモリリーク試験 23](#_Toc417393622)

[3.3.2. Corrupt試験 25](#_Toc417393623)

[3.4. 性能試験の手順 25](#_Toc417393624)

[3.5. マルチスレッド試験の手順 25](#_Toc417393625)

[3.6. 結合試験の手順 26](#_Toc417393626)

[3.6.1. 試験番号1.1.X 26](#_Toc417393627)

[3.6.2. 試験番号1.2.X 28](#_Toc417393628)

[3.6.3. 試験番号1.3.X 29](#_Toc417393629)

[3.6.4. 試験番号1.4.X 30](#_Toc417393630)

[3.6.5. 試験番号1.5.X 31](#_Toc417393631)

[3.6.6. 試験番号1.6.X 31](#_Toc417393632)

[3.6.7. 試験番号1.7.X 31](#_Toc417393633)

[3.6.8. 試験番号2.X 33](#_Toc417393634)

[3.6.9. 試験番号3.X 33](#_Toc417393637)

[3.6.10. 試験番号4.X 35](#_Toc417393638)

[3.6.11. 試験番号5.X 36](#_Toc417393639)

[3.6.12. 試験番号6.X 37](#_Toc417393640)

[3.6.13. 試験番号7 .X 39](#_Toc417393641)

[3.6.14. 試験番号8.X 40](#_Toc417393642)

[3.6.15. 試験番号9.X 40](#_Toc417393643)

[3.7. 移行試験の手順 41](#_Toc417393644)

[4. 付録 42](#_Toc417393645)

[4.1. ADBについて 42](#_Toc417393646)

[4.1.1. ADBの環境変数設定 42](#_Toc417393647)

[4.1.2. ADBが認識している端末の確認 42](#_Toc417393648)

[4.1.3. ADBの再起動 42](#_Toc417393649)

[4.1.4. Android端末のログ取得 42](#_Toc417393650)

[4.1.5. APKのインストール、再インストール 43](#_Toc417393651)

[4.1.6. アプリケーションのアンインストール 43](#_Toc417393652)

[4.2. APKの作成 43](#_Toc417393653)

[4.3. 文字コードの設定 45](#_Toc417393654)

[参考文献 46](#_Toc417393655)

# はじめに

本書では、CrypgetSQLの試験手順について記載します。

CrypgetSQLは、Android上で動作するライブラリで、暗号化DB機能を提供します。

CrypgetSQLの特徴として、Androidに標準搭載されているSQLiteを操作するJava APIと概ね同じJava APIで操作可能になっています（一部異なります）。なお、CrypgetSQLの機能の詳細については本書の対象外としています。プログラマーズガイドを参照下さい。

## 試験概要

本項では、試験の全体概要を記載します。試験の具体的な手順は3章で後述します。

試験は、機能試験、メモリリーク試験、性能試験、マルチスレッド試験、結合試験、移行試験から構成されます。

### 機能試験

機能試験は、データベースの機能として問題がないことを確認するSQL試験と、AndroidのSQLiteのAPIとして問題がないことを確認するJava API試験に分かれます。

SQL試験は、SQLiteの試験アプリをAndroid Javaに移植したものになります。

Java API試験は、CTS(Compatibility Test Suite)[[1]](#footnote-1)のうち、android.databaseパッケージを試験するアプリと、android.database.sqliteパッケージを試験するアプリとをCrypgetSQLを試験するように修正したものです。

### メモリリーク試験

CrypgetSQLを長時間利用してもメモリ使用量が増加傾向にないことを確認します。ヒートラン試験を兼ねています。

### 性能試験

　CrypgetSQLの性能を測定します。

### マルチスレッド試験

複数のスレッドからCrypgetSQLを利用できることを確認します。

### 結合試験

実アプリケーションから利用できることを確認する試験や、あるアプリケーションで保存したデータを、別のアプリケーションから読み取れないことを確認する試験、ある端末で保存したデータを別の端末で読み込めないことを確認する試験、などがあります。試験アプリケーションが複数あり、また手順が複雑になっています。

### 移行試験

　CrypgetSQLを旧バージョンから新バージョンに変えても、作成したデータベースファイルを継続利用できることを確認します。

## 試験アプリ概要

本項では、試験アプリをまとめたアーカイブの説明を記載します。

### 全体像

アーカイブの全体像は、表 1‑1の通りです。

表 1‑1 試験アプリの全体像

|  |
| --- |
| CrypgetSQLTest  | CrypgetSQL試験手順.docx：本書  + V1.1  | + app  | | unit\_test ：単体試験アプリ群です。  | | integration\_test ：結合試験アプリ群を格納しています。  | + doc ：試験仕様書を格納しています。  | + modules : 試験対象のモジュールを格納しています。  + V2V3  | + app  | | unit\_test ：単体試験アプリ群です。  | | integration\_test ：結合試験アプリ群を格納しています。  | + doc ：試験仕様書を格納しています。  | + modules : 試験対象のモジュールを格納しています。 |

※ V1.1は同梱していない場合があります。

※ modulesも格納されていない場合があります。その場合、共通基盤から取得して下さい。

### 単体試験アプリ群

いくつかのEclipseプロジェクトで構成されます。これらのプロジェクトで、機能試験、メモリリーク試験、性能試験、マルチスレッド試験を実施します。

### 結合試験アプリ群

Eclipseプロジェクト10個から構成されます。これらプロジェクトで、結合試験と移行試験を実施します。

## 本書の構成

本書の構成は以下の通りです。

* 2章 試験環境の準備

試験環境の準備方法を記載します。

* 3章 試験手順

試験の手順を記載します。

* 4章 付録

試験時に利用すると思われるツール（ADB）や、アプリケーションパッケージ（APK）の作成方法について少し記載します。

## 用語

本書を記載するにあたって、一般用語と区別しにくいなどで、理解しにくいと感じた用語を表 1‑2に記載します。

表 1‑2 用語

|  |  |  |
| --- | --- | --- |
| **No** | **用語** | **意味** |
| 1 | PULL | Android端末に保存されたファイルを、Android端末と接続したPC（Windows等）に移す処理のことです。  ADBでもCUIで実行できますが、本書ではFile ExplorerでGUIとして実行しています。（図 3‑3参考） |
| 2 | PUSH | Android端末と接続したPC（Windows等）から、Android端末にファイルを移す処理のことです。  ADBでもCUIで実行できますが、本書ではFile ExplorerでGUIとして実行しています。（図 3‑3参考） |

# 試験環境の準備

本章では、試験環境の準備について記載します。CrypgetSQLに依存する内容は、2.5節「試験アプリの準備」のみです。Androidの試験環境の知識がある方は、2.5節のみを読むことで問題ありません。

## SDKとADTの準備

本項では、EclipseにSDKとADT（Android Developer Tools）プラグインのインストール方法について記載します。

|  |
| --- |
| 【注意】  本書記載時（2015年4月）現在、IntelliJ IDEA をベースとしたAndroid Studioが公開さいます。今後、このAndroid StudioがGoogle公式のAndroid用IDEになるようです。  しかし、以下の理由から、本書は旧IDEであるEclipseベースのIDEを用いた説明になっております。   * EclipseベースのIDEでCrypgetSQLを試験してきたため * CrypgetSQLの試験アプリがEclipseプロジェクト形式のため * 筆者がAndroid Studio及びIntelliJ IDEAのノウハウを持っていないため   また、Android Studioが公開される以前、GoogleのサイトからSDKやADTを同梱したEclipse（ADTバンドル版Eclipse）が公開されていました。本書の記述はそれを利用したものになっています。しかし、本書記載時、ADTバンドル版Eclipseを公開していたサイトでは、Android Studioに変わってしまったようです。  ADTバンドル版Eclipseと類似の環境を得られると思われるインストール方法について記載しますが、本書を記載するにあたって利用した環境と異なっていることをご了承下さい。 |

1. SDKのインストール

以下のサイトからSDK Tools OnlyのところからSDKをダウンロードし、インストールして下さい。

<https://developer.android.com/sdk/index.html#Other>

1. ADT Pluginのインストール

以下のサイトを参考にADT Pluginのインストール方法を記載します。

<http://developer.android.com/sdk/installing/installing-adt.html>

なお、Eclipseはあらかじめインストールしていることを前提に記載します（Eclipseは3.7.2以上（2015年4月現在）を必要とします）。

* Eclipseのメニューの「Help」→「Install New Software」をクリックします。「Add」ボタンをクリックし、「ADT Plugin」と「https://dl-ssl.google.com/android/eclipse/」を入力します（図 2‑1参照）。

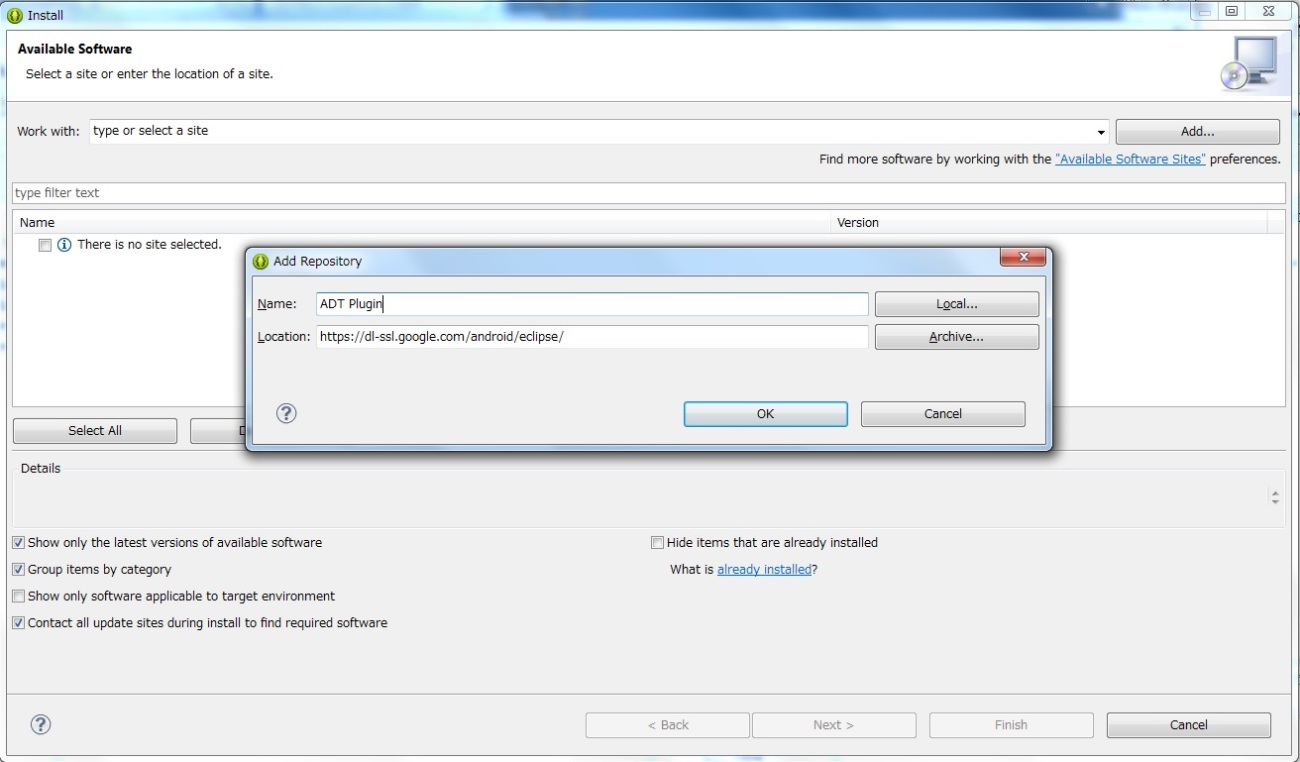


図 2‑1 ADT Pluginのインストール

* 「Developer Tools」にチェックを入れ、Nextボタンを押下します。次の画面でもNextをクリックし、その次で、ライセンスの同意を行い、Finishボタンをクリックします。インストール後はEclipseを再起動して下さい。
* 再起動後、Eclipseのメニューの「Window」→「Preferences」をクリックし、左側で「Android」にフォーカスを当て、右側のSDK Locationで上記①でインストールしたSDKを指定して下さい（図 2‑2参照）。

