JavaScriptだけでできるモバイルデバイス向けアプリ開発

Titanium Mobile

ではじめるiPhoneアプリケーション開発

O'TZLLY

@donayama

目次

はじめに

Appcelerator Titanium Mobile について

Appcelerator Titanium Mobile とは Appcelerator 社が提供するモバイルデバイス向けのソフトウェア開発環境です。

ちなみに WebCast によると Titanium の読み方は「チタニウム」ではなく「タイタニウム」と発音するようです。

プログラミング言語としては **JavaScript のみを利用**し、Objective-C や Java でコーディングすることなく iPhone SDK/Android SDK 向けのネイティブアプリケーションを開発できることがウリとなっています。

もちろん JavaScript だからといって Web インタフェイスのみをサポートしているわけではありません。

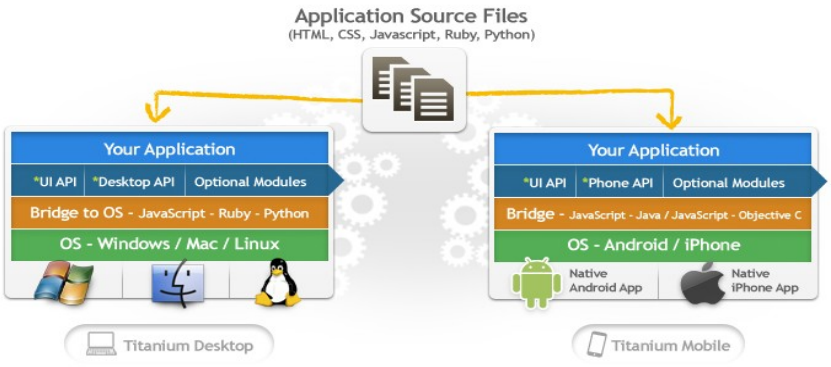
以下に挙げる全ての機能を実現できるようになっています。

- ラベル・テキスト入力・ボタン・スライダーなどの豊富なプラットフォームネイティブなユーザインタフェイス
- 画 像・動画の再生・録音・撮影・録画
- 地図表示
- ファイルシステムへのアクセス
- HTTP ベースの 非同期ネットワーク通信
- ハードウェアデバイス(カメラ・GPS・電子コンパス・加速度センサ)の利用
- デバイス内 データベース(連絡先など)の操作(予定)
- SQLite データベースの I/O

プラットフォームに依存し た一部の機能を利用しない限り、対応プラットフォームすべてに対してに同じコードを転用できると謳われており、JavaScript に よる開発効率の向上だけではなくマルチプラットフォーム展開を少ないコストで実現できることも魅力のひとつとなっています。

現時点で対応 しているプラットフォームとしては前掲した iPhone SDK と Android SDK のみになりますが、今後も追加されていく予定があります。(iPad は 2010 年 3 月対応予定)

また、兄弟製品として Titanium Desktop があり、こちらは Windows, Mac OSX, Linux というマルチプラットフォームで動作するデスクトップアプリケーションを開発できる環境も提供されています。



1st Step : はじめの一歩

Appcelerator Titanium Mobile で iPhone アプリ開発を始めよう!!

前提環境(iPhone SDK)

Titanium Mobile 自体は Titanium Developer という Desktop, Mobile でそれぞれ共用する開発環境を利用します。TItanium Developer の動作するプラットフォームとして、Windows, Mac OSX, Linux をサポートしています。

しかし本稿で取り上げる iPhone アプリケーションの開発に関しては iPhone SDK が必要になります。 そのため必然的に動作する環境としては Mac OSX のみとなってしまいます。

その点、ご注意ください。

iPhone SDK のインストールについて

Titanium Mobile での開発を始める前に、上記のように iPhone SDK のインストールを行っておく必要があります。

この SDK のセットアップから開発環境構築については下記記事をご参照ください。

目指せ！iPhone アプ リ開発エキスパート	gihyo.jp	http://gihyo.jp/dev/serial/01/iphone/0002

また、本稿では取り上げませんが、実機での動作検証（ならびに AppStore?で の配布）を行うためには iPhone Developer Program への登録が必要となります。

それについては次の記事をご参照く ださい。

目指せ！iPhone アプリ開発エキスパート	gihyo.jp	http://gihyo.jp/dev/serial/01/iphone/0009

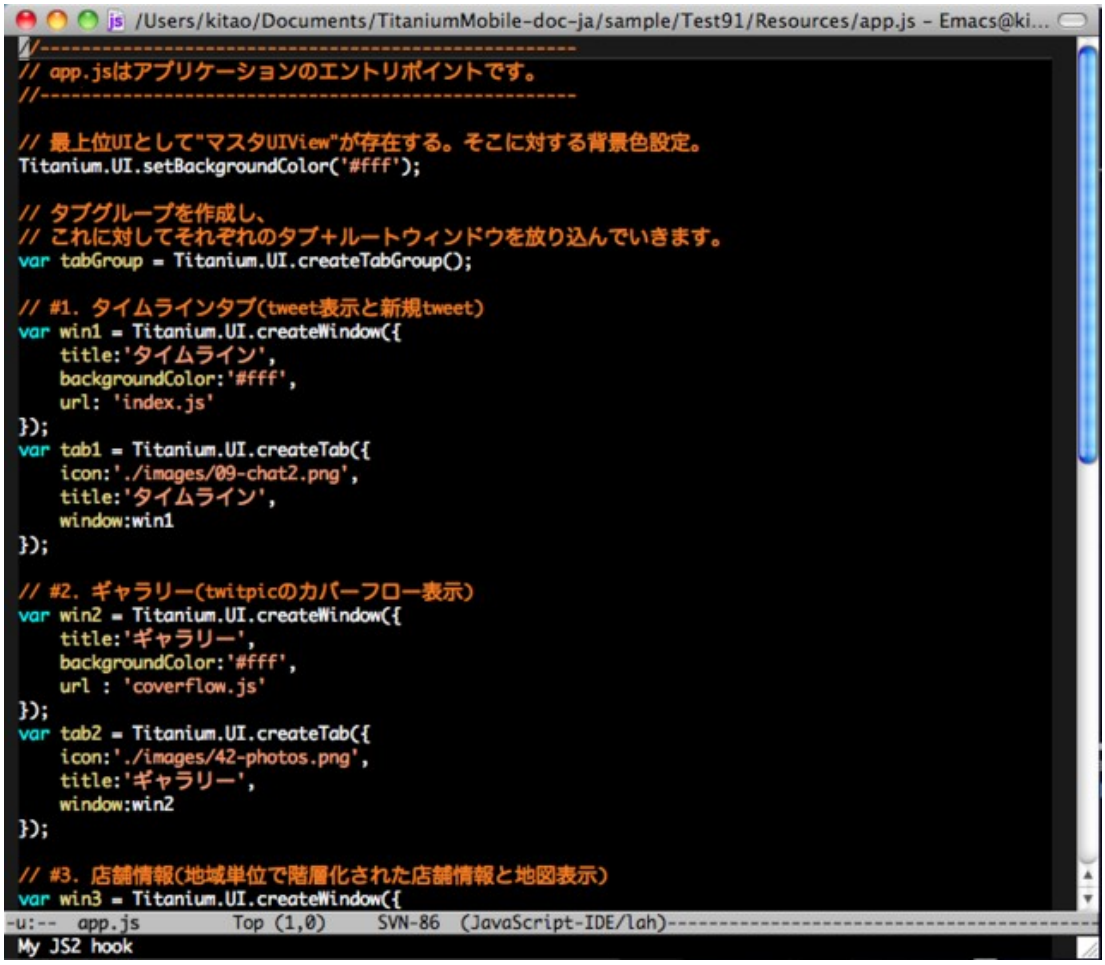
エディタの準備

Titanium は IDE ではありません。

むしろビルド専 用環境といってもいいフロントエンドプログラムなので、ソースプログラムの修正すらすることができませんので、なんらかのエディタを準備してください。

Appcelerator の中の人たちは TextMate などを使ってるようですが日本語入力に難があるので、手になじんだエディタで結構です。できれば JavaScript の 文法ハイライト機能や入力支援があるソフトの方が開発効率もグンとあがると思います。

ちなみに筆者は Carbon Emacs + JS2 環境で開発をしています。



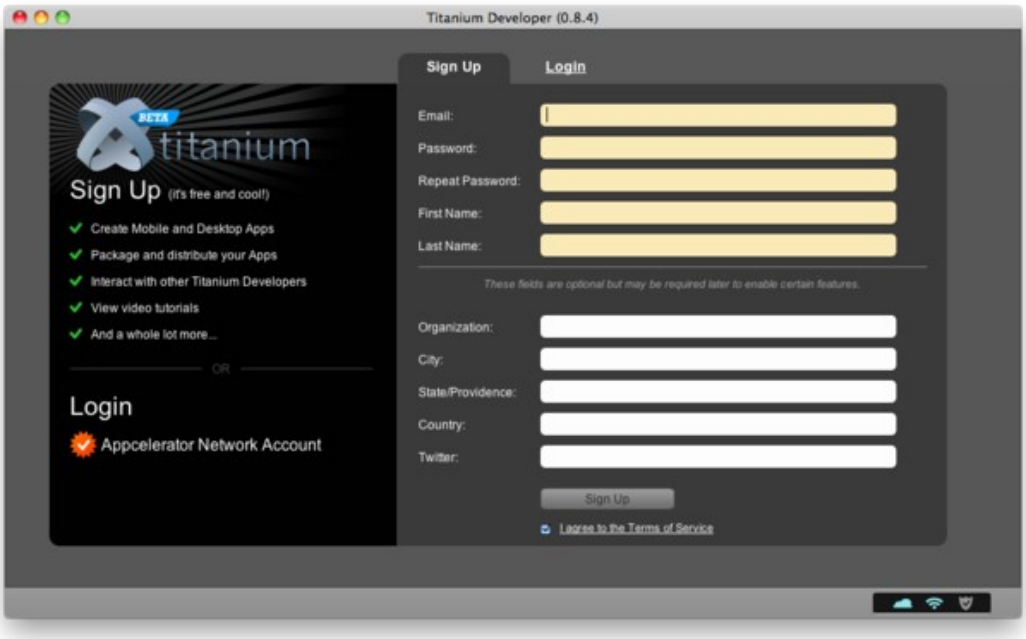
Titanium Developer のインストールと起動

こちら のページを開くと自動的にインストーライメージファイル(Titanium Developer.dmg)をダウンロード開始しますので、完了後インストールしてください。

ちなみにアプリケーションが 「Titanium Developer」となっていますが、これは Desktop・Mobile で共用されるツールであるため、このように名づけられています。

起動画面

インストール後に Titanium Developer 起動すると、次のような画面が表示されます。



Titanium を使うに当たって、この画面でアカウントを作る必要があります。

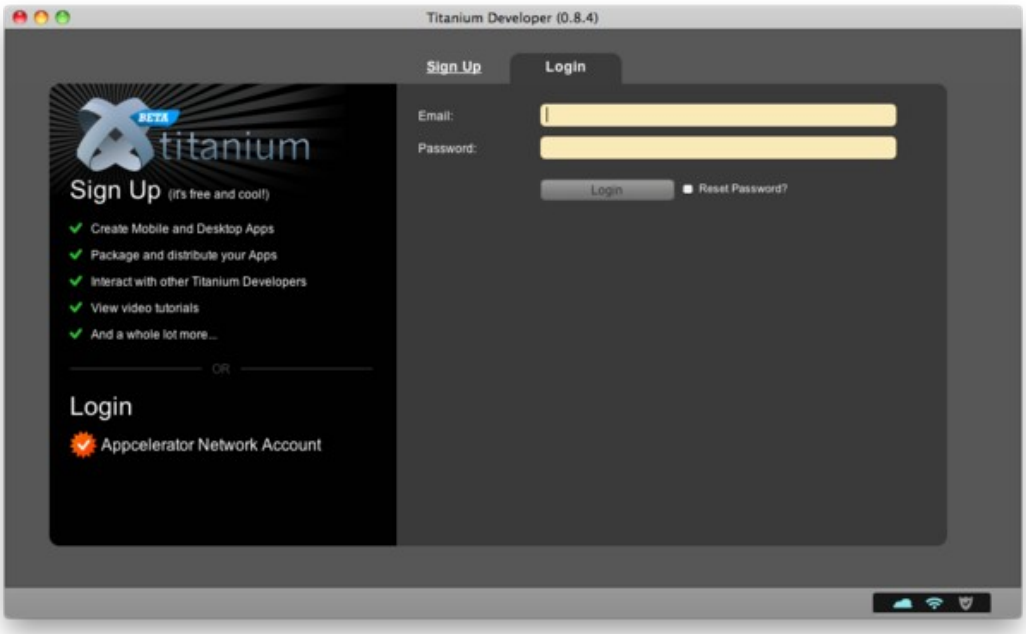
ログイン ID となるメールアドレスとパスワード、姓名といった個人情報を登録しないと Titanium Developer 環境が利用できませんので必ず登録してください。（twitter id をいれておくと勝手に公式アカウントがフォローしてくれます）

ここで登録したアカウントで Appcelerator のサポートページなどへのログインにも利用 します。

アカウント作成が完了すると、Titanium Developer の新規プロジェクト作成画面を開きます。

ここから、新規プロジェクトを作成し たり、既存のプロジェクトをインポートできます。

アプリケーション終了／(Titanium Developer 上での)ログアウト後はログインタブに切り替え、登録したメールアドレスとパスワードを入力して、立ち上げる流れになります。



Kitchen Sink を触ってみる ～ iPhone シミュレータの起動

インストールしたら早速「Hello, world 的なプログラムを始めましょう！」……と言いたいところですが、まずは開発元の Appcelerator から提供されているデモアプリケーショ ン「KitchenSink」 を利用して、アプリケーションのビルドから動作確認（iPhone シミュレータ上で、ですが）の流れを体験したいと思います。

KitchenSink は Titanium Mobile の UI 部品、API のカテゴリ単位に機能を個別に紹介した機能カタログにあたるデモアプリケーションです。

実際に Titanium Mobile での開発をすすめる「KitchenSink の これと同じようなことをやりたい」ということになります。その際、API リファレンスをひも解くよりも、KitchenSink の 該当部分のソースをコードスニペットとして利用するほうが早かったりしますので、KitchenSink で Titanium Mobile で「何が」「どこまで」できるのかを知っておく必要があります。

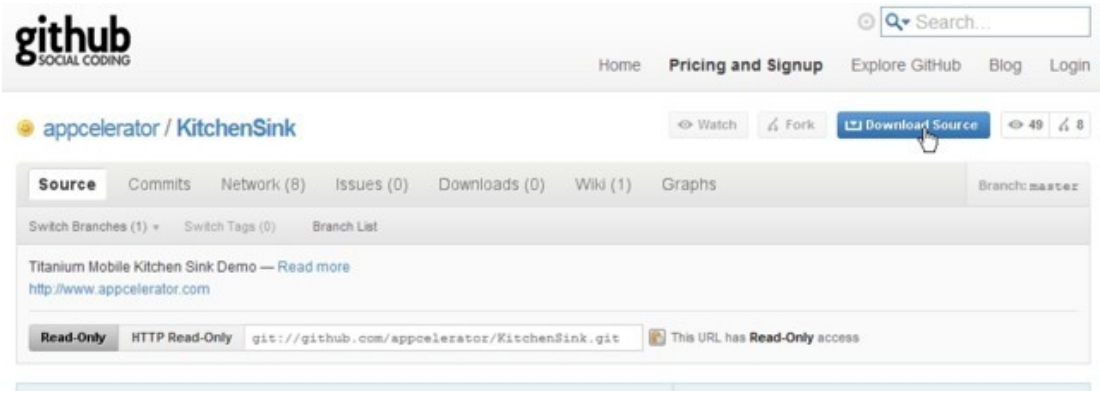
Kitchen Sink をダウンロードする

KitchenSink だけではなく、Appcelerator の製品はオープンソースソフトウェアとして開発がされており、ソースコードは github のレポジトリで公開 されています。

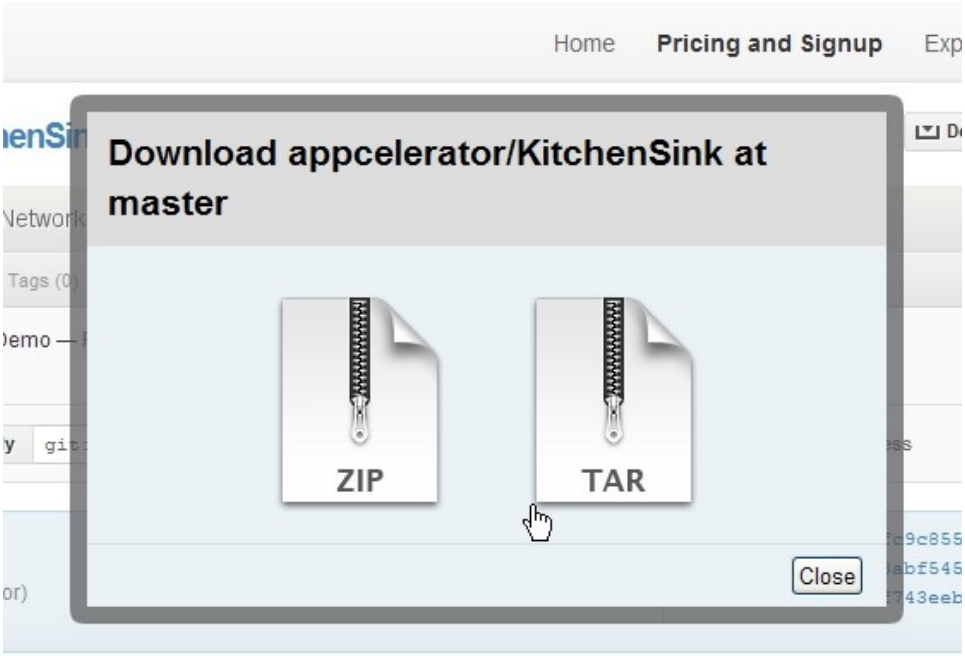
KitchenSink も同様に github で公開されています。

http://github.com/appcelerator/KitchenSink
取得だけをする場合 git クライアントは不要です。

まず上記リンクにアクセスし、github の右上 「Download Source」を選択します。



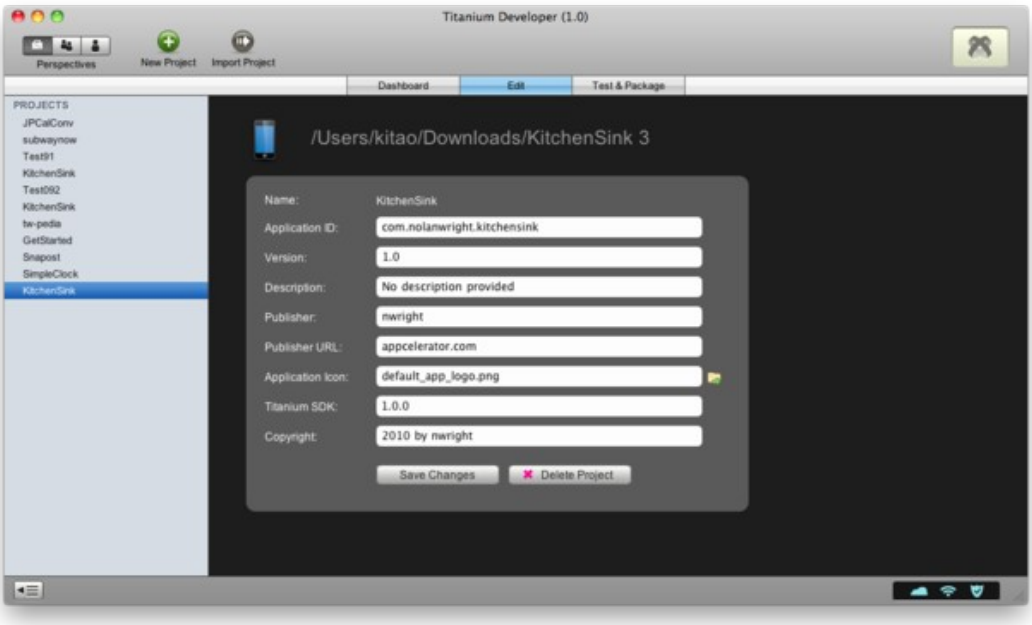
続いて表示される画面で zip アイコンか、tar アイコンのいずれかを選択し、ダウンロードします。完了後、展開してください。



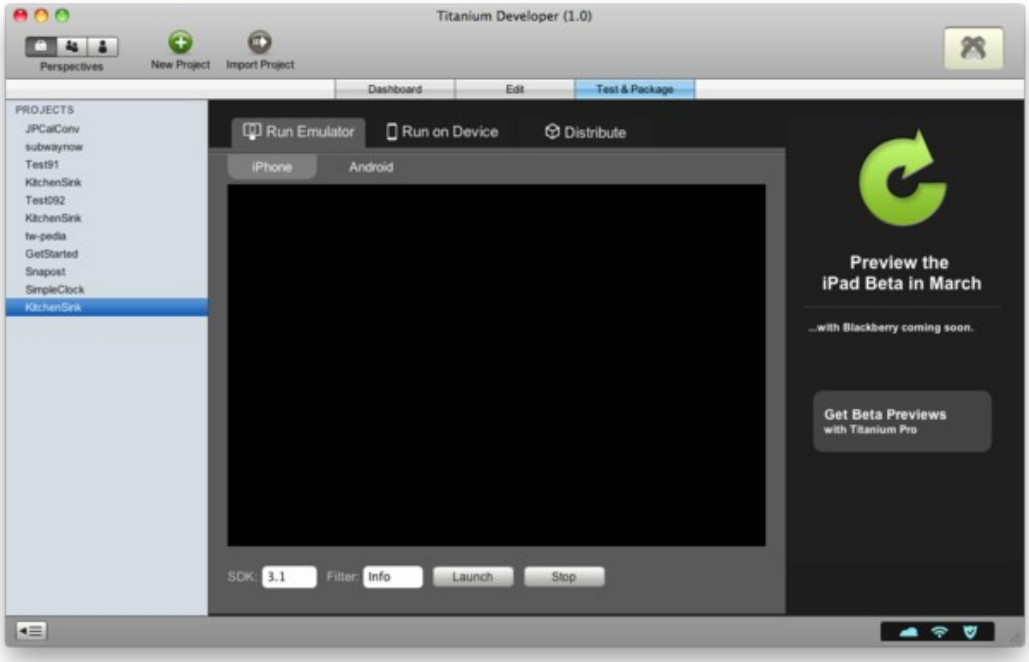
Kitchen Sink を動かす

展開した KitchenSink の アーカイブから最新のプロジェクトを Titanium Developer に取り込みます。

Titanium Developer のツールバー「Import Project」を選択して、先ほど展開した先の `/1.0.x/KitchenSink` フォルダを選択後「OK」ボタンをクリックすると、KitchenSink プ ロジェクトが PROJECTS 一覧に追加され、プロジェクト設定の編集画面が表示されます。