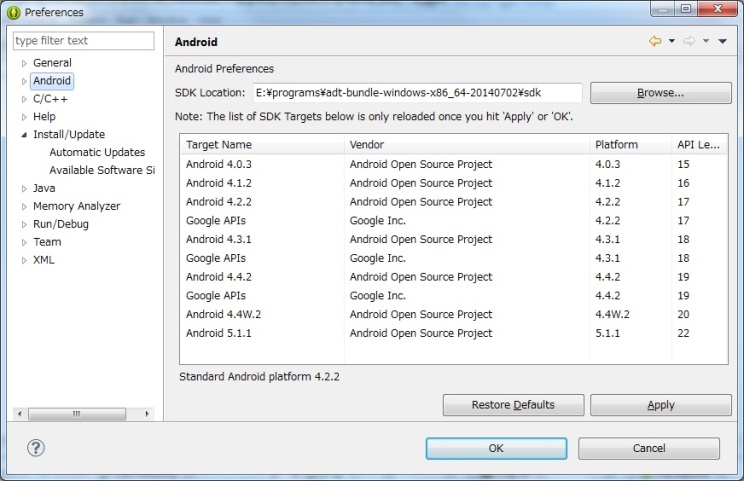


図 2‑2 SDK Locationの指定

## SDK Platformの準備

本項では、試験するAndroidバージョンのSDK Platformのインストールについて記載します。

1. 図 2‑3の赤丸のアイコンをクリックして下さい。

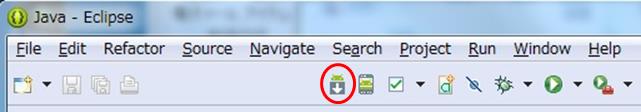


図 2‑3 Eclipseメニュー（SDK Managerのアイコン）

1. 図 2‑4のように試験環境に必要なAndroidのバージョンにおいて、必要なものにチェックし、「Install x Packages…」ボタンをクリックして下さい。

ここでは、Android4.1.2の環境をインストールしようとしています。少なくとも「SDK Platform」とエミュレータイメージの「ARM EABI v7a Image」のインストールは必須です。なお、CrypgetSQLは、2015年4月現在、**ARMにしか対応しておらず、IntelやMIPSは動作保証対象外です。**

【参考情報】

CPUがIntel Atom (x86)の実機の場合、OS組み込みのバイナリトランスレーション機能が有効となっており、自動変換でCrypgetSQLも動作しますが動作保証対象外です。エミュレータではこの機能は無効となっているので動作しません。

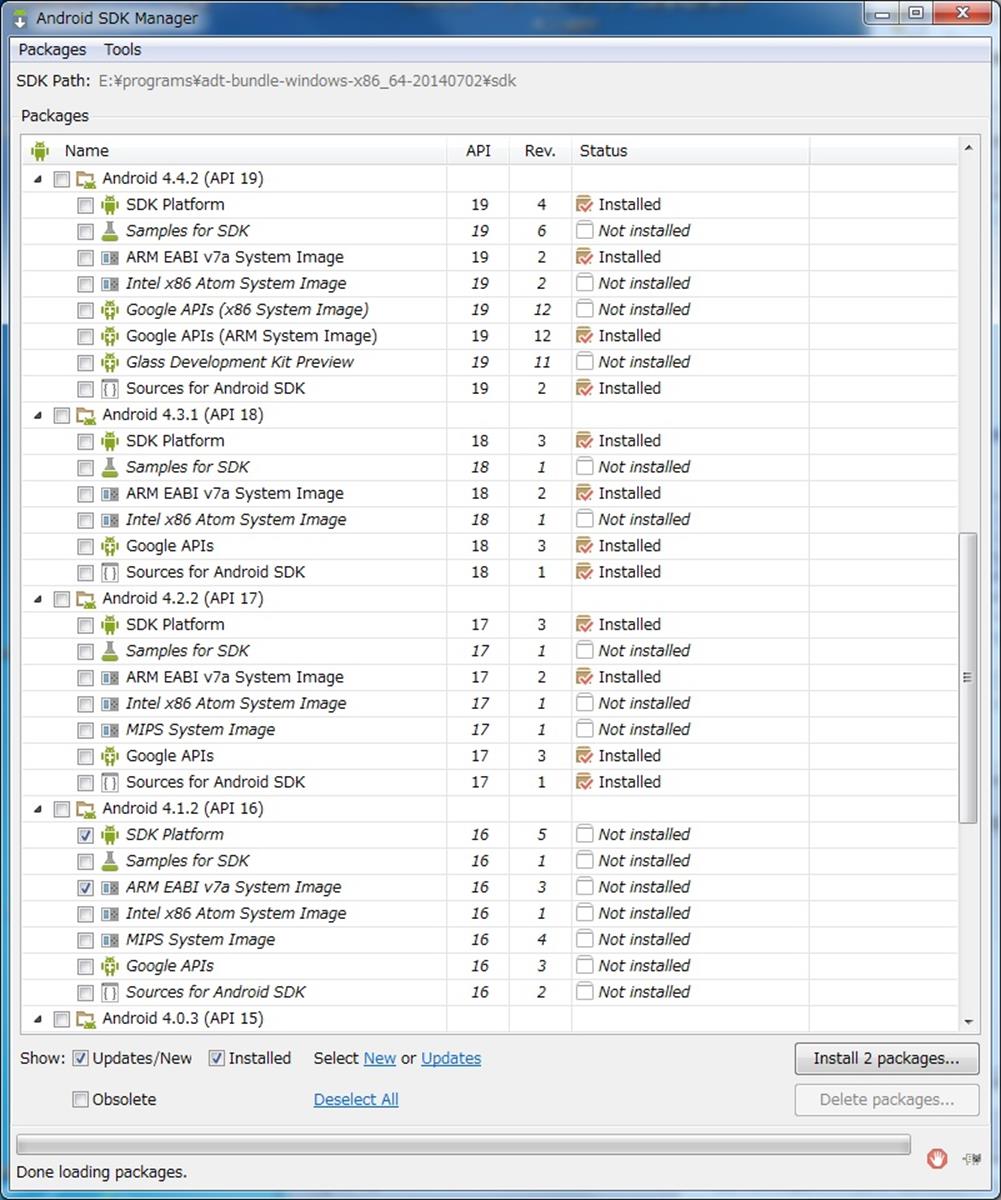


図 2‑4 SDK Manager

ライセンス条項を読み、Acceptのラジオボタンを選択後、インストールボタンを押下するとインストールが開始します。なお、インストールが完了すると、Eclipseの再起動を行う必要がある場合があります。

## エミュレータの準備

本節では、エミュレータの起動までを記載します。エミュレータを利用した試験を実施しない場合は、本節の内容を読み飛ばして下さい。なお、2.2節のSDK Platformのインストールを済ませていることを前提に記載しています。また、エミュレータのその他の設定（日本語化、タイムゾーン、プロキシ）は、参考文献[1]の2.2節を参考にして下さい。

1. Android Virtual Device(AVD) Managerのアイコンをクリックします。

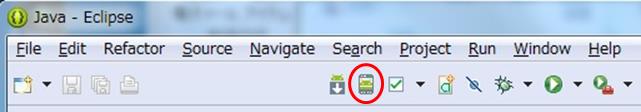


図 2‑5 Eclipseメニュー（AVD Managerのアイコン）

1. 「Create…」ボタンをクリックして下さい。（図 2‑6参照）

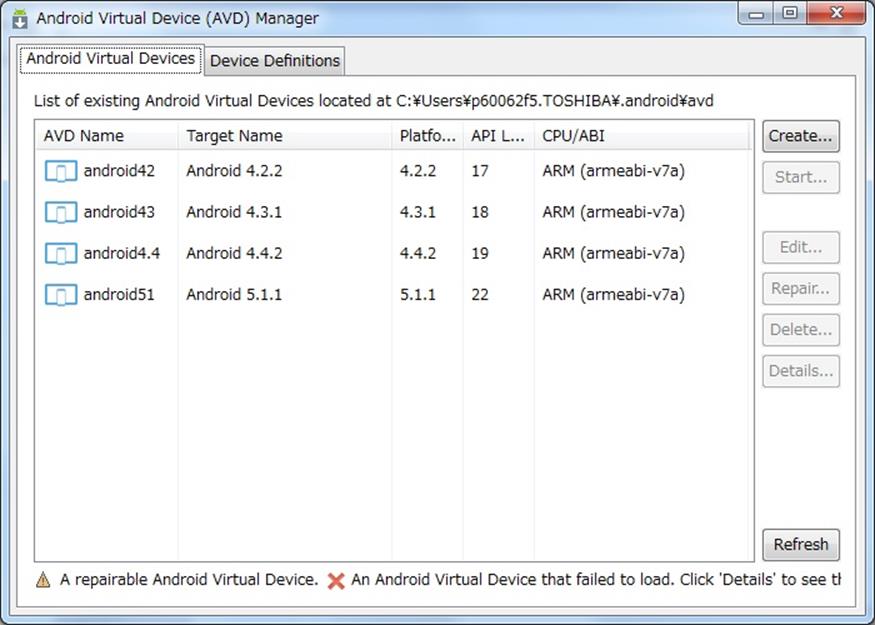


図 2‑6 AVD Manager

1. 試験したい環境に合わせて入力し、OKボタンをクリックします（図 2‑7参照）。なお、AVD Nameは任意です。Targetのプルダウンボックスの選択肢に、試験したいAndroidのバージョンがない場合は、2.2項を参考にSDKのダウンロードを実施して下さい。

（図 2‑7の詳細は、参考文献[1]の2.2.1節「エミュレータの作成」も参考になります。）

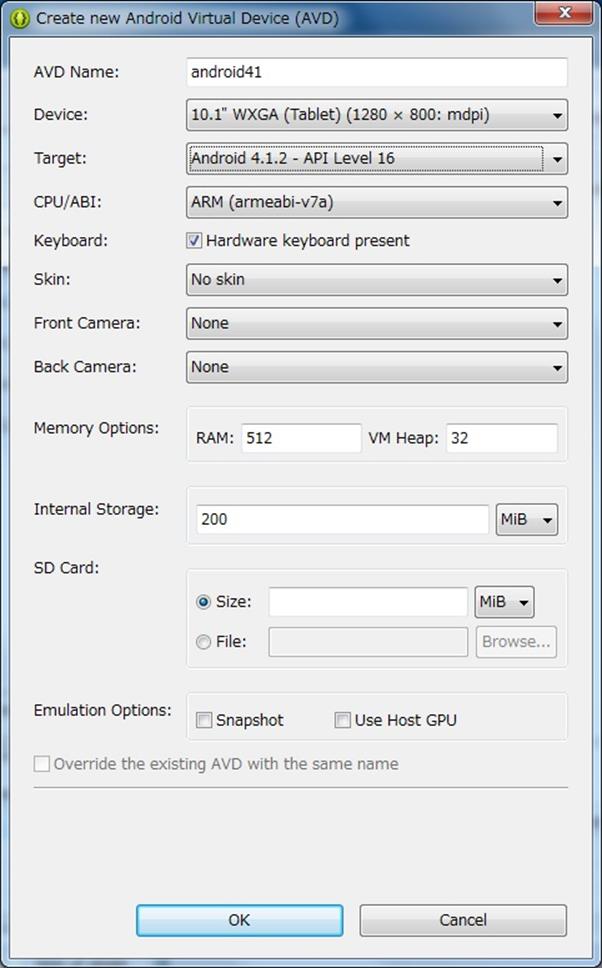


図 2‑7 Create new Android Virtual Device

1. 図 2‑6の左側のテーブルに作成したAVDが表示されますので、それにフォーカスを当て、「Start…」ボタンをクリックすると表示される画面で「Launch…」ボタンをクリックするとエミュレータが起動します。（起動には多少の時間がかかります。）
2. Eclipse上のDevices View（図 2‑8参照）でエミュレータが表示されることを確認して下さい。

（Eclipse上にDevices Viewが表示されない場合は、Eclipseメニューの「Window」→「Show View」→「Other…」をクリックし、「Android」→「Devices」にフォーカスを当て、「OK」ボタンをクリックします。）

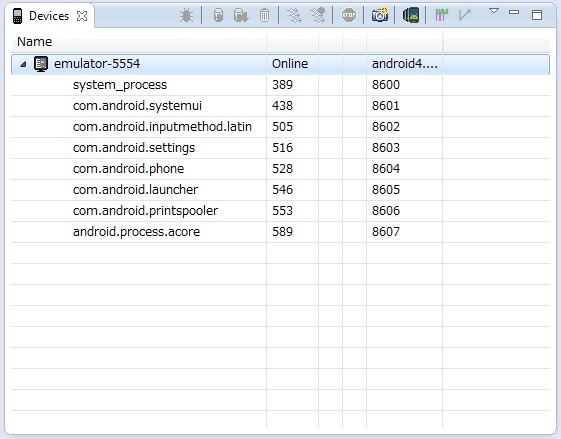


図 2‑8 Devices View

## 実機の準備

Eclipseを起動しているPC（Windowsマシン）とAndroid端末とをUSB接続する方法を記載します。本項での内容は、実機に依存している可能性が高く、記載した通りにできない場合も多々あることご了承下さい。