続いて、画面上部右の「Test & Package」を選択すると、タブで構成された処理選択が表示されます。



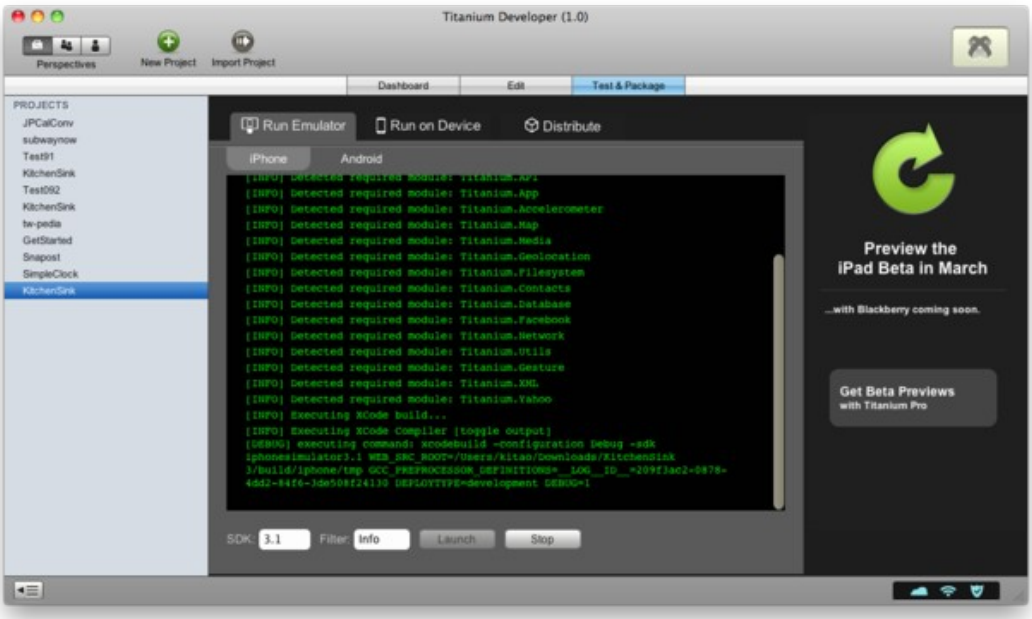
「Run Emulator」タブの下方に並ぶプラットフォームから「iPhone」を選択します。

画面中央を陣取る黒 い部分は実行時のログ表示部になります。ログレベルによって、表示内容の範囲が代わります（情報であったりエラーメッセージであったり）。

画 面下部には使用する SDK のバージョンやログレベルを選択するコンボボックスとシミュレータの起動と終了をするためのボタンが付いています。

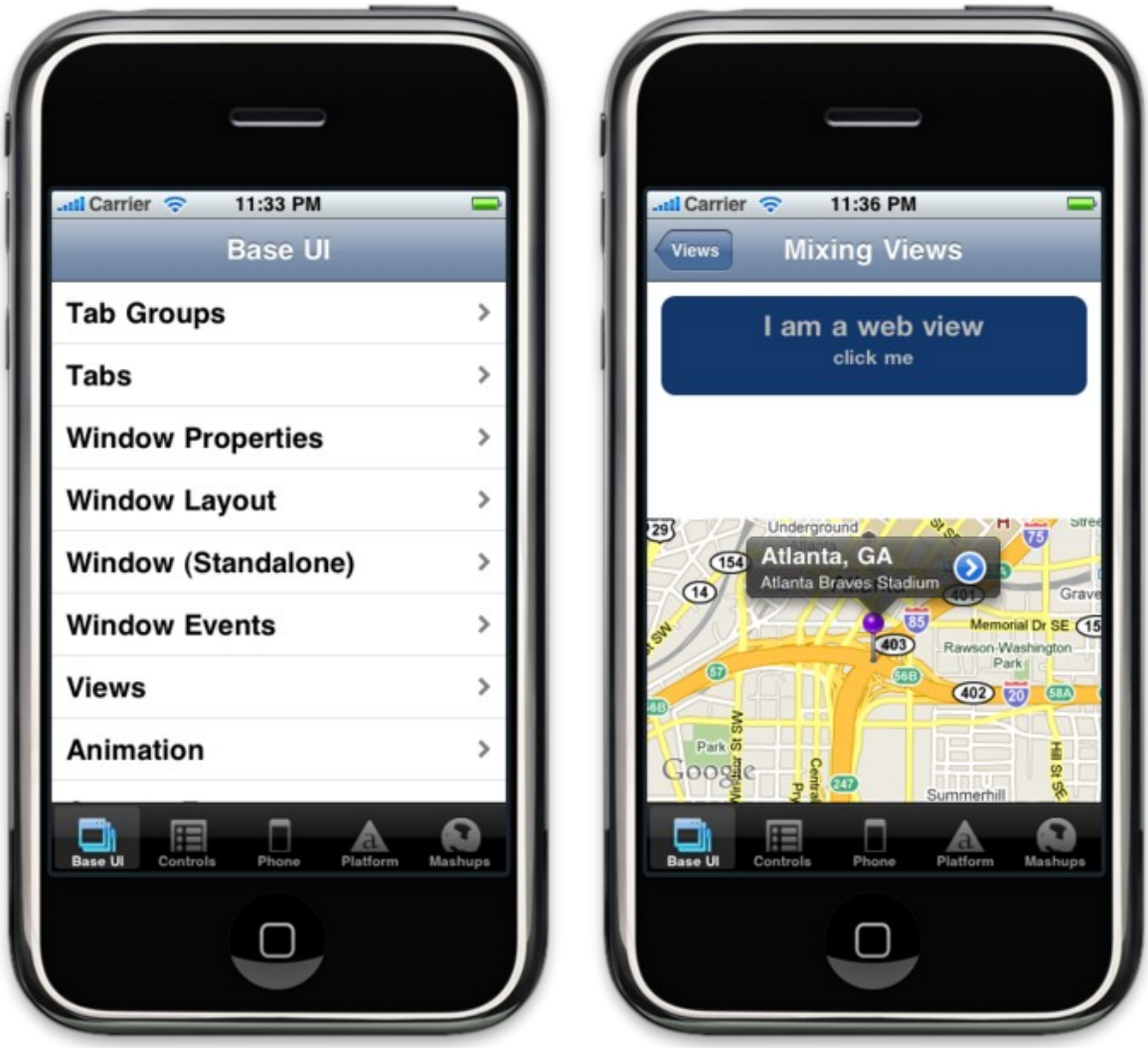
「Launch 」（ウィンドウ下部左側のボタン）をクリックすると、ビルドが開始されます。 マシンスペックなどにも依存しますが、初回時（と Titanium Developer の立ち上げ直後）は比較的ビルド時間がかかる傾向

があるようです。



表示内容を見ていただけば、内部で Xcode のコンパイラを呼び出しており iPhone ネイティブのバイナリに変換しているということが 分かっていただけのではないでしょうか。

ビルド完了後、自動的に iPhone シミュレータが起動し、更にビルドされたアプリケーションも 自動起動します。



あれこれ言うより実際にシミュレータ上のデモを見て、どのようなことが Titanium Mobile で実現できるかを体感してください。

(残念ながらデバイスハードウェア系（カメラ・加速度センサ・GPS・電子コンパスな ど）を中心に iPhone シミュレータでの動作がサポートされていない機能は使用できません。実機に転送する必要がありますが、本稿では触れません)

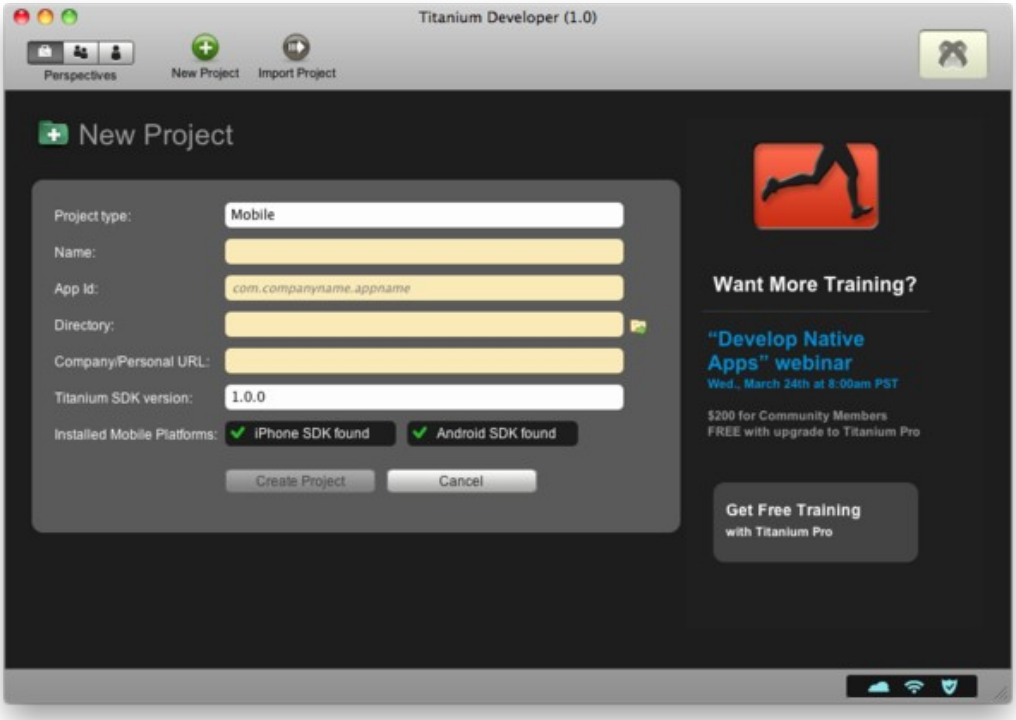
テストプロジェクトの作成からプロジェクト作成後のフォルダ構成

一通りの KitchenSink の 体験をしたら、次は実際にプログラムしていきましょう。

プロジェクトの作成

まずはプロジェクトの新規作成をします。

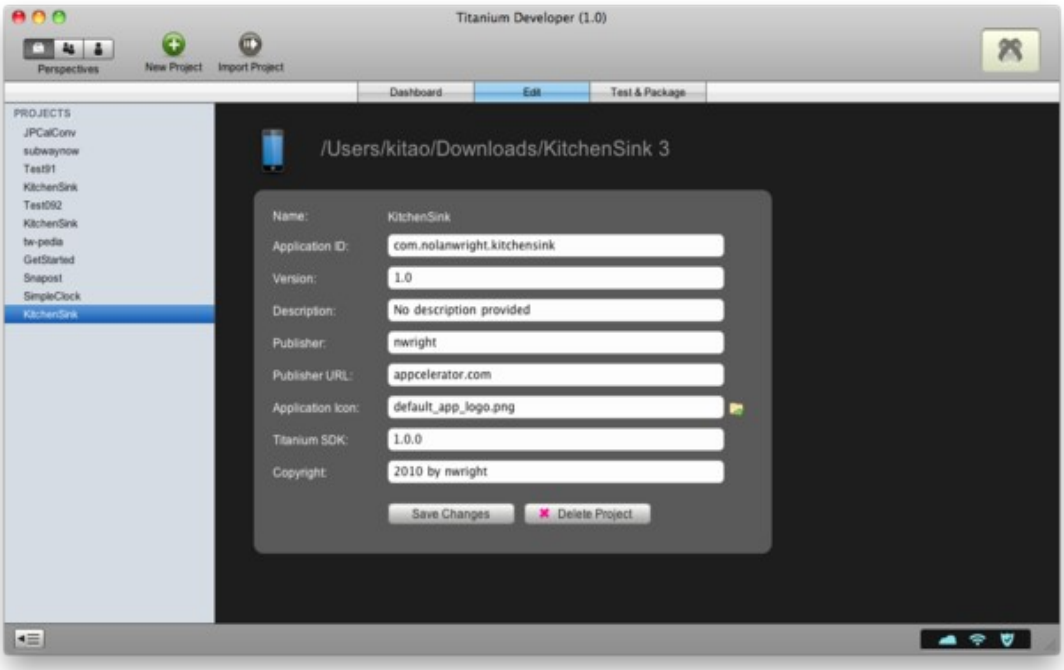
Titanium Developer のツールバーから「New Project」をクリックすると、New Project 画面が立ち上がるので、必ず Project type をデフォルトの「Desktop」から「Mobile」にするようにしてください。 （Mobile に切り替えた後、iPhone SDK と Android SDK の検出を行うため、切替え直後、画面の下部の Installed Mobile Platforms は緑チェックマークではありません）



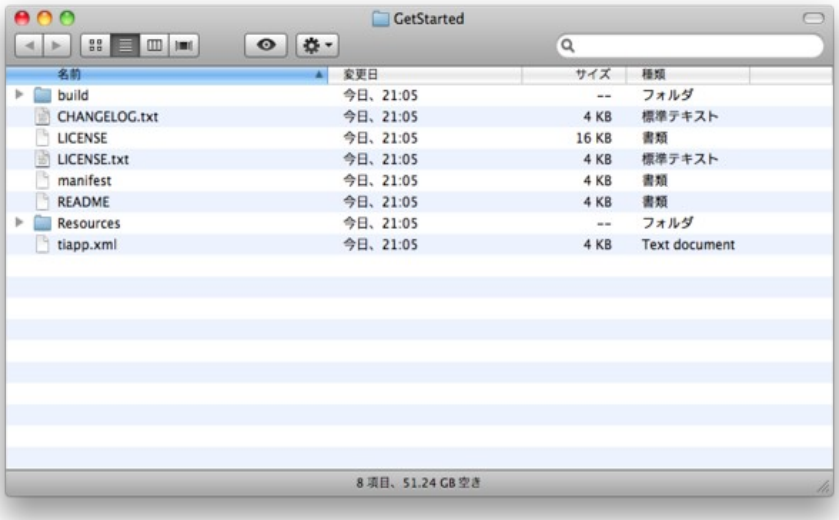
ここでは次のように入れてみましょう。

項目	設定値	補足
Project type	Mobile	モバイルデバイス向けの 際は必ず Mobile を選択してください。
Name	GetStarted	アプリケーションの名前です。残念ながら日本語文字を入れると現 バージョンではビルド時にエラーになります。また長すぎる名前をつけるとホーム画面で「…」で省略されるので注意しましょう（英数字 11 桁まで）。
App Id	com.yourdomain.applicationname	アプリケーションを一意 に判別するための ID です。Java の名前空間のようにドメイン名を反転させた名前の付け方をすることが推奨されていま す。（例：com.twitter.your_id.testapp）
Directory	/Users/your_account	ソースの生成先フォル ダ。このフォルダに上記 Name で指定した名前のフォルダが作られます
Company / Personal URL	http://www.yourdomain.com/	Publisher URL という項目に反映されます
Titanium SDK version	1.0.0	特段の問題がない限り、デフォルトになっている最新バージョンを使います

そして「Create Project」ボタンをクリックすると、ひな形に基づいたフォルダ・ファイル群が Directory/Name なフォルダに自動生成され、画面もプロジェクト編集画面に遷移します。



このプロジェクト編集画面の上部にあるパス名(終端は省略されることもありますが)を クリックすると、Finder が起動し、該当フォルダを開いてくれます。



プロジェクトのフォルダ構成

フォルダには以下のようなファイルが生成されています。

- CHANGELOG.txt
- LICENSE
- LICENSE.txt
- README
- manifest
- tiapp.xml
- /Resources
    - KS_nav_ui.png

        - KS_nav_views.png
        - app.js
        - /android
        - /iphone
            - Default.png
            - appicon.png
- /build
    - (以下略)

それぞれについて代表的なものをかなり簡単に説明します。

manifest	アプリケーション定義が 記述されています。
tiapp.xml	アプリケーション定義が記述されています。
/build	ビルド結果を格納する フォルダ。直接触る事はあまりないはずです。
/Resources	実際のアプリケーション開発をするソースなどはここに格納しま す。
/Resources/app.js	アプリケーションのエントリポイントとなるスクリプトファイルです。
/Resources/android	Android 向けに依 存したソース・リソースを格納します。
/Resources/iphone	iPhone 向けに依存したソース・リソースを格納します。

最初のふたつはアプリケーション定義に関するファイルです。

一部の例外を除き、プロジェクト編集画面で設定できる内容なので、本稿では触 れません。

最後のふたつに付いてはビルド時に一時的に iPhone 用は /Resources に/Resources/iphone をマージされた（Android も同様）状態とすることで、機種依存機能を吸収できるように なっています。（上記状態では iPhone 用のスプラッシュスクリーン画像とアプリケーションアイコンが収納されています）

ちなみに実行すると...

次のよう なシンプルな画面が立ち上がります。

すでに KitchenSink の 動きを見てるので拍子抜けするぐらいシンプルですね。

タブのボタンを押すと画面が切り替わることも含めて確認してください。

app.js からすべてはじまる

Titanium Mobile のアプリケーションはすべて Resources/app.js から起動されます。

app.js を見る

実際に生成されたソースコード を見てみましょう。(ムダな空行を間引いています)

```
// this sets the background color of the master UIView (when there are no windows/tab
groups on it)
Titanium.UI.setBackgroundColor('#000');
// create tab group
var tabGroup = Titanium.UI.createTabGroup();
// create base UI tab and root window
var win1 = Titanium.UI.createWindow({
  title:'Tab 1',
  backgroundColor:'#fff'
});
var tab1 = Titanium.UI.createTab({
  icon:'KS_nav_views.png',
  title:'Tab 1',
  window:win1
```

```
});
var label1 = Titanium.UI.createLabel({
  color:'#999',
  text:'I am Window 1',
  font:{fontSize:20,fontFamily:'Helvetica Neue'}
});
win1.add(label1);
// create controls tab and root window
var win2 = Titanium.UI.createWindow({
  title:'Tab 2',
  backgroundColor:'#fff'
});
var tab2 = Titanium.UI.createTab({
  icon:'KS_nav_ui.png',
  title:'Tab 2',
  window:win2
});
var label2 = Titanium.UI.createLabel({
  color:'#999',
  text:'I am Window 2',
  font:{fontSize:20,fontFamily:'Helvetica Neue'}
});
win2.add(label2);
//  add tabs
tabGroup.addTab(tab1);
tabGroup.addTab(tab2);
// open tab group
tabGroup.open();
```

やっていることは次のような流れです。

- ベー スの背景色を黒(#000)にする
- TabGroup を 生成し、
  - そこに格納するための Tab(タブのアイコンやタイトルも指定している)と Window を生成する
    - Window には Label を配置する
    - Tab に Window を格納し
  - TabGroup に Tab を格納し、
- TabGroup を 表示する

この結果、次のような親子関係になっています。

- app.js (MasterUIView)
  - tabGroup
    - tab1
      - win1
        - label1
    - tab2
      - win2
        - label2

app.js から Window 単位のスクリプト分離

このように画面 生成・ロジック記述（そしてここには含まれませんがイベント処理）も含めてすべて JavaScript で 記述するというシンプルな思想によって Titanium Mobile アプリケーションは開発していくことになります。

とはいえ、このま まアプリケーション規模が増えると際限なく app.js が膨らんでいきます。スコープ管理も

大変で可読性も悪くなります。

そこで別の JavaScript ファ イルを切り出してみましょう。

まず現状の tab1 関連を再度抜き出してみます。

```
var win1 = Titanium.UI.createWindow({
  title:'Tab 1',
  backgroundColor:'#fff'
});
var tab1 = Titanium.UI.createTab({
  icon:'KS_nav_views.png',
  title:'Tab 1',
  window:win1
});
var label1 = Titanium.UI.createLabel({
  color:'#999',
  text:'I am Window 1',
  font:{fontSize:20,fontFamily:'Helvetica Neue'}
});
win1.add(label1);
```

現在 Label がひとつだからいいですが、複数の UI 部品が混ざってくると可読性は一気に悪くなります。

そのため、上記ソースを次のように 変えます。

```
var win1 = Titanium.UI.createWindow({
  title:'Tab 1',
  backgroundColor:'#fff',
  url: 'win1.js'
});
var tab1 = Titanium.UI.createTab({
  icon:'KS_nav_views.png',
  title:'Tab 1',
  window:win1
});
```

変 わったのは createWindow の引数に url プロパティが追加されている部分、label1 の定義と win1 への追加部分が削除されている部分で す。

これにより win1 の表示時に win1.js という外部スクリプトファイルをロードし、それを実行するという流れになります。

Resorces フォルダに呼び出される側の win1.js を作成します。

```
// /Resources/win1.js
var label1 = Titanium.UI.createLabel({
  color:'#999',
  text:'I am Window 1',
  font:{fontSize:20,fontFamily:'Helvetica Neue'}
});
Titanium.UI.currentWindow.add(label1);
```

win1.js で新しく登場したのが Titanium.UI.currentWindow というプロパティです。

win1.js を評価している最中の currentWindow とはすなわち app.js における win1 なので、ここには win1 のオブジェクトがセットされます。

そ のため最終的な結果は同じようになります。

(実際には生成処理の評価タイミングが違うために表示の待ち時間が発生するようになる)

こ のように window 単位で js ファイルを分割していくと、処理と画面をコンパクトに記述でき、また管理もしやすくなります。

部品配置とイベントリスナへの登録

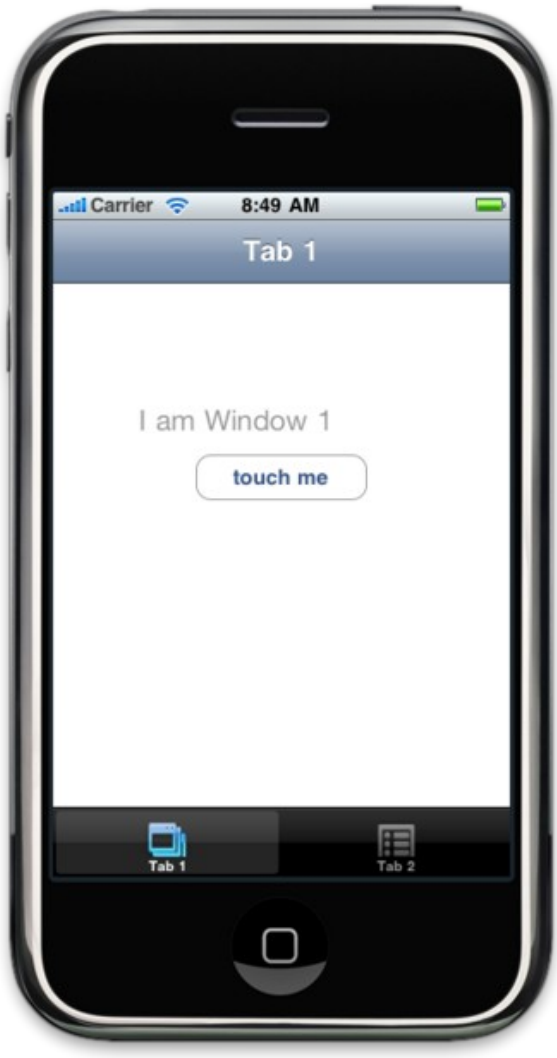
もう少し win1.js に画面部品を追加してしていこうと思います。

```
// よく使うのでこのように再定義しておくと便利です。
var win    = Titanium.UI.currentWindow;
// 一段 View を間に挟むようにします。
var view   = Ti.UI.createView();
// label1 に表示位置の指定を追加します。
var label1 = Titanium.UI.createLabel({
  color:'#999',
  text:'I am Window 1',
  font:{fontSize:20,fontFamily:'Helvetica Neue'},
  height: 32,
  width: 200,
  top:80
});
// button1 を生成します。
var button1 = Ti.UI.createButton({
  title: 'touch me',
  height: 32,
  width: 120,
  top: 120
});
// win←view に部品を追加します
view.add(label1);
view.add(button1);
win.add(view);
// ボタンクリック時のイベント
button1.addEventListener('click', function(){
  Titanium.UI.createAlertDialog({
    title:   'タイトル',
    message: 'クリックされました'
  }).show();
});
```