### USBドライバのインストール

実機ごとにUSBドライバのインストール方法が異なり、統一的なインストール方法は不明です。端末の機器に応じたUSBドライバのインストール方法をインターネットで検索し、それに従いインストールして下さい。

例えば、富士通製の一部の端末は、USBドライバとインストール方法を記載したテキストを、以下のサイトからダウンロードできます。

例）富士通のダウンロードサイト

<http://spf.fmworld.net/fujitsu/c/develop/sp/android/>

### 実機での設定

Android実機側で、USBデバッグを行えるように設定する方法を記載します。

（日本語対応のAndroid端末の場合は「Settings」→「設定」、「Developer Options」→「開発者向けオプション」、「USB Debugging」→「USBデバッグ」、「Build Number」→「ビルド番号」と表示されます。適宜、読み替えて下さい。）

1. Android実機を起動し、アプリケーションの中からSettingsのアイコン（図 2‑9参考）をタップします。

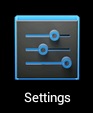


図 2‑9 Settingsのアイコン

1. 「Developer Options」をタップし、「USB Debugging」のチェックボックスをチェックします（図 2‑10参考）。※デフォルトで「Developer Options」が表示されていない実機を時々みかけます。その場合、「About Phone」をタップし、「Build Number」を7回連続でタップすると表示されます。

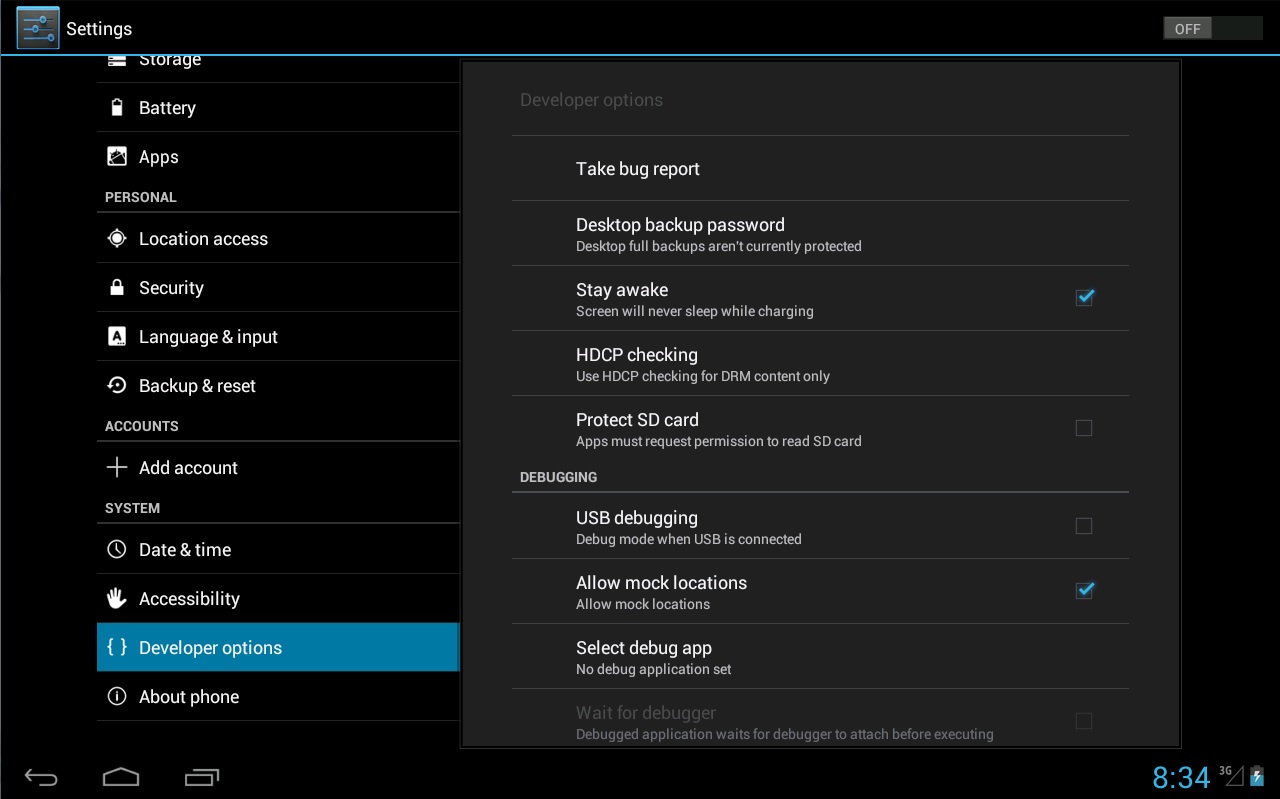


図 2‑10 Developer Options

1. Android実機をUSB接続します。初回接続時にはAndroidの画面に「Allow USB debugging?」の確認ダイアログが出るので「OK」をタップして下さい。

### 接続の確認

Eclipseを起動しているWindows PCが、USB接続したAndroid実機を認識した場合、図 2‑8のエミュレータの場合と同じく、Devices Viewに表示されます。

## 試験アプリの準備

本項では、試験アプリの準備する方法を記載します。

### 試験アプリのインポート

試験アプリをEclipseにインポートします。

1. Eclipseのメニューの「File」→「Import」をクリックします。
2. 表示された画面で、「Android」→「Existing Android Code Into Workspace」にフォーカスを当て、「Next >」ボタンをクリックします。
3. Root Directoryの「Browse．．．」ボタンをクリックし、試験アプリのフォルダを選択します。

### 試験アプリの設定

試験するAndroidのバージョンを設定します。

1. Eclipseの試験アプリのプロジェクトを右クリックし、「Properties」をクリックします（図 2‑11参照）。

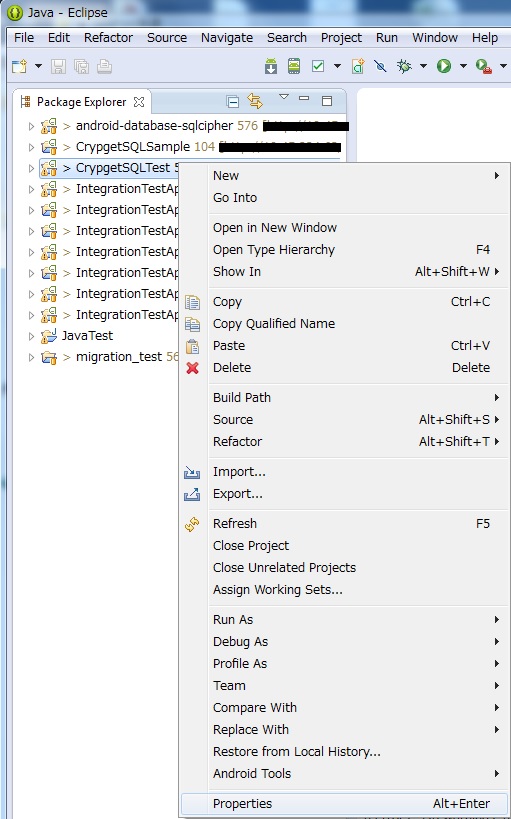


図 2‑11 Eclipseプロジェクトの右クリック

1. 左側の「Android」をクリックし、右側の「Project Build Target」を選択します（図 2‑12参照）。なお、「Project Build Target」に試験したいAndroidのバージョンが表示されない場合は、2.2節を参考に、そのバージョンのSDK Platformをインストールできているか確認下さい。

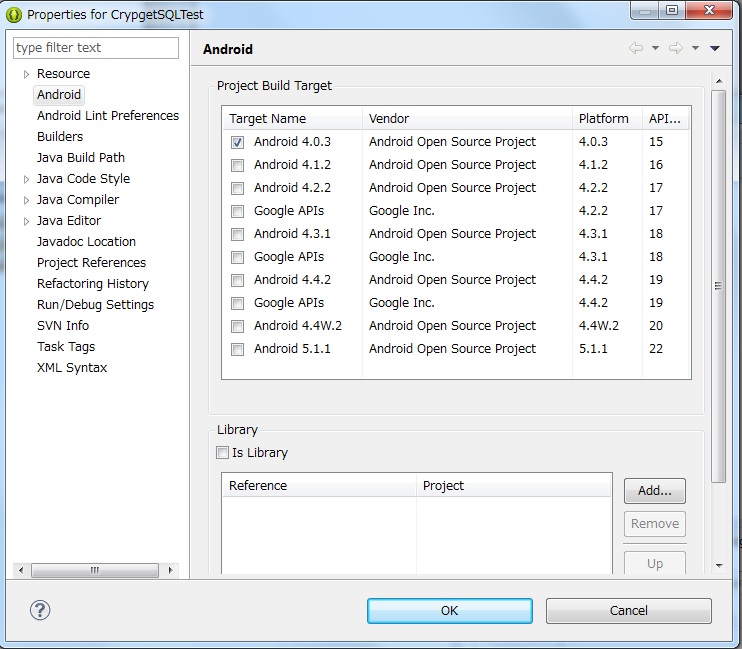


図 2‑12 Eclipse Project Properties

1. コンパイルエラーとなる場合、文字コード設定の不整合が考えられます。4.3文字コードの設定に従って文字コードの変更と再コンパイルを行って下さい。

### モジュールの配置

試験するモジュールを、試験アプリのプロジェクトのルートフォルダ直下のlibsに配置して下さい。

# 試験手順

本章では、1.1節で記載した試験の具体的な手順を記載します。なお、2章に記載した試験環境の準備は終えていることを前提とします。

## 共通の手順

本節では、試験アプリケーションの2種類の実行方法を記載します。一つはAndroid JUnit Testの実行方法で、もう一つはAndroid Applicationの実行方法になります。

単体試験（機能試験、メモリリーク試験、性能試験、マルチスレッド試験）を実施する方法は、一部を除きAndroid Junit Testの実行方法になります。3.1.1項をご参照下さい。

結合試験と移行試験を実施する方法は、Android Applicationの実行方法になります。3.1.2項をご参照下さい。

### Android JUnit Testの実行方法

EclipseのPackage Explorer（またはNavigatorや、Project Explorer）ビューで、Android JUnitクラスやテストスイートのファイルを右クリックします。「Run as」→「Android JUnit Test」をクリックします（図 3‑1参照）。

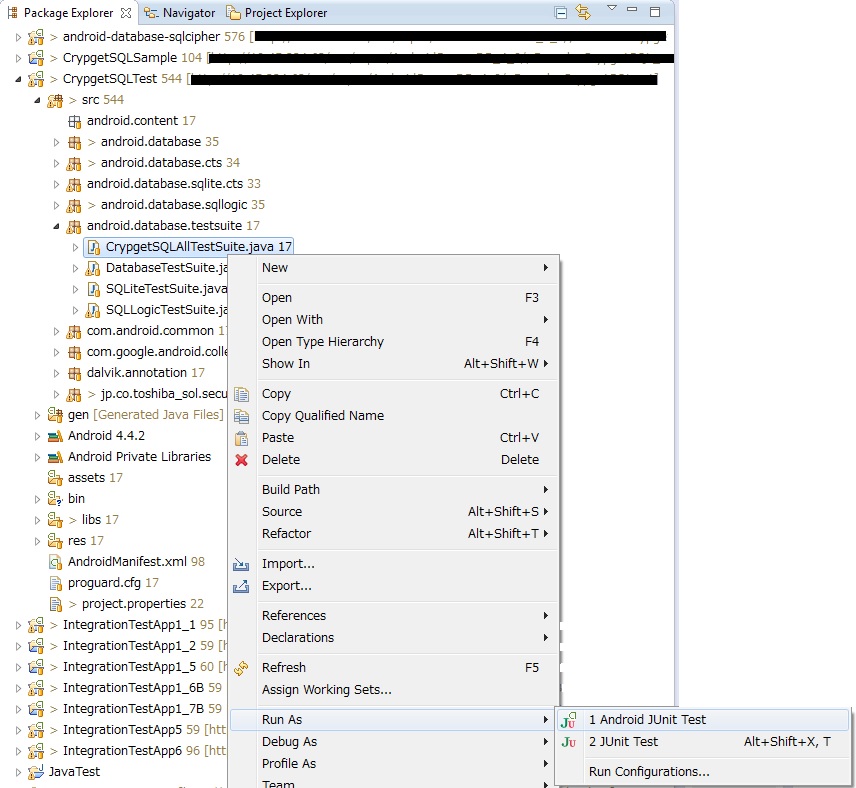


図 3‑1 Android JUnitの実行

### Android Applicationの実行方法

EclipseのPackage Explorer（またはNavigatorや、Project Explorer）ビューで、Eclipseプロジェクトを右クリックします。「Run as」→「Android Application」をクリックします（図 3‑2参照）。