ボタンを一つ追加し、そのクリックイベントを定義してみました。

部品の位置の指定方法は CSS での指定に準拠しています。

幅(width)や高さ(height)を指定しないと、コンテナ領域の全体に設定されることもあるの で、コントロールの大きさや位置は明示的に指定するべきです。



またイベントリスナの登録については（一部の例外をのぞき）次のように 記述されます。

```
object.addEventListener('EVENT_NAME', function(event){
   // コールバック無名関数
});
```

まずは win2 側も js ファイル化してみて、同じようにボタンをいくつか配置してみて慣れてみましょう。

2nd Step : Ui カタログ

ユーザインターフェイス概説

Titanium Mobile はイメージ的に Objective-C へのプリプロセッサとして動きます。

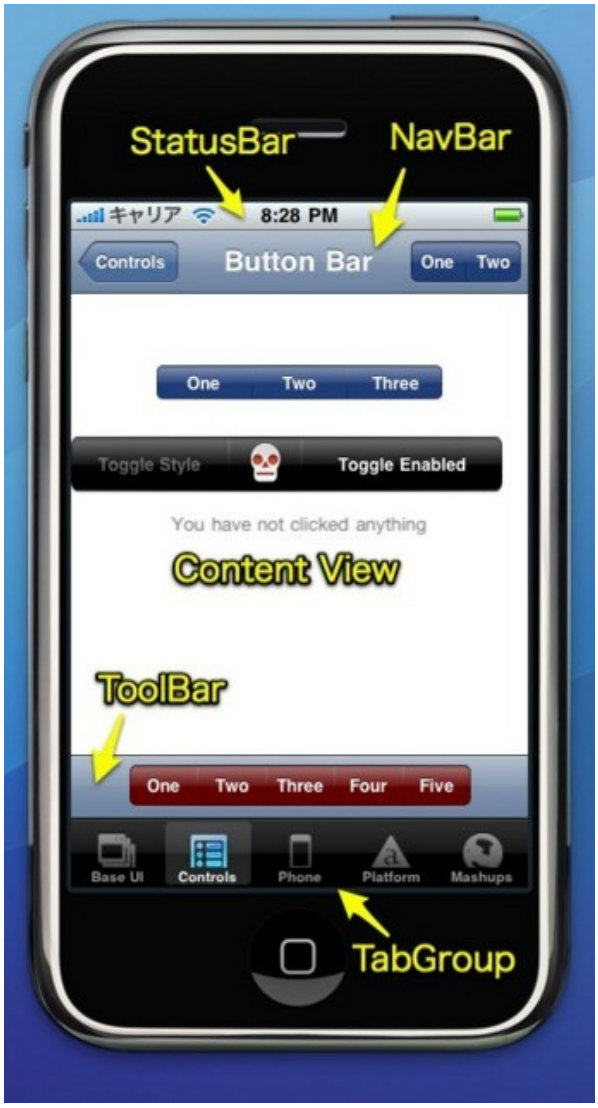
オブジェクトの作成や解放、各種オブジェクトの抽象化(iPhone/Android 間)したところで、実 際のユーザインタフェイスの構造までは抽象化できません。

親亀となるオブジェ クトコンテナ（Window や View）を作って、その上に子亀となるオブジェクト（View や Control）を載せるということをする必要があるわけ です。

そのためにはどういうもので UI が構成されているのかを知る必要があり ます。

部品の名前と役割

それぞれの部品をなんというかというところについては次の画像を見ていただくのが一番早いと思います。



画面を構成する表示領域

Window	すべての UI 部品や View を格納する親オブジェクト
View	コンテンツ表示部の総称。いろいろな種類がある。
StatusBar	デバ イスの表示部最上方にある表示領域
NavBar	Status Bar の直下にある表示領域
ToolBar	画面下部に配置される表示領域
TabGroup	(図では `TabBar`)最下部に配置される特殊な表示領域。 複数の Window を切り替える操作をする 機能を持つ

表示部には次のような階層構造に なってると考えれば（オブジェクトアクセス的にも）いいでしょう。

- Application 全体(tiapp.xml 含む)
    - Window(s)
        - StatusBar
        - NavBar
            - Control(s)
        - View(s)
            - View(s)/Control(s)...
        - ToolBar
            - Control(s)
    - TabBar

代表的な View

WebView	HTML 部品を表示するための View。NativeUI を使わずに HTML+Javascript+CSS でアプリケー ションを開発する場合でもこの View が配置されている寸法になります。
TableView	縦 1 列にデータが並んでいる形式です。Safari のブックマークなどが代表になるでしょうか。「設定」アプリケーション のような表組みもこちらにあたります。
ImageView	画像表示を行うための View。Coverflow のためには CoverFlowView と いう別のものがあります。
MapView	地図表示を行うための View。「マップ」アプリケーションの地図表示部を想像すればいいかと。

代表的な Control

Label	ラベル。文字を表示するだけの部品。
Button	ボタン。いろんなところで出番があります。
TextField	一行入力のテキストボックス。非常にオプションが多いです。
TextArea	複数行入力可能なテキスト入力部品。

View にも Control にもその他大勢の種類があるので UI カタログ内でどういうものがあるのかを見ていこうと思います。

UI カタログ - Window

モード

Window には通常モードとフルスクリーンモードの二つがあります。

フルスクリーンモード下では、StatusBar・NavBar・ToolBar・ TabGroup と いった部品をもつ事ができませんが、デバイスの表示領域一杯を使う事ができます。

一方、通常モードは StatusBar と NavBar が 標準で表示されます。TabGroup を 配置する場合、Window オブジェクト群は TabGroup の 子オブジェクトとして配置されます。

Window の作成

window を生成するために `Titanium.UI.createWindow` という API が用 意されています。

仕様上ひとつの window しか同時に表示できないため、作成された window は window スタックに格納されるだけになります。 現在表示されている window を閉じると、スタックの一つ手前にある window が再び表示されるようになります。

```
// 切り替え時にはアニメーションする。独立した表示
var win = Titanium.UI.createWindow();
win.open({animated:true});
// 現在の Tab に所属させるのなら次のとおり。
Titanium.UI.currentTab.open(win,{animated:true});
```

上記のように `open` メソッドの引数として `animeted` プロパティを指定 する事により、アニメーション制御できます。この例では window が左へスライドしていくような動きをします。`false` 指定時は切 り替わるだけの動きになります。

表示されている Window の 取得と操作

現在表示されている window は `Titanium.UI.currentWindow` プ ロパティを用いてアクセスできます。

たとえば、この window に対して `close` メソッドを実行すると現在表示されている window を閉じることができます。

```
Titanium.UI.currentWindow.close();
```

Window に関する特殊なイベント

window も配下になる view の一環なので view がもつすべてのイベントをハンドルすることが できますが、それとは別に window のイベントとしては次のようなものがあります。

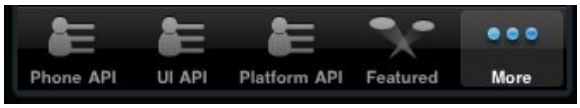
イベント	発生するタイミング
open	開く
close	閉じる
focus	選択状態になる
blur	選択外状態になる

関連する API ドキュメント

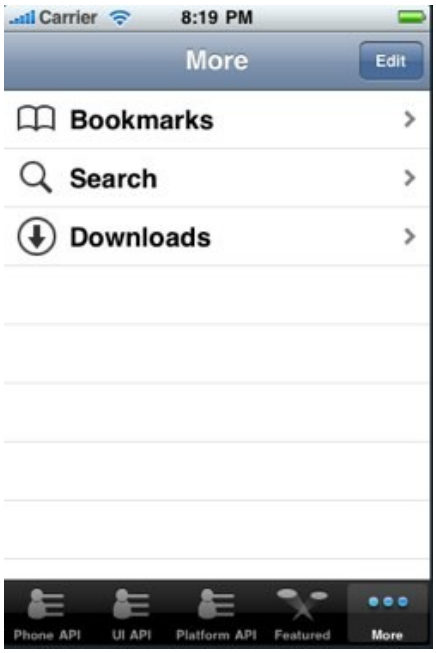
- https://developer.appcelerator.com/apidoc/mobile/latest/Titanium.UI.Window

UI カタログ - TabGroup

TabGroup は 複数の window を束ねるアプリケーション UI の根幹を担います。



タ ブは同時に 5 つまでの表示しかできません。6 つ以上の表示をしようとすると 5 つ目以上は「more」タブの中に入り、そこから選択する動きになります。



TabGroup と Tab の生成

基本的に app.js 上での記述になりますが、次のような形が定石となります。

```
// ただ生成する。
var tabGroup = Titanium.UI.createTabGroup();
// Tab の生成はヒモづく Window とのセットという感じになります。
var win = Ti.UI.createWindow({title:'New Tab Window',barColor:'#000'});
var newtab = Titanium.UI.createTab({
  icon:'../images/tabs/KS_nav_mashup.png',
  title:'New Tab',
  win:win
});
// Tab を TabGroup に追加する
tabGroup.addTab(newtab);
// 表示する
```

```
tabGroup.open();
```

TabGroup と Tab の操作

TabGroup は 次のように取得でき、そこから Tab を操作できます。

```
// TabGroup の取得
var tabGroup = Titanium.UI.currentWindow.tabGroup;
// 所属するタブは tabs プロパティにある
alert(tabGroup.tabs[0].title);
// 強制的に tab 切り替えもできる
tabGroup.setActiveTab(1);
tabGroup.setActiveTab(tabGroup.tabs[2]);
```

バッジ機能

個々のタブには状態を通知するためのするための「タブ バッジ」と呼ばれるものをもつ事ができます。 一応、文字列も表示できるのですが、数値を表示するのに適していますので、そのように使ったほうがよいでしょう。


```
// すべてのタブに対してバッジを付与する。
var tabs = Titanium.UI.currentWindow.tabGroup.tabs;
tabs[0].setBadge(1);
tabs[1].setBadge(2);
tabs[2].setBadge(3);
```

関連する API ドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.TabGroup
- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.Tab

UI カタログ - StatusBar (iPhone のみ)

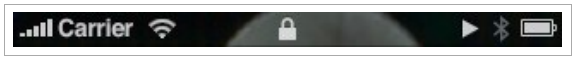
StatusBar は ３種類の設定を切り替えることができます。

また、表示非表示も切り替える事が出来ます。

```
// これは TRANSLUCENT_BLACK 指定時
Titanium.UI.iPhone.setStatusBarStyle(Titanium.UI.iPhone.StatusBar.TRANSLUCENT_BLACK);

// ステータスバーを消す
Titanium.UI.iPhone.hideStatusBar();
// 再表示する
Titanium.UI.iPhone.showStatusBar();
```

表示例

Titanium.UI.iPhone.StatusBar.GREY	

Titanium.UI.iPhone.StatusBar.TRANSLUCENT_BLACK	
Titanium.UI.iPhone.StatusBar.OPAQUE_BLACK	

関連する APIドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.iPhone
- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.iPhone.StatusBar

UI カタログ - NavBar

Window の StatusBar の 直下にあり、画面遷移をしていく画面でのナビゲーション・指示を司る UI 部品です。

表 示・非表示の切り替え

```
// 現在の window から表示→非表示
Titanium.UI.currentWindow.hideNavBar();
// 現在の window から非表示→表示
Titanium.UI.currentWindow.showNavBar();
```

背景色の変更

Window 生成時に指定する方法と動的な変更 の二種類があります。

```
// window 生成時に指定する。
var win = Titanium.UI.createWindow({barColor:'#336699'});
// 現在表示中の Window の NavBar の背景色を変更する。
Titanium.UI.currentWindow.barColor = '#336699';
// デフォルト色に戻す。
Titanium.UI.currentWindow.barColor = null;
```

表示テキストの変更

タイトル部分・プロンプトを 表示する場合は次のようにします。

```
// タイトル変更
Titanium.UI.currentWindow.title = 'タイトル';
// プロンプトの表示(再び無効にする場合は null を設定する)
Titanium.UI.currentWindow.titlePrompt = 'プロンプトが表示されます。';
```

また、戻るボタンの表示内容も同様に変更できます。

```
Titanium.UI.currentWindow.backButtonTitle = '戻る！';
```

タイトル部への画像表示

タイトルのかわ りに画像を表示する事も可能です。

```
// 画像を表示する(消す場合はnullを設定)
Titanium.UI.currentWindow.titleImage = '../images/slider_thumb.png';
// 戻るボタンも画像化することが可能です
Titanium.UI.currentWindow.backButtonTitleImage = null;
```

コントロールの配置

NavBarも コントロールコンテナなので、各種コントロールを載せる事ができます。

ここではButtonを 配置していますが、Switchや Slider、TabbedBarや TextFieldと いったものまで配置できます。

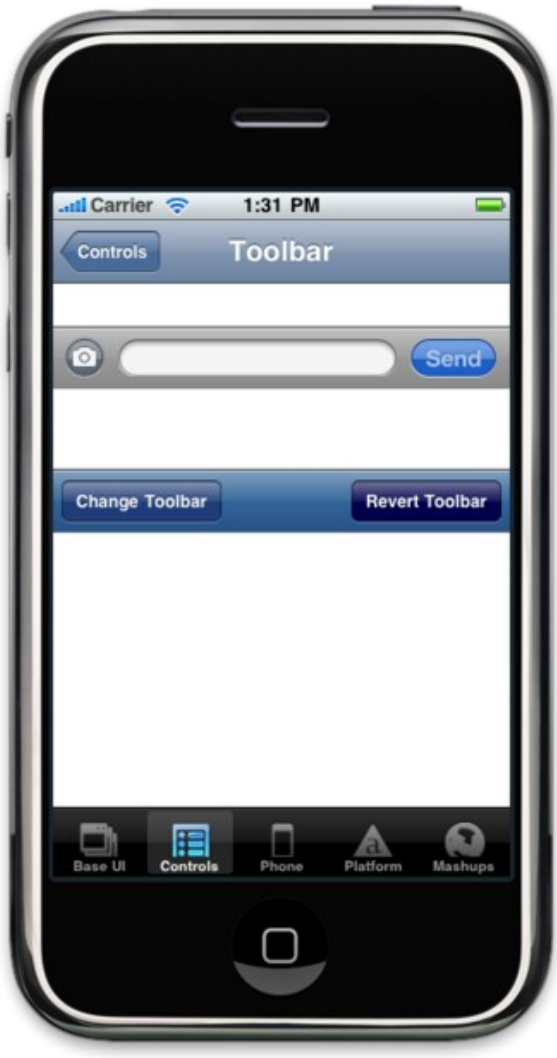
```
var b1 = Titanium.UI.createButton({title:'Left Nav'});
var b2 = Titanium.UI.createButton({title:'Title'});
var b3 = Titanium.UI.createButton({title:'Right Nav'});
// タイトル部にボタン配置
Titanium.UI.currentWindow.titleControl = b2;
// 左右のボタンにも配置可能です
Titanium.UI.currentWindow.leftNavButton = b1;
Titanium.UI.currentWindow.setLeftNavButton(null);
Titanium.UI.currentWindow.rightNavButton = b3;
Titanium.UI.currentWindow.setRightNavButton(null);
```

関連するAPIドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.Window

UIカタログ - ToolBar

ToolBarは （主に画面下部に配置する）ボタンなどのコントロールを収納するコンテナです。




```
// ツールバーに載せる部品群を定義します。
var flexSpace = Titanium.UI.createButton({
    systemButton:Titanium.UI.iPhone.SystemButton.FLEXIBLE_SPACE
});
var tf = Titanium.UI.createTextField({
    height:32,
    backgroundImage:'../images/inputfield.png',
    width:200,
    font:{fontSize:13},
    color:'#777',
    paddingLeft:10,
    borderStyle:Titanium.UI.INPUT_BORDERSTYLE_NONE
});
var camera = Titanium.UI.createButton({
    backgroundImage:'../images/camera.png',
    height:33,
    width:33
})
camera.addEventListener('click', function(){
    Titanium.UI.createAlertDialog({title:'Toolbar',message:'You clicked
camera!'}).show();
});
```

```
var send = Titanium.UI.createButton({
    backgroundImage:'../images/send.png',
    backgroundSelectedImage:'../images/send_selected.png',
    width:67,
    height:32,
});
send.addEventListener('click', function(){
    Titanium.UI.createAlertDialog({title:'Toolbar',message:'You clicked send!'}).show();
});
// ツールバーを新たにつくり、上記で定義したコントロールを並べます。
var toolbar1 = Titanium.UI.createToolbar({
    items:[flexSpace,camera, flexSpace,tf,flexSpace, send,flexSpace],
    top:30,
    borderTop:true,
    borderBottom:false,
    translucent:true,
    barColor:'#999',
});
Titanium.UI.currentWindow.add(toolbar1);
```

関連するAPIドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.Toolbar

UI カタログ(API) - ダイアログ関連

シンプルなアラート(Alert Dialog)

タイトルとメッセージ、OK ボタンだけというようなダイ アログのパターンは次のように記述します。

```
var dialog = Titanium.UI.createAlertDialog();
dialog.setTitle('アラートのテスト');
dialog.setMessage('メッセージはここに指定します。');
dialog.show();
```

このような引数を引き渡すことにより、ボタンを増やしたりキャンセルボタンの認識が可 能になる生成方法もあります。

```
var alertDialog = Titanium.UI.createAlertDialog({
  title: 'キャンセルのテスト',
  message: 'テスト',
  buttonNames: ['OK','きゃんせる'],
  // キャンセルボタンがある場合、何番目(0 オリジン)のボタンなのかを指定できます。
  cancel: 1
});
```

```
alertDialog.addEventListener('click',function(event){
  // Cancel ボタンが押されたかどうか
  if(event.cancel){
    // cancel 時の処理
  }
  // 選択されたボタンの index も返る
  if(event.index == 0){
    // "OK"時の処理
  }
});
alertDialog.show();
```

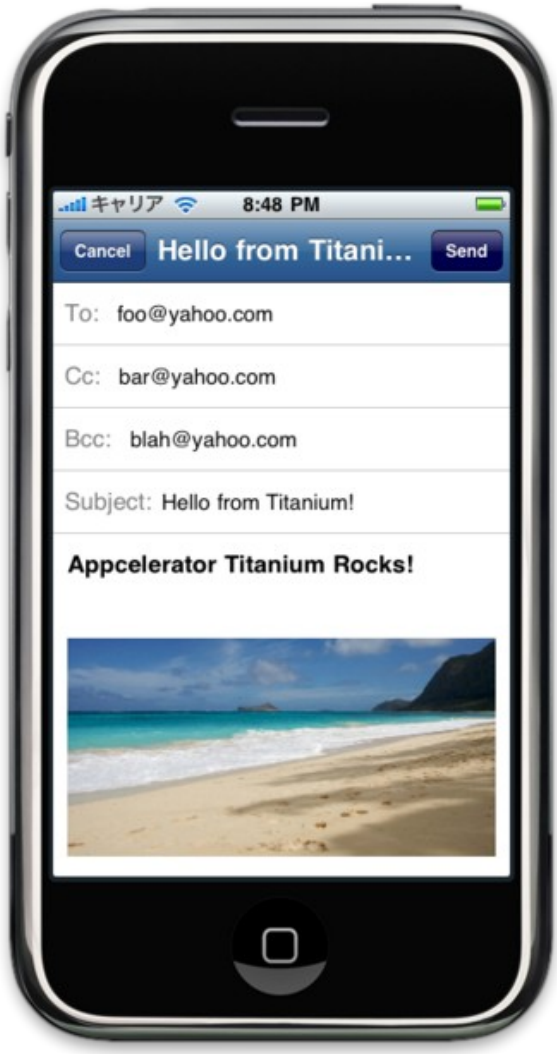
処理選択をするダイアログ(Option Dialog)

プロンプトと共に処理内容を選択させるような画 面です。

```
// ダイアログの生成
var dialog = Titanium.UI.createOptionDialog();
// タイトルということになっていますが、プロンプト的な位置づけですね。
dialog.setTitle('どの処理を実行しますか？');
```

```
// ボタンの配置（ちなみに配列なので 0 オリジンで index を持ちます）
dialog.setOptions(["更新","削除","キャンセル"]);
// 削除などの破壊的な挙動をするボタンは赤くするという規定が
// iPhone にはあるのでそれに該当するボタンの index を指定します。
dialog.setDestructive(1);
// キャンセルボタンにも同様の規定があるので、index を指定します。
dialog.setCancel(2);
// ボタン選択時の処理はイベントハンドラを記述します。
// 第一引数の index プロパティで選択されたボタンの index が設定されます。
dialog.addEventListener('click',function(event){
  if(event.index == 0){
    // 更新処理
  }
  else if(event.index == 1){
    // 削除処理(event.desctructive == true でも可能)
  }
  // キャンセル時は event.cancel == true となる
});
// ダイアログを表示します。
dialog.show();
```

E-mail 作成ダイアログ



いわゆる e-mail の作成画面です。

ぼくは MMS に対応してからめっきり使わなくなりましたが(^^;

```
var emailDialog = Titanium.UI.createEmailDialog()
// 題名の初期値をセットします
emailDialog.setSubject('題名');
// To, Cc, Bcc については文字列配列として引き渡します。
emailDialog.setToRecipients(['foo1@yahoo.com', 'foo2@yahoo.com']);
emailDialog.setCcRecipients(['bar@yahoo.com']);
emailDialog.setBccRecipients(['hoge@yahoo.com']);
// 本文と添付(ここではすでに image というオブジェクトがある前提)を初期設定します。
emailDialog.setMessageBody('this is a test message');
emailDialog.addAttachment(image);
// ツールバー色を指定して画面を開きます。
emailDialog.setBarColor('#336699');
emailDialog.open();
```

関連する API ドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.AlertDialog
- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.OptionDialog
- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.EmailDialog

UI カタログ - View 共通

Titanium Mobile の Window には View、もしくは Control を格納できます。

複雑な UI 制御をする際には Window 上に直接 Control を配置するのではなく、View の上に 配置していくほうが制御しやすいです。

View の追加と表示

生成された View は必ず Window もしくは他の View に追加しなければ利用できません。

```
// 新たに作られた myView を Window に追加する
Titanium.UI.currentWindow.add(myView);
```

最後に追加された View が Window 上では初期表示されますが、他の追加しておいた View に切り替えたり、最初から `visible: false` として追加した View を表示させるには `window#animate` を実行 する事により、表示させる事が出来ます。