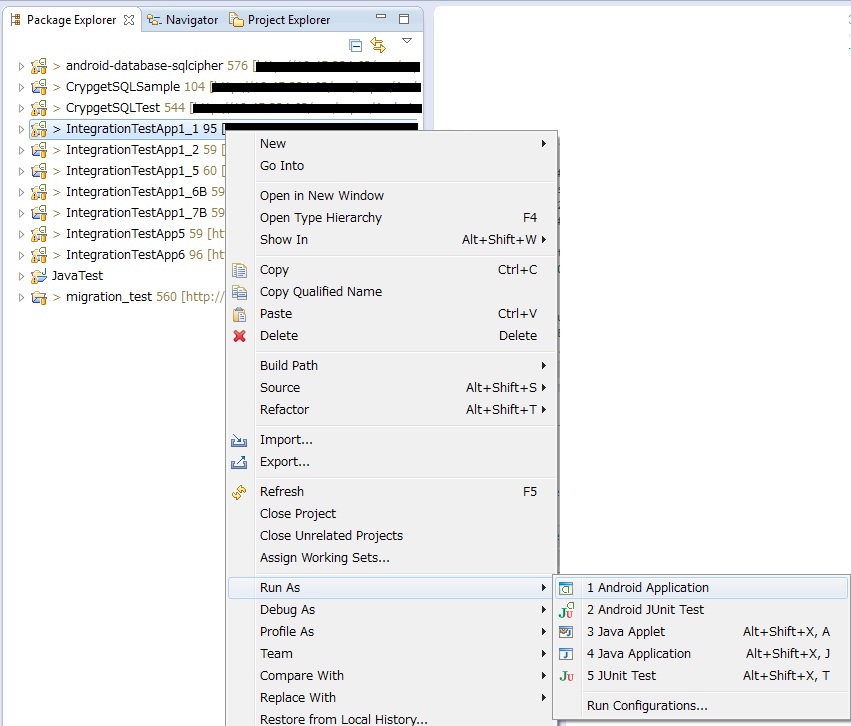


図 3‑2 Android Applicationの実行

## 機能試験の手順

機能試験は、SQL試験とJava API試験に分かれます。

### SQL試験の手順

SQL試験は、後述のクラッシュ試験を除き、自動化しています。

1. 自動化されている試験を実施します。表 3‑1のファイルを実行して下さい。

表 3‑1 SQL試験で実行するファイル

|  |  |
| --- | --- |
| **V1.1** | android.database.testsuite.SQLLogicTestSuite.java |
| **V2.0/V3.0** | test.jp.co.toshiba\_sol.adnroid.sec.database.TestSuiteSQL.java |

1. クラッシュテストを実施します。

クラッシュテストは、トランザクション中にエラーを発生させ、ジャーナルファイルで回復可能かの確認を行う試験です。エミュレータ上で実施します。

表 3‑2のファイルを2回実行します。この試験の手順は、多少複雑になります。

表 3‑2 クラッシュテストで実行するファイル

|  |  |
| --- | --- |
| **V1.1** | android.database.sqllogic.ZZCrashTest.java |
| **V2/V3** | test.jp.co.toshiba\_sol.android.sec.database.sql.ZZCrashTest.java |

* 1回目と2回目、それぞれ実施前にソースコードを修正して下さい。

1回目は、Test Name: crush-2.1からTest Name:crash-2.4までを実施します（表 3‑3参照）。2回目は、Test Name:crash-2.4のみを実施します（表 3‑4参照）。表 3‑3と表 3‑4の赤色太字のコメント部分を、実施する回毎に修正して下さい。

表 3‑3 クラッシュテストのソースコード（1回目）

|  |
| --- |
| **///\***  // Test Name: crush-2.1  （略）  **// \*/**  //################################################  // 2.4　はクラッシュの際に作成されたデータを使用して実行すること  //################################################  // Test Name: crash-2.4  assertTrue(this.verifySignature(SIGNATURE2)); |

表 3‑4 クラッシュテストのソースコード（2回目）

|  |
| --- |
| **/\***  // Test Name: crush-2.1  （略）  **\*/**  //################################################  // 2.4　はクラッシュの際に作成されたデータを使用して実行すること  //################################################  // Test Name: crash-2.4  assertTrue(this.verifySignature(SIGNATURE2)); |

* 1回目実行中に、データベースファイルとジャーナルファイルを取得し、2回目実行に利用して下さい。

1回目実行中にデータベースファイルと、ジャーナルファイルを取得します。Logcatで「######## ここでクラッシュ ##############」と表示されたタイミング（Logcatのフィルタに「#」を入力しておくとわかりやすくなります）で、/data/data/[パッケージ名]/databasesをPULL（Android端末からWindows端末にファイルをコピーする処理）します。

図 3‑3に、EclipseでLogcatとFile Explorerを表示して、PULLする方法を記載します。なお、EclipseにLogcatとFile Explorerが表示されていない場合は、Eclipseメニューの「Window」→「Show View」→「Other…」をクリックし、「Android」→「Logcat」や「Android」→「File Explorer」をフォーカスし、「OK」ボタンをクリックして下さい。

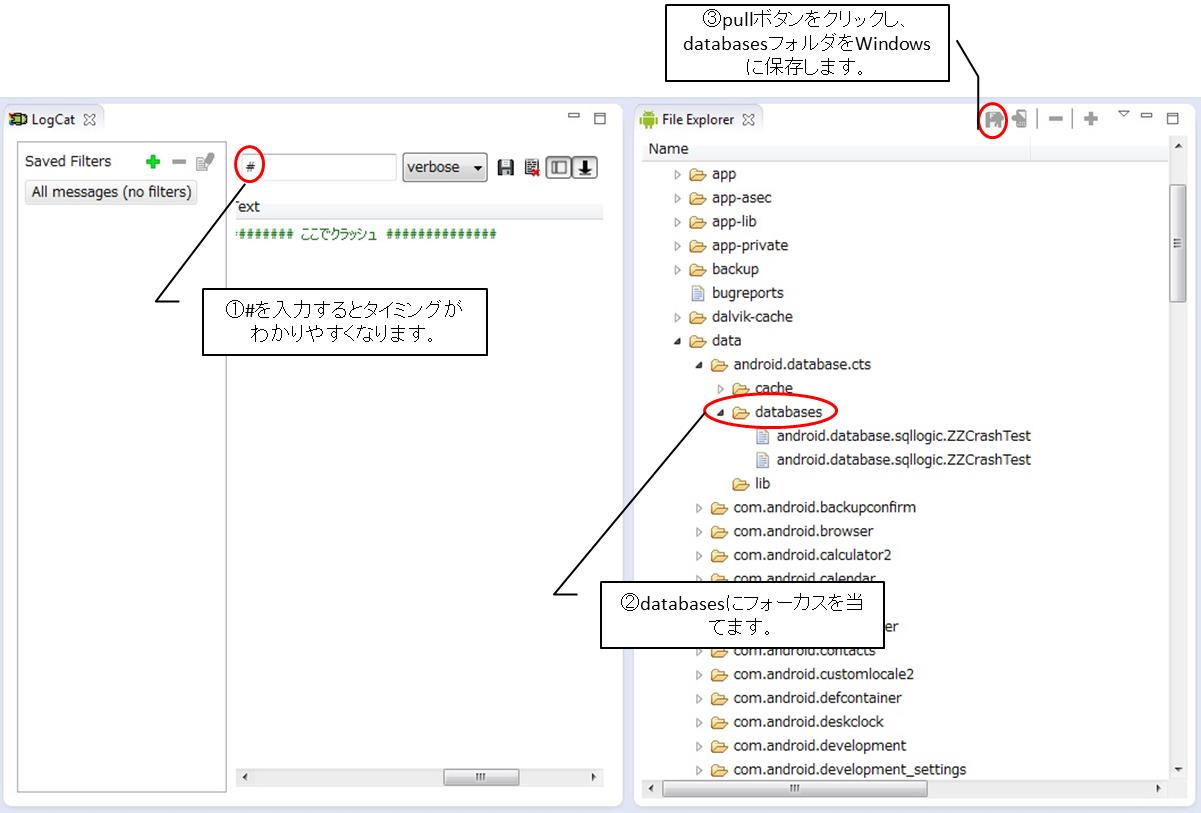


図 3‑3 クラッシュテストでPULLするときの画面

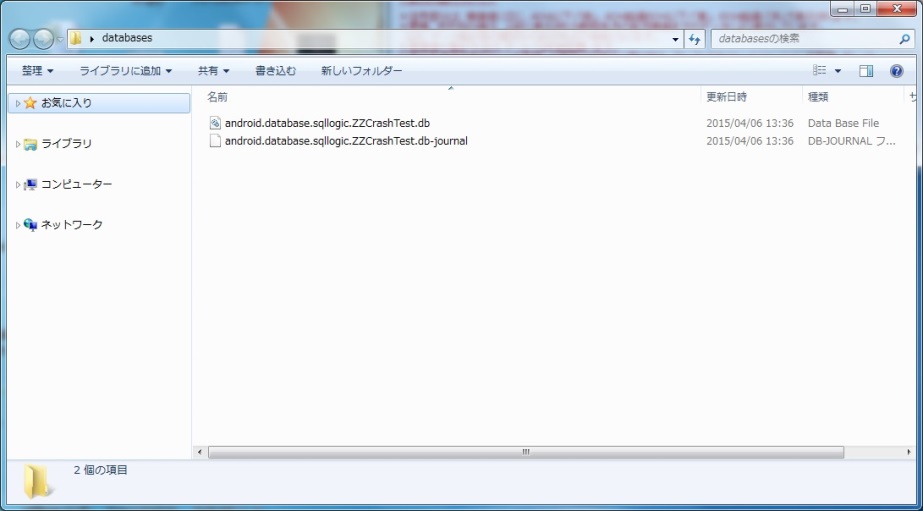


図 3‑4 Windowsに保存されたデータベースファイルとジャーナルファイル

1回目の試験は通りません（JUnitで赤色になります）。2回目は、1回目でPULLしたファイルを同じ場所にPUSH（Windows上のファイルをAndroid端末に保存）します。図 3‑3のpullボタンの右隣が、pushボタンになります。表 3‑4で示した部分を実行すれば試験は通ります（JUnitで緑が表示されます）。緑が表示されなかった場合、1回目でPULLしたタイミングが悪かった可能性があります。再度、1回目から実施し直して下さい。

### Java API試験の手順

Java API試験は、全自動化しています。JUnitを実行し、正常完了することを確認して下さい。

表 3‑5 Java API試験で実行するファイル

|  |  |
| --- | --- |
| **V1.1** | android.database.testsuite.DatabaseTestSuite.java  android.database.testsuite.SQLiteTestSuite.java |
| **V2/V3** | test.jp.co.toshiba\_sol.android.sec.database.TestSuiteCTS.java  test.jp.co.toshiba\_sol.adnroid.sec.database.TestSuiteCTS2.java |

## メモリリーク試験

メモリリーク試験は、その目的（CrypgetSQLを長時間利用してもメモリ使用量が増加傾向に無いことを確認する）通りの試験であるメモリリーク試験（3.3.1項）と、データベースファイルのページ数が変化することに耐えうるか確認するためのCorrupt試験（3.3.2項）に分かれます。

### メモリリーク試験

メモリリーク試験では、一般的なSQL（CREATE TABLE文、SELECT文、INSERT文、UPDATE文、DELETE文）を繰り返し実行しています（耐久試験を兼ねていることから、リソースを解放してしまう可能性があるDROP TABLEは対象外としています）。

試験実施前に、ソースコード中のループ回数を必要に応じて調整して下さい（本試験は、試験に利用する実機の性能により、試験が過不足します）。testMemoryleakメソッドのOUT\_LOOP\_NUM変数の値を変更することで調整できます。ループ処理の前後に、明示的GC処理と、30秒間のSleep処理がありますが、試験者の都合（ループ処理前後のメモリ使用量の確認がしやすい）のための処理で、試験の本質とは関係ありません。コメントアウトして頂いても構いません。