その際に transition プロパティを 指定する事により、すでに用意されているアニメーションを簡単に実行できます。

次の 例では左へフリップするアニメーションで表示します。

```
// myView はすでにあるものとする
Titanium.UI.currentWindow.animate({
  view: myView,
  transition: Ti.UI.iPhone.AnimationStyle.FLIP_FROM_LEFT
});
```

Transition アニメーションスタイルには次の 4 つがあります。

- `Titanium.UI.iPhone.AnimationStyle.CURL_UP`
- `Titanium.UI.iPhone.AnimationStyle.CURL_DOWN`
- `Titanium.UI.iPhone.AnimationStyle.FLIP_FROM_LEFT`
- `Titanium.UI.iPhone.AnimationStyle.FLIP_FROM_RIGHT`

View のイベント

View 系のオブジェクトには共通して以下のようなイベントをハンドリングできるようになっています。

```
// タッチ開始
view.addEventListener('touchstart', function(e){
  // e.x, e.y：座標
});
// タッチしながら移動
view.addEventListener('touchmove', function(e){
  // e.x, e.y：座標
});
```

```
// タッチ終了
view.addEventListener('touchend', function(e){
  // e.x, e.y：座標
});
// タッチ中止
view.addEventListener('touchcancel', function(e){
  // e.x, e.y：座標
});
// シングルタップ
view.addEventListener('singletap', function(e){
  // e.x, e.y：座標
});
// ダブルタップ
view.addEventListener('doubletap', function(e){
  // e.x, e.y：座標
});
// 二本指でのシングルタップ
view.addEventListener('twofingertap', function(e){
  // e.x, e.y：座標
});
// スワイプ
view.addEventListener('swipe', function(e){
  // e.x, e.y：座標
  // e.direction：スワイプの向き(left | right)
});
// クリック
view.addEventListener('click', function(e){
  // e.x, e.y：座標
});
// ダブルクリック
view.addEventListener('dblclick', function(e){
  // e.x, e.y：座標
});
```

関連する APIドキュメント

https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.View

UI カタログ - WebView

WebView は HTML ベースでのコンテンツを表示するための領域です。

safari で表示できる内容はすべて表示できると考えて問題ありません。

ま た直接 HTML 文字列を引き渡す事も可能ですので、動的に生成した HTML によるリッチな表現をすることができます。


```
// 単純なURLのロード
var webview = Ti.UI.createWebView();
// こういったイベントの取得も可能です
webview.addEventListener('load',function(e){
   Ti.API.debug("webview loaded: "+e.url);
});
webview.url = "http://www.google.co.jp/";

// HTMLを動的に作成してhtmlプロパティにセットする例
var webview = Ti.UI.createWebView({
  backgroundColor: '#fff',
  borderRadius: 15,
  borderWidth : 5,
  borderColor : 'red'
});
webview.html = '<html><body><div style="color:white;">Hello from inline HTML. You should see white text and black background</div></body></html>';
```

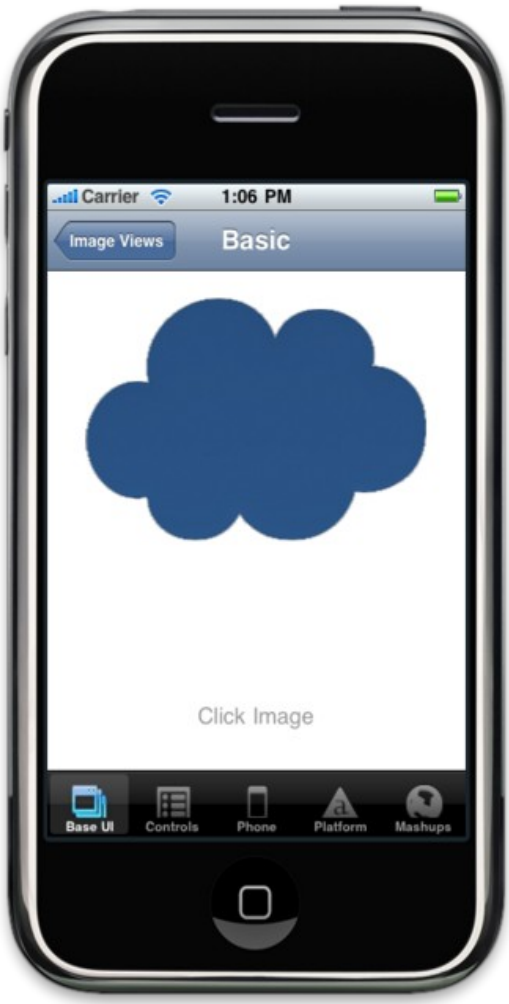
関連するAPIドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.WebView

UI カタログ - ImageView

画像表示が可能な View です。

画像のアニメーションもさせることができます。




```
// ローカルにある画像を指定する場合。
//  (http://〜で始まるパスを入れるとリモートから取得されます)
var imageView = Titanium.UI.createImageView({
  url:'../images/cloud.png',
  width:261,
  height:178,
  top:20
});
Ti.UI.currentWindow.add(imageView);
```

アニメーション

images プロパティを設定する事によ り、パラパラマンガの要領でアニメーションされます。

```
var animationFrames = [
  '../images/frame01.jpg',
  '../images/frame02.jpg',
```

```
  '../images/frame03.jpg',
  '../images/frame04.jpg'
];
var animationView =  Titanium.UI.createImageView({
  height: 200,
  width: 200,
  top: 30,
  images: animationFrames,
  // ミリ秒単位で次のフレームまでの間隔を指定します
  duration: 100,
  // 繰り返し回数を指定します（0 の場合は無限に繰り返します）
  repeatCount: 0
});
Ti.UI.currentWindow.add(animationView);
// パラパラマンガを開始する
animationView.start();
```

関連する API ドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.ImageView

UI カタログ - CoverFlowView

iPod や iTunes でおなじみの処理である画像を左右にスクロールさせながら拡大表示するカバーフローですが、この View を利用することにより、画像の配列を渡すだけで簡単に実現できます。


```
// 表示対象の画像は配列として渡します
var images = [
  '../images/imageview/01.jpg',
  '../images/imageview/02.jpg',
  '../images/imageview/03.jpg',
  '../images/imageview/04.jpg',
  '../images/imageview/05.jpg'
];
// 背景色とセットで画像一覧を引き渡します
var view = Titanium.UI.createCoverFlowView({
  images: images,
  backgroundColor: '#000'
});
// 画像選択時のイベント
view.addEventListener('click',function(e){
  Titanium.API.info("image clicked: "+e.index+', selected is '+view.selected);
});
// フリックなどで選択中の画像が変わったときのイベント
view.addEventListener('change',function(e){
  Titanium.API.info("image changed: "+e.index+', selected is '+view.selected);
});
Ti.UI.currentWindow.add(view);
```

関連する API ドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.CoverFlowView

UI カタログ - MapView

地図表示を行う View です。

単体では下の例のようなツールバーはついていません。


```
// マーカーは Annotation オブジェクトとして表現される。
var atlanta = Titanium.Map.createAnnotation({
    latitude:33.74511,
    longitude:-84.38993,
    title:"Atlanta, GA",
    subtitle:'Atlanta Braves Stadium',
    pincolor:Titanium.Map.ANNOTATION_PURPLE,
    animate:true,
    leftButton:'images/atlanta.jpg',
    rightButton: Titanium.UI.iPhone.SystemButton.DISCLOSURE,
```

```
    myid:3 // CUSTOM ATTRIBUTE THAT IS PASSED INTO EVENT OBJECTS
});
// MapViewオブジェクトを作成する。
var mapview = Titanium.Map.createView({
    mapType: Titanium.Map.STANDARD_TYPE,
    region: {latitude:33.74511, longitude:-84.38993, latitudeDelta:0.01,
longitudeDelta:0.01},
    animate:true,
    regionFit:true,
    userLocation:true,
    annotations:[atlanta]
});
Ti.UI.currentWindow.add(mapview);
```

地図表示形式

地図の表示方法には 3 種類用意されています。

- Titanium.Map.STANDARD_TYPE
- Titanium.Map.SATELLITE_TYPE
- Titanium.Map.HYBRID_TYPE

ピンの色

ピンの色は次の 3 つが用意されています。

- Titanium.Map.ANNOTATION_GREEN
- Titanium.Map.ANNOTATION_PURPLE
- Titanium.Map.ANNOTATION_RED

拡大縮小

zoom メソッドがあり、引数に縮尺の変化を指示するかたち となっています。

```
// zoom in
mapview.zoom(1);
// zoom out
mapview.zoom(-1);
```

関連する API ドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.Map
- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.Map.MapView
- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.Map.Annotation

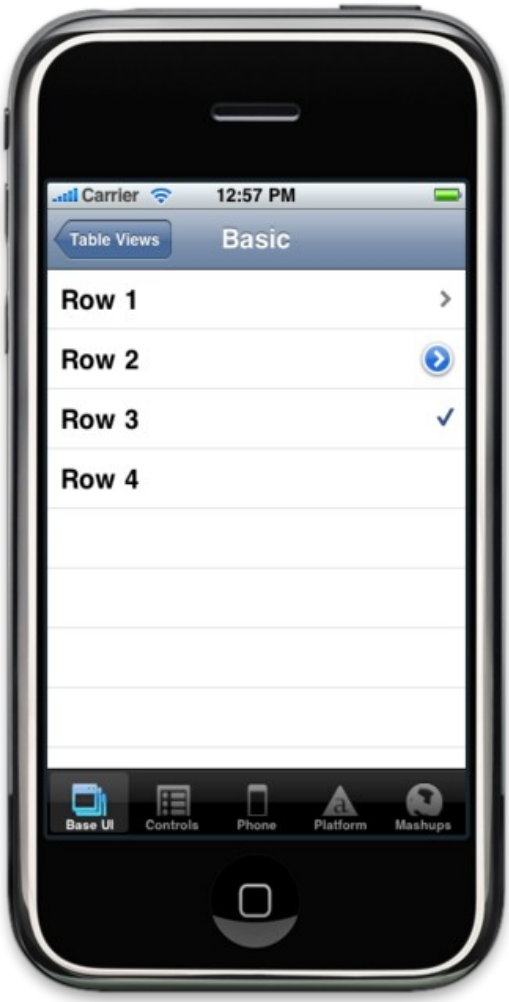
UI カタログ - TableView (基本編)