メモリ使用量は、参考文献[1]に記載されているバッチファイル(リスト5-2)を用いて取得します。Androidのdumpsysコマンドのmeminfoオプションで取得していますが、**出力フォーマットはOSバージョンあるいは端末機種によって異なる場合があり、修正しないと利用できないことがあります。**修正方法は機種依存のため、出力フォーマットから推測して作業者が修正して下さい。systemプロセスと、メモリリーク試験アプリのプロセスそれぞれのDalvikヒープ使用量と、Nativeヒープ使用量の増減が時系列でわかれば問題ありません。フィルタリングせずに時系列で記録して、後から情報を取り出してもかまいません。

メモリ使用量に増加傾向が見当たる場合、MAT(Memory Analyzer)を用いて解析しますが、MATについても参考文献[1]をご参照下さい。

表 3‑6 メモリリーク試験で実行するファイル

|  |  |
| --- | --- |
| **V1.1** | jp.co.toshiba\_sol.securedb.test.SQLMemoryLeakTest.java |
| **V2/V3** | CrypgetSQL V2/V3ではメモリリーク試験アプリは、Android JUnitではなくAndroid Applicationになります。MemoryleakTestを利用します。 |

CrypgetSQL V2/V3のメモリリーク試験は、Android ApplicationであるMemoryleakTestを利用します。アプリケーションの画面上にボタンは「start」の一つしかありません。ボタンをタップして、画面上に「finish.」と表示されたら完了です。完了後、外部ストレージから「CrypgetSQL\_Memoryleak\_[日付]\_[時刻].txt」ファイルを取得して下さい（図 3‑5参照）。File Explorerを利用していますが、この利用方法は図 3‑3周辺を参考下さい。なお、このテキストファイルの出力箇所は端末依存になります。

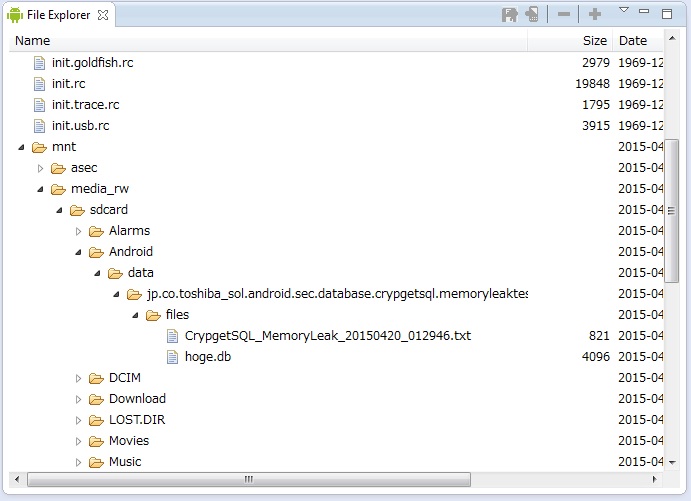


図 3‑5 Memoryleak結果テキスト

このファイルの内容は、表 3‑7のようなCSVになります。一行目に内容が記載されています。Excel等で、グラフ化するなどして、メモリの増減を可視化し、メモリリークの有無を確認して下さい。

表 3‑7 Memoryleak結果テキストの詳細

|  |
| --- |
| 2015-04-20-01:29:46.125,app\_dalvik\_total,app\_dalvik\_alloc,app\_native\_total,app\_native\_alloc  2015-04-20 01:29:46.154,3153920,3057008,1970176,1963424,beforeOpen  2015-04-20 01:29:50.281,3383296,3171088,2162688,1991744,end of 0  2015-04-20 01:29:53.120,3592192,3047776,2162688,1993792,end of 1  2015-04-20 01:29:55.532,3592192,3447656,2166784,1996392,end of 2  2015-04-20 01:29:57.742,3592192,3331240,2166784,1993864,end of 3  2015-04-20 01:29:58.979,3600384,3201920,2166784,1993888,end of 4  2015-04-20 01:30:00.528,3600384,3078440,2166784,1993888,end of 5  2015-04-20 01:30:02.746,3600384,3478312,2166784,1996488,end of 6  2015-04-20 01:30:04.962,3600384,3361368,2166784,1993888,end of 7  2015-04-20 01:30:07.361,3600384,3233816,2166784,1993888,end of 8  2015-04-20 01:30:09.316,3600384,3110008,2166784,1993888,end of 9 |

### Corrupt試験

データベースファイルのヘッダにページサイズがあり、暗号化後のページサイズが0x0200(512)、0x0800(2048)などを示している場合に、CrypgetSQLは、データベースファイルが暗号化されていないとみなして（本当は暗号化されている）、正確にファイル操作できなくなるバグが以前ありました。そのバグは修正済みですが、本試験では、正しく修正され、同バグが再現しないことを確認します。JUnitを実行し、正常完了することを確認して下さい。

表 3‑8 Corrupt試験で実行するファイル

|  |  |
| --- | --- |
| **V1.1** | jp.co.toshiba\_sol.securedb.test.CorruptTest.java |
| **V2/V3** | test.jp.co.toshiba\_sol.android.sec.database.performance.CorruptTest.java |

## 性能試験の手順

性能試験は、SQL文16種の性能を計る試験と、ライブラリロード等の性能を計る試験と、データサイズ別の性能を計る試験があります。

表 3‑9のSQLCrypgetPerformanceTest.javaは、SQL文16種の性能を計る試験です。SQLCrypgetPerformanceTest2.javaは、ライブラリロード等の性能を計る試験です。CrypgetSQLPerformanceTrendTest.javaは、データサイズ別の性能を計る試験です。

表 3‑9 性能試験で実行するファイル

|  |  |
| --- | --- |
| **V1.1** | jp.co.toshiba\_sol.securedb.test.SQLCrypgetPerformanceTest.java  jp.co.toshiba\_sol.securedb.test.SQLCrypgetPerformanceTest2.java  jp.co.toshiba\_sol.securedb.test.CrypgetSQLPerformanceTrendTest.java |
| **V2/V3** | test.jp.co.toshiba\_sol.android.sec.database.performance.SQLitePerformanceTest.java  test.jp.co.toshiba\_sol.android.sec.database.performance.SQLitePerformanceTest2.java  test.jp.co.toshiba\_sol.android.sec.database.performance.SQLitePerformanceTrendTest.java |

## マルチスレッド試験の手順

マルチスレッドでCrypgetSQLを利用できるかの試験は、表 3‑10のファイルを実行するのみです。特に説明を記載しません。

表 3‑10 マルチスレッド試験で実行するファイル

|  |  |
| --- | --- |
| **V1.1** | jp.co.toshiba\_sol.securedb.test.CrypgetSQLMultiThreadTest.java |
| **V2/V3** | test.jp.co.toshiba\_sol.android.sec.database.performance.CrypgetSQLMultiThreadTest.java |

## 結合試験の手順

結合試験は、自動化できていません。試験仕様書の順番で、試験項目ごとに試験の手順や試験の意図を記載します。なお、結合試験アプリケーションは、Android Applicationになっております。実行方法が不明な場合は、3.1.2項を参照下さい。

### 試験番号1.1.X

* 試験の内容

画面遷移を行わないAndroidアプリケーションからCrypgetSQLを利用できることを確認します。

* 試験アプリ

IntegrationTestApp1\_1（図 3‑6参照）を利用します。ボタンの説明は表 3‑11に記載します。

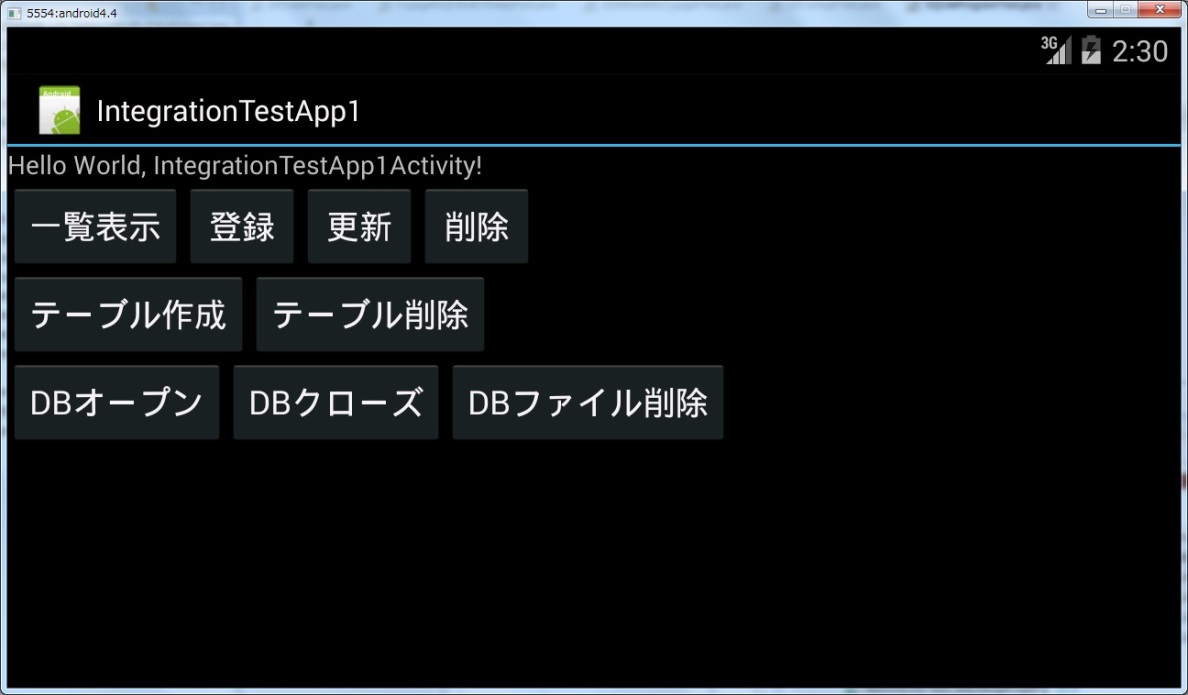


図 3‑6 IntegrationTestApp1\_1の画面

表 3‑11 IntegrationTestApp1\_1のボタンの説明

|  |  |  |
| --- | --- | --- |
| **No** | **ボタン** | **処理の内容** |
| 1 | 一覧表示 | 登録されているデータを検索し、画面上に表示します。データは「DBオープン」ボタンの下に表示されます。データが登録されていない場合は、表示されません。 |
| 2 | 登録 | 4件のデータを登録します。「東芝太郎、45」「東芝花子、40」「浜松太郎、35」「浜松花子、30」が登録されます。複数回タップすれば、タップした分だけ4件ずつ追加登録されます。 |
| 3 | 更新 | 「東芝太郎、45」を「東芝次郎、20」に更新します。 |
| 4 | 削除 | 登録されているデータを一括削除します。 |
| 5 | テーブル作成 | テーブルを作成します。 |
| 6 | テーブル削除 | テーブルを削除します。 |
| 7 | DBオープン | DBをオープンします。 |
| 8 | DBクローズ | DBをクローズします。 |
| 9 | DBファイル削除 | データベースファイルを削除します。 |
| 10 | 画面遷移 | 別の画面を遷移します。  IntegrationTestApp1\_1には当ボタンはありません。後述のIntegrationTestApp1\_2、IntegrationTestApp1\_5にあります。 |
| 11 | 前画面に戻る | 前の画面に遷移します。  IntegrationTestApp1\_1には当ボタンはありません。後述のIntegrationTestApp1\_2、IntegrationTestApp1\_5にあります。 |

以下、IntegrationTestApp1\_1の正常な動作例を記載します（図 3‑7参照）。

図 3‑7の左から1番目の図は、アプリケーションを起動し「DBオープン」→「テーブル作成」→「一覧表示」をタップしたものです。さらに「登録」→「一覧表示」をタップしたものが、左から2番目の図です。さらに「更新」→「一覧表示」をタップしたものが左から3番目の図です。さらに「削除」→「一覧表示」をタップしたものが左から4番目の図です。



図 3‑7 アプリケーションの利用例（その１）

別の正常な動作例を記載します。データが登録されていない状態から「登録」→「更新」→「登録」→「一覧表示」の順にボタンをタップした場合、図 3‑8のようになります。



図 3‑8 アプリケーションの利用例（その２）

なお、テーブルを作成していない状態や、DBオープンしていない状態で、「一覧表示」や「登録」「更新」「削除」を行うと、エラーが発生し、アプリケーションが停止します。

* 試験の手順

試験を実施する前に、IntegrationTestApp1\_1が端末にインストールされていないことを確認して下さい。

IntegrationTestApp1\_1を新たに端末にインストールして起動し、試験仕様書の各試験項目の操作を順番に実施して下さい。難しくないため説明を省きます。

### 試験番号1.2.X

* 試験の内容

画面遷移があるAndroidアプリケーションからCrypgetSQLを利用できるかを確認します。

* 試験アプリ

IntegrationTestApp1\_2を利用します。IntegrationTestApp1\_1（図 3‑6、表 3‑11）と概ね同じです。

* 試験の手順

試験を実施する前に、IntegrationTestApp1\_2が端末にインストールされていないことを確認して下さい。

IntegrationTestApp1\_2を新たに端末にインストールして起動し、試験仕様書の各試験項目の操作を順番に実施して下さい。難しくないため説明を省きます。

### 試験番号1.3.X

* 試験の内容

画面遷移を行わないAndroidアプリケーションからCrypgetSQLを利用できるかを確認します。

* 試験アプリ

IntegrationTestApp1\_1を利用します。

* 試験の手順

試験仕様書の通りに、ボタンを押下します。難しくないため説明を省きます。

* 備考

試験番号1.1.Xとの違いは、すでにIntegrationTestApp1\_1がAndroid端末にインストールされているところから、アプリケーションを起動します。

試験番号1.1.Xの試験時にインストールされているので、Android端末の画面を操作し、いったん強制終了し再度起動すれば問題ありません。

強制終了は、Settings（図 2‑9参照）をタップし、左側から「Apps」をタップします（図 2‑10のDeveloper Optionsの8つ上にあります）。アプリケーションが一覧されますので、そこからアプリケーション名をタップすると表示されるアプリ情報画面（図 3‑10参照、ただし、IntegrationTestApp1\_5の画面です）で「強制停止」ボタンをタップします。

起動には、図 3‑9のようなボタンをタップすると、インストール済みのアプリケーションのアイコンが一覧されるので、IntegrationTestApp1\_1のアイコンをタップして起動して下さい。



図 3‑9 アプリケーションのアイコンを一覧するボタン



図 3‑10 アプリ情報画面

### 試験番号1.4.X

* 試験の内容

画面遷移があるAndroidアプリケーションからCrypgetSQLを利用できるかを確認します。

* 試験アプリ

IntegrationTestApp1\_2を利用します。

* 試験の手順

試験仕様書の通りに、ボタンを押下します。

* 備考

試験番号1.2.Xとの違いは、すでにIntegrationTestApp1\_2がAndroid端末にインストールされているところから、アプリケーションを起動します。

試験番号1.2.Xの試験時にインストールされているので、Android端末の画面を操作し、いったん強制終了し、再度起動すれば問題ありません。アプリケーションの強制起動方法および再起動方法については、前項（3.6.3項）の備考を参考下さい。

### 試験番号1.5.X

* 試験の内容

画面遷移があるAndroidアプリケーションからCrypgetSQLを利用できるかを確認します。

* 試験アプリ

IntegrationTestApp1\_5を利用します。

* 試験の手順

試験仕様書の手順通り操作して下さい。確認内容欄に「（A画面B画面両方で確認する）」と記載しているところは、「画面遷移」や「前画面に戻る」ボタンをタップし、2つの画面から確認して下さい。後述の備考欄にあるよう、この試験アプリでは、2つの画面から1つのデータベースを操作しています。A画面で操作し、B画面に遷移後、「一覧表示」ボタンをタップし、A画面の操作がB画面でも反映できていることを確認します。

* 備考

試験番号1.2.Xと類似していますが、試験番号1.5.Xでは複数画面から同じデータベースを操作し、試験番号1.2.Xでは複数画面それぞれ別のデータベースを操作している点が異なります。

### 試験番号1.6.X

* 試験の内容

あるアプリケーションから別のアプリケーションのデータベースを操作することができる機能（ContentProvider）が利用できることを確認します。

* 試験アプリ

IntegrationTestApp1\_1とIntegrationTestApp1\_6Bを利用します。

* 試験手順

IntegrationTestApp1\_1がバックグラウンドで動作している端末でIntegrationTestApp1\_6Bを操作します。IntegrationTestApp1\_6Bの操作方法については、難しくないため説明を省きます。IntegrationTestApp1\_1をバックグラウンドで動作させるには、IntegrationTestApp1\_1を通常通りの起動（3.1.2項参考）を行い、その後、IntegrationTestApp1\_6Bを起動すれば、IntegrationTestApp1\_1はバックグラウンドになります。

### 試験番号1.7.X

* 試験の内容

データベースファイルを作成したアプリケーションと同じUIDを持つアプリケーションは、そのデータベースファイルが操作できることを確認します。

* 背景の知識

UIDとは、Linuxの用語でユーザIDです。Androidでは、アプリケーションごとにUIDが割り当てられます。本来、CrypgetSQLでは、あるアプリケーションで作成したデータベースファイルは、別のアプリケーションでは解読（復号）できず操作できません。ただし、同じUIDの場合、あるアプリケーションで作成したデータベースファイルを別のアプリケーションから操作することが可能です（UIDだけではなく同じ端末、など条件がそろう必要があります）。

なお、UIDを同じにするには、アプリケーションのAndroidマニフェストファイル（Eclipseプロジェクト直下にあるAndroidManifest.xml）でsharedUserIdの値を同じにします。sharedUserIdには、一つ以上のピリオドを含む必要があります。

なお、アプリケーションのUIDについては、ADB（4.1節参考）で確認することができます（表 3‑12の赤太字がUID）。通常、suコマンドはエミュレータでは使用できますが、実機では使用できません。この場合は確認できません。

表 3‑12 adbでのアプリケーションのUID確認方法

|  |
| --- |
| > adb shell  > ps | grep [アプリケーションのパッケージ名]  u0\_aXX …  > su u0\_aXX  > id  uid=**xxxxx**(u0\_aXX) gid=xxxxx(u0\_aXX) |

* 試験アプリ

IntegrationTestApp1\_1とIntegration.app1\_7Bを利用して試験します。これらは、sharedUserIdの値を”jp.co.toshiba\_sol.android.test”と同じにしています。

* 試験手順

1. IntegrationTestApp1\_1でいくつかデータを登録します。
2. IntegrationTestApp1\_1からsqlcrypget.dbをPULLします。PULLは、File Explorerでできます。図 3‑3をご参考下さい。
3. IntegrationTestApp1\_7Bに上記②で取得したsqlcrypget.dbをPUSHします。PUSHは、File Explorerでできます。図 3‑3をご参考下さい。
4. IntegrationTestApp1\_7Bで「DBオープン」ボタンを押下後、試験仕様書に記載された操作を行います。
5. IntegrationTestApp1\_7Bからsqlcrypget.dbをPULLして、それをIntegrationTestApp1\_1にPUSHします。
6. IntegrationTestApp1\_1で「DBオープン」ボタンを押下後、「一覧表示」ボタンで上記④での操作が反映されているか確認します。

### 試験番号2.X

CrypgetSQLがデータベースを暗号化できているかを確認します。データベースファイルをバイナリエディタ等で開き、可読性がないことを確認します。データベースファイルは、クラッシュテスト時にも利用したFile Explorerで取得できます（図 3‑3参照）。PC側にPULLしたデータベースファイルの内容をバイナリエディタで確認します。バイナリエディタについては特に指定ありませんが、ここではBZを用います。図 3‑11の右側で可読性がないことを確認します。SeedとIVについては次項の試験番号3.Xで記載します。

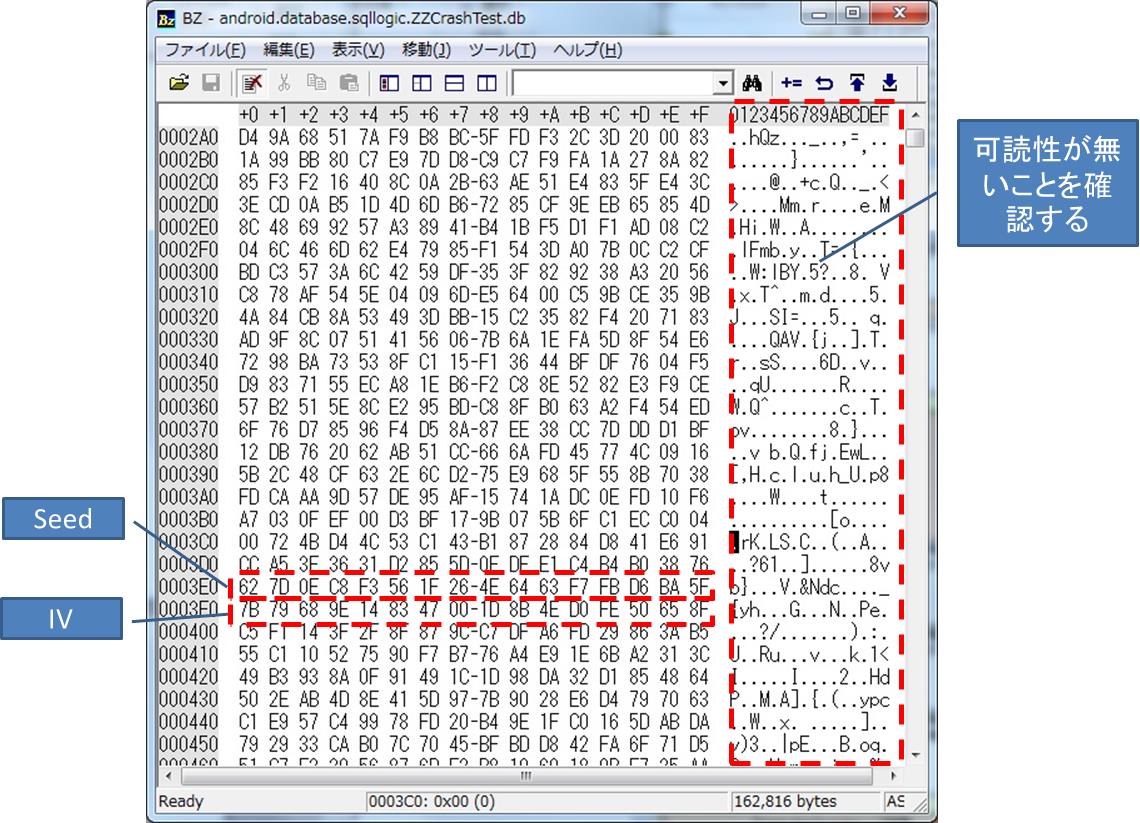


図 3‑11 バイナリエディタで開いたデータベースファイル

### 試験番号3.X

* 試験の内容

CrypgetSQLの鍵管理が、意図したとおりか確認します。試験番号2.Xと同じく、データベースファイルをバイナリエディタで読み込みます。

* 背景の知識

CrypgetSQLは、データベースファイルのページにSeed（鍵シード）やIV（イニシャルベクタ）を埋め込んでいます（図 3‑12参照）。CrypgetSQLのページサイズは1024byte固定です。ページの個数（≒データベースファイルのサイズ）は、保存しているデータが多ければ多くなります。各ページの最後の16byteがIV、その前の16byteがSeedになります。なお、バイナリエディタBZを利用した場合、1024(10進数)→400(16進数)と変換することにご注意下さい（図 3‑11参照）。

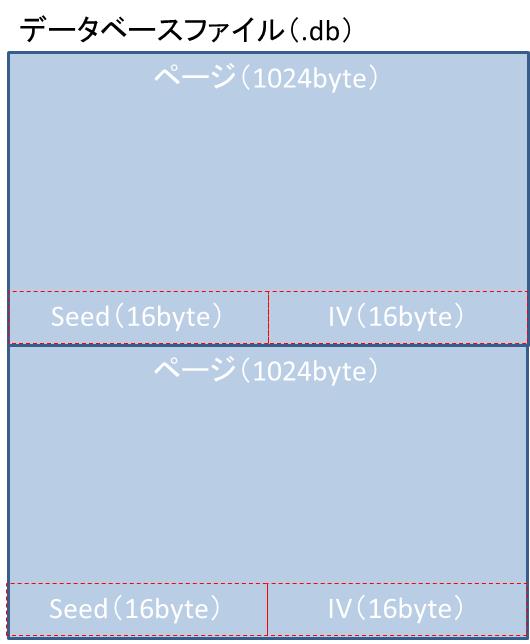


図 3‑12 データベースファイルの構造

* 試験の手順

試験仕様書に従って操作し、データベースファイルを取得して下さい。バイナリエディタで開いて、取得したデータベースファイルを比較して下さい。データベースファイルの取得は、File Explorerでできます（図 3‑3の右側参考）。

なお、試験番号3.1.2と3.1.3は操作が同じなので、試験番号3.1.2で取得した3つのデータベースファイルを試験番号3.1.3で利用して問題ありません。

例えば、試験番号3.1.2は、3つのデータベースファイルを取得し、それぞれIVが異なることを確認します。読み難いですが、図 3‑13それぞれIVが異なることが分かります。データベースファイル名は、File ExplorerでWindows等のPCに保存するときに変更しても問題ありません。試験番号3.1.2ではIVより前の992バイト部分が異なっていることを確認します。

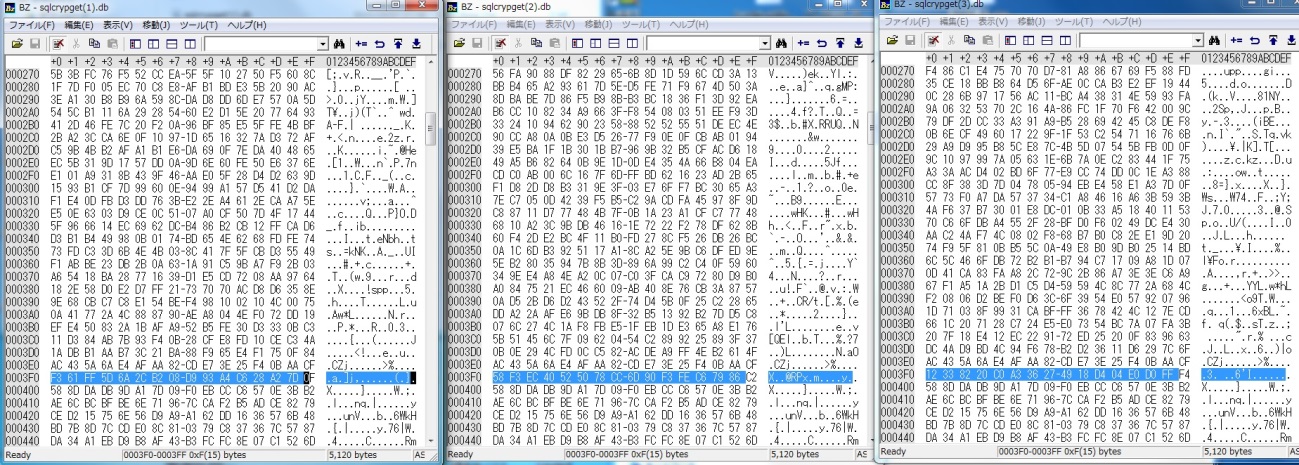


図 3‑13 データベースファイルの比較

### 試験番号4.X

* 試験の内容

ある端末上で作成したデータベースファイルが別の端末で操作できないことを確認します。

* 試験の背景

CrypgetSQLは端末認証機能を持っており、ある端末上で作成したデータベースファイルを別の端末で開けません。本試験では、この機能が正常に動作しているかを確認します。

* 試験アプリ

IntegrationTestApp1\_1を利用します。

* 注意事項

エミュレータの場合、端末を識別する端末IDがnullになります。エミュレータ2台で試験すると、同じ端末と認識され、開けてしまいます。

実機では、File Explorerから/data/data/以降のファイルにアクセスできません。SDカード(/mnt/sdcard/)を用いて試験します。

**※SDカードのパスはAndroid実機により異なる場合があります。適宜修正して下さい。**

* 試験の手順

1. エミュレータでIntegrationTestApp1\_1を実行します。

/data/data/[パッケージ名]/databases/temp.dbをPULLして下さい。パッケージ名は、以下になります。

jp.co.toshiba\_sol.android.sec.database.crypgetsql.test.integration.app1

1. ソースコードを修正します。

IntegrationTestApp1\_1のsrcの以下のファイルにおいて、onCreateメソッドの処理を表 3‑13に従って修正して下さい。赤太字のところが修正箇所です。（コメントアウトを変更しています）

jp.co.toshiba\_sol.android.sec.database.crypgetsql.test.integration.app1. IntegrationTestApp1Activity.java

表 3‑13 ソースコードの修正（試験番号4.X）

|  |  |
| --- | --- |
| **修正前** | // - ▼ - 試験No.4用コード  File dbFile**//**= new File("/mnt/sdcard/", "temp.db"); //実機用  = new File(  "/data/data/jp.co.toshiba\_sol.android.sec.database.crypgetsql.test.integration.app1/databases/", "temp.db"); //エミュレータ用 |
| **修正後** | // - ▼ - 試験No.4用コード  File dbFile = new File("/mnt/sdcard/", "temp.db"); //実機用  **//**= new File(  "/data/data/jp.co.toshiba\_sol.android.sec.database.crypgetsql.test.integration.app1/databases/", "temp.db"); //エミュレータ用 |

1. 実機の/mnt/sdcard/に、上記①でPULLしたtemp.dbをPUSHして下さい。
2. 上記②で修正したIntegrationTestApp1\_1を実機で起動します。

Locatを見て、以下の例外が発生していることを確認します。

例外クラス：jp.co.toshiba\_sol.android.sec.database.crypgetsql.SQLiteException

例外メッセージ：file is encrypted or is not a database

### 試験番号5.X

* 試験の内容

あるアプリケーションで作成したデータベースファイルが、別のアプリケーションから操作できないことを確認します。ただし、試験番号5.1.3は、同じアプリケーションで、アップデートした場合に、操作できることを確認します。

* 試験の背景

CrypgetSQLはアプリケーション認証機能を持っており、あるアプリケーション上で作成したデータベースファイルを別のアプリケーションで開けません。本試験では、この機能が正常に動作しているか確認します。ただし、試験番号5.1.3は、同じアプリケーションで、アップデートした場合に、操作できることを確認します。

* 試験アプリケーション

IntegrationTestApp1\_1とIntegrationTestApp5で試験します。

* 試験の手順

後述の注意事項を把握した上で、試験仕様書の通り実施して下さい。File Explorerでできます。図 3‑3をご参考下さい。

試験番号5.1.3は、Eclipse上で、ソースコードを一部変更して、再度実行します。

jp.co.toshiba\_sol.android.sec.database.crypgetsql.test.integration.app1.IntegrationTestApp1ActivityのonCreateメソッドの3行目にSystem.out.println("試験番号5.1.3");があるので、それをコメントアウト(もしくはコメントイン)して下さい。

* 注意事項

試験番号5.1.2において、Android4.3以前では、アプリケーションをアンインストールした後、もう一度インストールしてもUID（3.6.7節参照）が同じになってしまい、データベースファイルを開けてしまいます。アンインストール後、なんらかのアプリケーションをインストールしてから、試験に必要なアプリケーション（ここではIntegrationTestApp1\_1）をインストールして下さい。

### 試験番号6.X

* 試験の内容

CrypgetSQLのパスワードの違いにおける振る舞いを確認します。

* 試験アプリケーション

IntegrationTestApp6を用いて試験します。IntegrationTestApp6の説明を記載します。

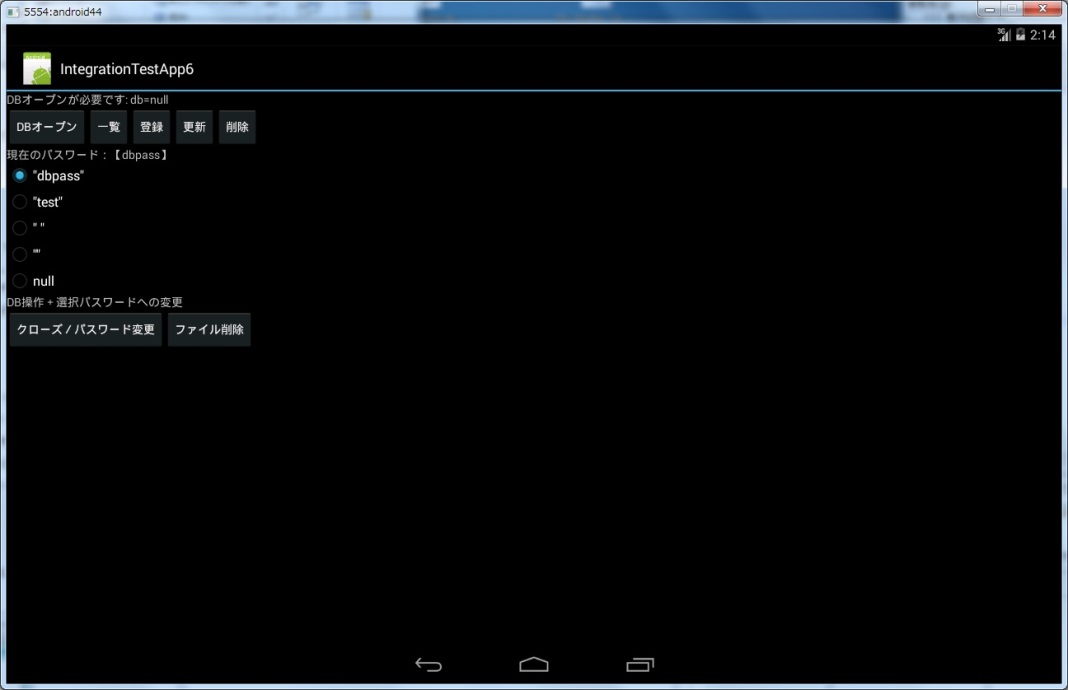


図 3‑14 IntegrationTestApp6

表 3‑14 IntegrationTestApp6のボタンの説明

|  |  |  |
| --- | --- | --- |
| **No** | **ボタン** | **処理の内容** |
| 1 | DBオープン | ラジオボタンで指定したパスワードにてDBをオープンします。DBをオープンする前に、No.7のボタンにより、クローズする必要があります。 |
| 2 | 一覧 | IntegrationTestApp1\_1と同じです。表 3‑11参照。 |
| 3 | 登録 | IntegrationTestApp1\_1と同じです。表 3‑11参照。 |
| 4 | 更新 | IntegrationTestApp1\_1と同じです。表 3‑11参照。 |
| 5 | 削除 | IntegrationTestApp1\_1と同じです。表 3‑11参照。 |
| 6 | 現在のパスワード | ラジオボタンで、利用するパスワードを指定できます。パスワードの変更の反映には、No.7のボタンでクローズし、No.1のボタンでオープンする必要があります。 |
| 7 | クローズ/パスワード変更 | DBをクローズします。パスワード変更するときなどに利用します。パスワード変更の手順は、No.6でラジオボタンの指定を変更した後、No.7のボタンをタップします。 |
| 8 | ファイル削除 | データベースファイルを削除します。暗号化したパスワードを別のパスワードで開こうとすると、画面上に「file is encrypted or is not a database」というエラーメッセージが表示されます。その場合、ファイル削除した後、No.1のボタンでオープンすれば、再度アプリの利用を継続できます。 |

* 試験の手順

試験仕様書の通り実施して下さい。難しくないので説明を省きます。ただし、図 3‑14の「現在のパスワード【dbpass】」のように画面上に現在のパスワードが表示されますので、意図しているパスワードで操作できていることを確認しながら実施して下さい。

* 備考

CrypgetSQLでは、パスワード””(空文字列)とnullが同じ扱いになります。そのため、試験番号6.1.7はエラーが発生しません。

なお、データベースを作成したときのパスワードと、読み込むときに設定したパスワードが異なる場合は、「file is encrypted or is not a database」が画面に表示されます。

### 試験番号7 .X

* 試験の内容

CrypgetSQLが作成したデータベースファイルのパーミッションが意図したとおりであることを確認します。

* 試験アプリ

どのアプリケーションでもよいですが、sqlcrypget.dbファイルのパーミッションを確認して下さい。（他のファイル名、例えば、IntegrationTestApp1\_1におけるtemp.dbは、試験番号4.Xを実施するためにデータベースファイルの作成方法が異なるので、パーミッションが違います。）

* 試験の手順

CrypgetSQLが作成したデータベースファイルのパーミッションが660(-rw-rw----)であることを確認します。パーミッションは、図 3‑3でも利用したFile Explorerで、図 3‑15のように確認できます。

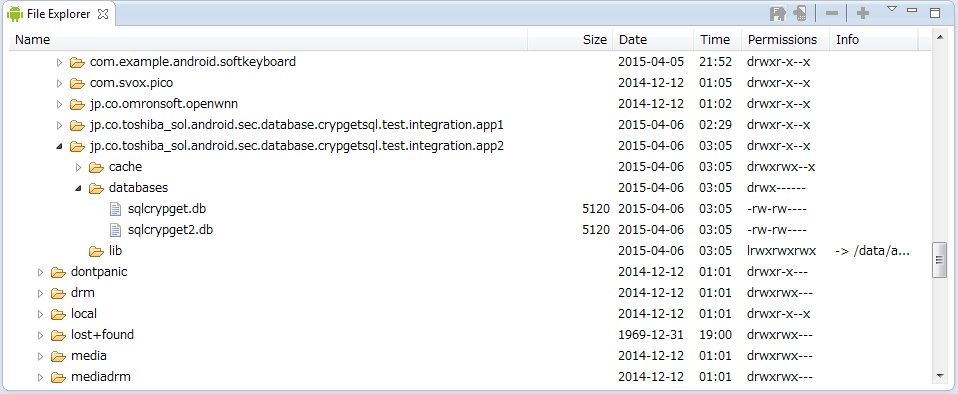


図 3‑15 File Explorerにおけるパーミッション

### 試験番号8.X

* 試験の内容

CrypgetSQL V1.1では、Android環境に応じて複数のsoファイルに分かれています。Java層のライブラリで、ライブラリが動作しているAndroidのバージョンを取得し、利用するsoファイルを切り替える処理があります。この処理が動作しているか確認します。CrypgetSQL V2/V3では、本書執筆2015年4月現在、soファイルが統合されているため、当試験は対象外になります（実施しません）。

* 試験の手順

soファイルは、Eclipseプロジェクトのlibs/armeabiフォルダ直下にあります（2.5.3項で配置したものです）。試験している環境が利用するsoファイルを表 3‑15から把握し、libs/armeabiから削除して下さい。

表 3‑15 CrypgetSQLのsoファイルとAndroidの関係（CrypgetSQL V1.1sp3）

|  |  |
| --- | --- |
| **Androidのバージョン** | **利用しているsoファイル** |
| Android4.0.3 (API Level 15) | libcrypgetsql\_android15.so  libdatabase\_crypgetsql15.so |
| Android4.1 (API Level 16)  Android4.2 (API Level 17) | libcrypgetsql\_android16.so  libdatabase\_crypgetsql16.so |
| Android4.4 (API Level 19) | libcrypgetsql\_android19.so  libdatabase\_crypgetsql19.so |

IntegrationTestApp1\_1で起動すると以下のようなリンクエラーが発生します。

Logcatで確認して下さい（Logcatについては、図 3‑3を参考下さい）。

|  |
| --- |
| java.lang.UnsatisfiedLinkError: Couldn't load crypgetsql\_android19 from loader dalvik.system.PathClassLoader[DexPathList[[zip file "/data/app/jp.co.toshiba\_sol.android.sec.database.crypgetsql.test.integration.app1-2.apk"],nativeLibraryDirectories=[/data/app-lib/jp.co.toshiba\_sol.android.sec.database.crypgetsql.test.integration.app1-2, /system/lib]]]: findLibrary returned null |

### 試験番号9.X

* 試験の内容

CrypgetSQLが対応しているプラットフォームで起動できることを確認します。サンプルアプリを起動し、操作できることを確認します。試験対象となっているプラットフォームでの試験は不要です（つまり、CrypgetSQL V1.1SP3の場合、試験対象プラットフォームがAndroid4.4であれば、Android4.0.3とAndroid4.1とAndroid4.2になります）。

* 試験の手順

サンプルアプリを起動して、試験仕様書の通り、操作して下さい。難しくないため、説明は省きます。

## 移行試験の手順

* 試験の内容

CrypgetSQLの旧バージョンから新バージョンに変えても、作成したデータベースファイルを継続利用できることを確認します。

* 試験アプリ

指定はありませんが、IntegrationTestApp1\_1で試験できます。

* 試験の手順

1. 試験するAndroid端末を起動します。
2. Eclipseで、旧バージョンのCrypgetSQLをlibsに配置して、プロジェクトをクリーンします。
3. Eclipseで、APKファイルを作成します。作成方法は4.2節に記載します。
4. Eclipseで、新バージョンのCrypgetSQLをlibsに配置して、プロジェクトをクリーンします。
5. Eclipseで、APKファイルを作成します。作成方法は4.2節に記載します。

上記③で作成したAPKを上書きしないように、別のフォルダに保存して下さい。また、上記③で作成したAPKと同じ鍵で署名して下さい。

1. コマンドプロンプトで、旧バージョンのCrypgetSQLで作成したAPK（上記③で作成）をインストールします。（4.1.5項参照。このときオプション-rの指定は不要です）
2. 端末の画面上で、インストールしたアプリケーション（上記⑥でインストール）でいくつかデータを登録します。
3. コマンドプロンプトで、新バージョンのCrypgetSQLで作成したAPK（上記⑤で作成）をインストールします。（4.1.5項参照。このときオプションに-rを指定して下さい）
4. 端末の画面上で、インストールしたアプリケーション（上記⑧でインストール）でデータの一覧表示を行います。上記⑦で登録したデータが表示されることを確認します。

* 備考

V1系からV2系への移行はできますが、V2系からV3系への移行はできません。V3系からV3系(例：V3.0からV3.1)はできることを想定しています[[2]](#footnote-2)。

# 付録

## ADBについて

Androidで試験を実施するに当たり、ADB（Android Debug Bridge）という、WindowsマシンからAndroid端末（実機・エミュレータ）を操作するためのツールが有用です。本項では、ADBについていくつか記載します。詳細な利用方法は、以下のサイトを確認下さい。

<http://developer.android.com/tools/help/adb.html>

### ADBの環境変数設定

コマンドプロンプトからadbコマンドが認識されない場合は、環境変数で以下のSDKのadb.exeにパスを通して下さい。なお、SDKをインストールしていない場合は、2.1節の①を参考にインストールして下さい。

[SDKのインストールフォルダ]\platform-tools\adb.exe

### ADBが認識している端末の確認

Devices View（図 2‑8参照）でも見えますが、adbでも見ることができます。

|  |
| --- |
| > adb devices |

### ADBの再起動

ときどきADBの動作がおかしくなることがあります。例えば、認識されていた端末が急に認識されなくなった場合に、ADBの再起動で直る場合があります。ADBの再起動は以下のコマンドになります。

|  |
| --- |
| > adb kill-server  > adb start-server |

なお、このstart-serverでエラーが発生する場合は、Windowsのタスクマネージャーから、adb.exeのプロセスを停止して、再度、adb start-serverを行うと、動く場合があります。

### Android端末のログ取得

|  |
| --- |
| > adb logcat |

詳細なオプションは以下のサイトを確認して下さい。

<http://developer.android.com/tools/help/logcat.html>

### APKのインストール、再インストール

APKフォルダがあるディレクトリでのAPKのインストールのコマンドは以下になります。

|  |
| --- |
| > adb install hogehoge.apk |

APKフォルダがあるディレクトリでのAPKの再インストールのコマンドは以下になります。

|  |
| --- |
| > adb install –r hogehoge.apk |

### アプリケーションのアンインストール

|  |
| --- |
| > adb uninstall [パッケージ名] |

なお、adbコマンドではなくとも、Android端末からアンインストールすることも可能です（図 3‑10の「アンインストール」ボタンをタップします）。

## APKの作成

本項では、APKファイルの作成方法を記載します。APKとは、android application packageの略で、Androidのアプリケーションに必要なファイル（プログラム本体、プログラムが利用する画像などのリソース、設定情報など）をZIP化したファイルフォーマットです。4.1.5節で示した方法などで、Android端末にインストールすることができます。

1. Eclipseのメニューの「File」→「Export…」をクリックします。
2. 次のSelect画面では「Android」の「Export Android Application」を選択し、「Next >」ボタンをクリックします（図 4‑1参照）。

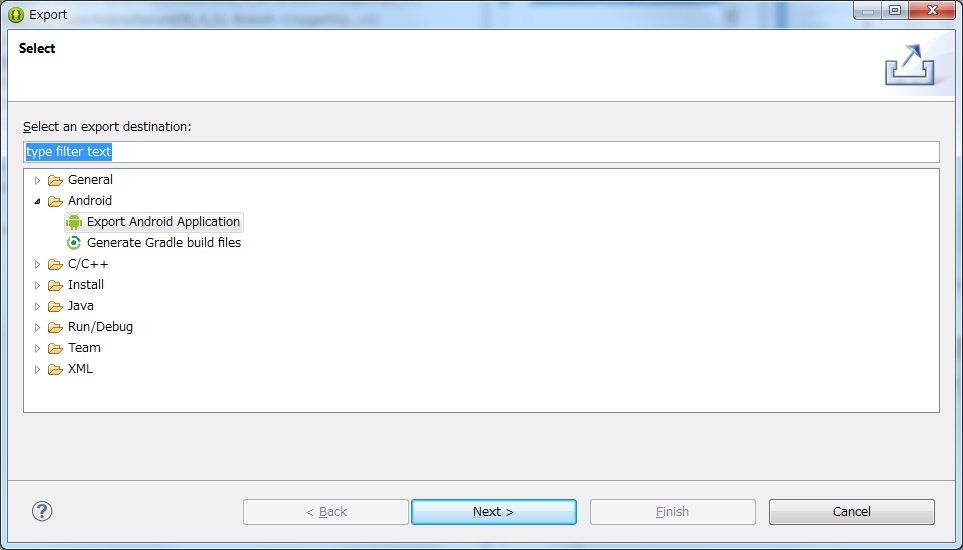


図 4‑1 Export Select

1. 次のProject Checks画面では「Browse…」ボタンをクリックし、APKファイルを作成するプロジェクトを選択します。その後、「Next >」ボタンをクリックします。
2. 次のKeystore selection画面では、APKの署名に利用する鍵を保存しているKeystoreを指定します。

既存のものを利用する場合は、「Use existing keystore」をチェックし、「Location」でKeystoreのファイルを指定します。Keystoreはパスワードで保護されているので、Keystoreを開くためのパスワードを「Password」欄に入力します。その後「Next>」ボタンをクリックします。

新規にKeystoreを作成する場合は、「Create new keystore」をチェックし、「Location」でKeystoreファイルの保存場所を指定し、Keystore保護用のパスワードを「Password」欄と「Confirm」欄に入力します。その後「Next>」ボタンをクリックします。

「Use existing keystore」または「Create new keystore」のどちらを選択したかで、後述の手順が変わります。

1. Key alias selection画面についてです。上記④で「Use existing keystore」をチェックした場合、Keystoreから既存の鍵を利用する「Use existing key」か、新規に鍵を作成する「Create new key」か、をチェックします。

「Use existing key」の場合、Alias（鍵を指定するための値（別名））と、その鍵を利用するパスワードを入力し、「Next>」ボタンをクリックし、後述⑦に進みます。

「Create new key」の場合、「Next>」ボタンをクリックし、後述⑥に進みます。

1. Key Creation画面についてです。上記④で「Create new keystore」をチェックした場合、もしくは、上記⑤で「Create new key」をチェックした場合にこの画面になります。必要事項を入力して「Next>」ボタンをクリックして下さい。
2. 最後にDestination and key/certificate checks画面で、APKの出力フォルダを指定して、「Finish」ボタンをクリックします。

## 文字コードの設定

アプリケーションの出力が文字化けしている場合、文字コードの設定を確認して下さい。ソースコードの文字コードはUTF-8またはMS932です。

Eclipseの試験アプリのプロジェクトを右クリックし、「Properties」をクリックします（図 2‑11参考）。左側の「Resource」にフォーカスを当て、右側の「Text file encoding」で「Other」をチェックし、文字コード名を入力するかプルダウンから選択します。文字コード変更後は、Eclipseのメニューの「Project」の「Clean」で再コンパイルが必要です。

# 参考文献

[1] Staveware Mobile for AndroidTM / Developers Toolkit AndroidTM アプリケーション開発・テストガイド

1. https://source.android.com/compatibility/cts-intro.html [↑](#footnote-ref-1)
2. 試験対象について、できると想定しているものについて試験をお願いします。

   試験対象がV3.1であれば、V3.0からV3.1への移行の確認のみになります。

   試験対象がV2.0であれば、V1系で利用されているバージョンを生技殿に確認し、そのバージョンからV2.0への移行の確認をお願いします。 [↑](#footnote-ref-2)