TableView は 単純な行選択から複雑なレイアウト、グループ表示など多岐に渡る奥の深いものです。

ここではまず基本的な部分について見ていきます。

標準的な Table View

標準的な TableView は 次のような単純な行選択をするための UI になります。

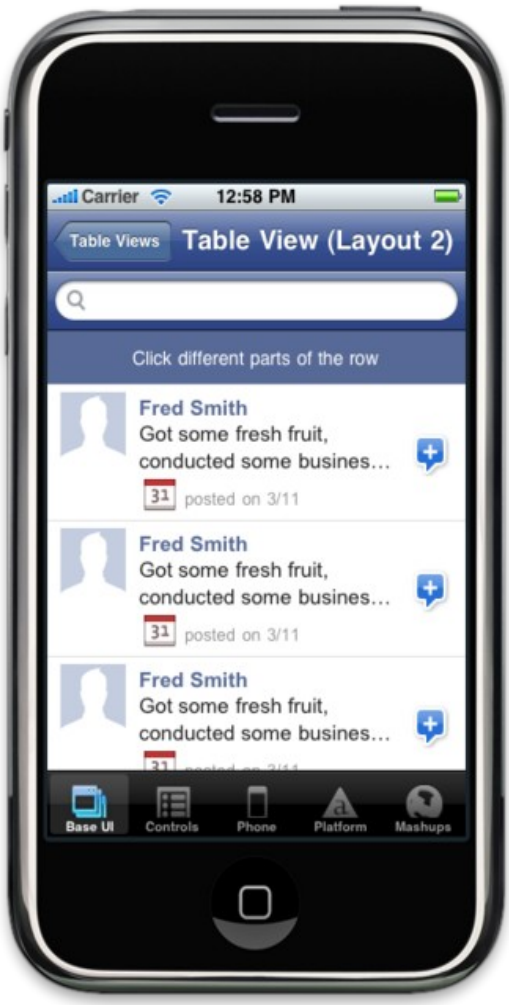



```
// Table Viewに表示するデータを作成しておきます
// hasChild, hasDetail, hasCheckプロパティがあると上の例のような表示になります。
var rows = [
  {title:'Row 1', hasChild:true},
  {title:'Row 2', hasDetail:true},
  {title:'Row 3', hasCheck:true},
  {title:'Row 4'}
];
// 先ほどのデータに基づいてTable Viewを起こします。
var tableview = Titanium.UI.createTableView({
  data: rows
});
// イベントリスナにクリック時のイベントを登録します。
tableview.addEventListener('click', function(e){
  var index     = e.index;
  var section  = e.section;
  var row       = e.row;
  var rowdata = e.rowData;
});
```

```
// Windowに追加する
Titanium.UI.currentWindow.add(tableview);
```

レイアウト

このように各行に対するレイアウトデザインすることも 可能です。




```
// Navbarの色を変えつつ...
var win = Titanium.UI.currentWindow;
win.barColor = '#385292';
// 検索バーも配置しておきます
var searchBar = Titanium.UI.createSearchBar({
  barColor:'#385292',
  showCancel:false
});
// 格納データとともに宣言
var tableView;
var rowData = [];
// ヘッダ部
var headerRow = Ti.UI.createTableViewRow();
headerRow.backgroundColor = '#576996';
headerRow.selectedBackgroundColor = '#385292';
headerRow.height = 40;
var clickLabel = Titanium.UI.createLabel({
```

```
    text:'Click different parts of the row',
    color:'#fff',
    textAlign:'center',
    font:{fontSize:14},
    width:'auto',
    height:'auto'
});
headerRow.className = 'header';
headerRow.add(clickLabel);
rowData.push(headerRow);
// レイアウト行
for (var c = 1; c < 50; c++){
  // datarow クラスとして TableViewRow を作成
  var row = Ti.UI.createTableViewRow();
  row.selectedBackgroundColor = '#fff';
  row.height  =100;
  row.className = 'datarow';
  // 画像を配置する
  var photo = Ti.UI.createView({
    backgroundImage:'../images/custom_tableview/user.png',
    top:5,
    left:10,
    width:50,
    height:50
  });
  photo.rowNum = c;
  photo.addEventListener('click', function(e){
    // 上でセットした rowNum にあたる e.source.rowNum でデータを特定する
  });
  row.add(photo);
  // ラベルを配置する
  var user = Ti.UI.createLabel({
    color:'#576996',
    font:{fontSize:16,fontWeight:'bold', fontFamily:'Arial'},
    left:70,
    top:2,
    height:30,
    width:200,
    text:'Fred Smith '+c
  });
  user.rowNum = c;
  user.addEventListener('click', function(e){
    // 上でセットした rowNum にあたる e.source.rowNum でデータを特定する
  });
  row.add(user);
  // ラベル２個目
  var comment = Ti.UI.createLabel({
    color:'#222',
    font:{fontSize:16,fontWeight:'normal', fontFamily:'Arial'},
    left:70,
    top:21,
    height:50,
```

```
    width:200,
    text:'Got some fresh fruit, conducted some business, took a nap'
  });
  row.add(comment);
  // View も配置できる
  var calendar = Ti.UI.createView({
    backgroundImage:'../images/custom_tableview/eventsButton.png',
    bottom:2,
    left:70,
    width:32,
    height:32
  });
  calendar.rowNum = c;
  calendar.addEventListener('click', function(e){
    // 上でセットした rowNum にあたる e.source.rowNum でデータを特定する
  });
  row.add(calendar);
  // ボタンを配置する
  var button = Ti.UI.createView({
         backgroundImage:'../images/custom_tableview/commentButton.png',
         top:35,
         right:5,
         width:36,
         height:34
  });
  button.rowNum = c;
  button.addEventListener('click', function(e){
    // 上でセットした rowNum にあたる e.source.rowNum でデータを特定する
  });
  row.add(button);
  // ラベルは省略
  var date = Ti.UI.createLabel({
    // (...)
    text:'posted on 3/11'
  });
  row.add(date);
  // 以上の内容の行を追加する
  rowData.push(row);
}
tableView = Titanium.UI.createTableView({
  data:    rowData,
  search: searchBar
});
```

設定

また「設定」画面のようなグループ表示も TableView の 一環としてサポートされていますので、作法も TableView に 則る必要があります。

グループ表示をするには TableViewSection?と TableViewStyle?の 指定を用います。

```
// 外枠となる Table View Section を生成する。
var groupData = Ti.UI.createTableViewSection({
  headerTitle: 'Group 1'
});
// ここではダミーデータを追加する...
for (var x = 0; x < 10; x++){
  var row = Ti.UI.createTableViewRow({
    title: 'Group 1, Row ' + (x + 1)
  });
  // TableViewSection に TableViewRow を追加する
  groupData.add(row);
}
// テーブルビューの style を指定する。
var tableview = Titanium.UI.createTableView({
  data: gruopData,
  style: Titanium.UI.iPhone.TableViewStyle.GROUPED
});
```

ま たコントロールの配置ももちろん出来ます。

```
// 格納する行データ配列を用意する
var rowData = [];
```

```
// １つ目のスイッチとボタン
var row1 = Ti.UI.createTableViewRow({height:50});
var sw1  = Ti.UI.createSwitch({
  right:10,
  value:false
});
row1.add(sw1);
var button1 = Ti.UI.createButton({
  style:Titanium.UI.iPhone.SystemButton.DISCLOSURE,
  left:10
});
row1.add(button1);
row1.className = 'control';
rowData.push(row1);
// ２つ目は省略
// 先ほど同様に GROUPEDstyle を指定する
var tableView = Ti.UI.createTableView({
  data: rowData,
  style: Titanium.UI.iPhone.TableViewStyle.GROUPED,
  top:50
});
```

関連する APIドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.TableView
- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.TableViewRow
- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.TableViewSection

UI カタログ - ScrollView

スクロール可能な View です。

スワイプやボタンによるアクションでスクロールさせることが可能になります。


```
var view1 = Ti.UI.createView({
  backgroundColor: 'red'
});
var view2 = Ti.UI.createView({
  backgroundColor: 'blue'
});
var view3 = Ti.UI.createView({
  backgroundColor: 'green'
});
var view4 = Ti.UI.createView({
  backgroundColor: 'black'
});
// 上記の view を配列として views プロパティに引き渡します。
var scrollView = Titanium.UI.createScrollableView({
  views: [view1,view2,view3,view4],
  showPagingControl: true,
  pagingControlHeight: 30,
  maxZoomScale: 2.0
});
// スクロールビューを配置する
Titanium.UI.currentWindow.add(scrollView);
```

関連する API ドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.ScrollView
- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.ScrollableView

UI カタログ(API) - Virtical Layout

指定した Window や View の中に配置したコントロールや View が垂直に自動配置されていくレイアウト指定です。

次の例では左が Window と中心部の VIew, 右が Table View 内の Row で Virtical Layout を指定したものになります。


```
// Window の layout プロパティに 'virtical' を指定する。
var win = Ti.UI.currentWindow;
win.layout = 'vertical';
// ヘッダ部となる View を設定する。（べつにヘッダというものが Virtical Layout にあるわけではない）
var header = Ti.UI.createView({
  height:50,
  borderWidth:1,
  borderColor:'#999'
});
```

```
var headerLabel = Ti.UI.createLabel({
  color:'#777',
  top:10,
  textAlign:'center',
  height:'auto', text:'Header'
});
header.add(headerLabel);
win.add(header);
// ボディ部となるViewを設定する。（同様にボディ部があるわけでもない）
// このボディ自体もVirtical Layoutするようにする(局所的なVirtical Layout)
var body = Ti.UI.createView({
  height:'auto',
  layout:'vertical'
});
var bodyView1 = Ti.UI.createView({backgroundColor:'#336699', height:100, left:10,
right:10});
var bodyView2 = Ti.UI.createView({backgroundColor:'#ff0000', height:50, left:10,
right:10, top:10});
var bodyView3 = Ti.UI.createView({backgroundColor:'orange', height:50, left:10,
right:10, top:10});
body.add(bodyView1);
body.add(bodyView2);
body.add(bodyView3);
win.add(body)
// 同様に「フッタ」を作る
var footer = Ti.UI.createView({
  height:50,
  borderWidth:1,
  borderColor:'#999'
});
var footerLabel = Ti.UI.createLabel({color:'#777', textAlign:'center', height:'auto',
text:'Footer'});
footer.add(footerLabel);
win.add(footer);
```

UIカタログ(コントロール) - Label

文字を表示するコントロールです。


```
var win = Titanium.UI.currentWindow;
var l1 = Titanium.UI.createLabel({
    text:'Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis
nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat',
    width:200,
    height:150,
    top:10,
    color:'#336699',
    textAlign:'center'
});
win.add(l1);
var l2 = Titanium.UI.createLabel({
    text:'Appcelerator',
    height:50,
    width:'auto',
    shadowColor:'#aaa',
    shadowOffset:{x:5,y:5},
    color:'#900',
    font:{fontSize:48},
    top:170,
    textAlign:'center'
});
```

```
win.add(l2);
// 中略
var b2 = Titanium.UI.createButton({
    title:'Change Label 2',
    height:40,
    width:200,
    top:280
});
b2.addEventListener('click', function(){
    l2.color = '#ff9900';
    l2.shadowColor = '#336699';
    l2.font = {fontSize:20};
});
win.add(b2);
```

関連する APIドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.Label

UI カタログ(コントロール) - Button

ボタンは次の４つのいずれかで表現されます。

- テキストのみ
- 画像のみ
- シ ステムボタン
- 画像とテキストの組み合わせ

```
// 背景画像としてblue.png, blue_pressed.pngが用意されているものとします。
var button1 = Titanium.UI.createButton({
 backgroundImage:'blue.png',
 backgroundSelectedImage:'blue_pressed.png',
 title:'Hello',
 color:'#ff0000',
 height:47,
 top:30
});
button1.addEventListener('click', function(e) {
 // ボタンクリック時のイベント
});
// こちらはシンプルなボタンです。
var button2 = Titanium.UI.createButton({
 title:'Hello',
 color:'#ff0000',
 height:30,
 width:100,
 top:100
});
button2.addEventListener('click', function(e){
  // ...
});
```

上 記のコード例で次のようなボタンイメージとなります。

システムボタンとボタン形状

- シ ステムボタンアイコンの使い方
- ボ タン形状の指定

スペイサー

コントロール間の距離を調整するためには、ボタンを特殊な形式に指定する必要があります。

可変幅のスペイサー

ツールバーなどで左右端や中央 にボタンを配置したい場合があると思います。 そういった場合には可変サイズの指定をした特殊なボタンを配置することで対処できます。

```
// 中央にボタンを配置するパターン
var flexSpace = Titanium.UI.createButton({
  systemButton:Titanium.UI.iPhone.SystemButton.FLEXIBLE_SPACE
});
var button = Titanium.UI.createButton({title: 'ど真ん中'});
Titanium.UI.currentWindow.setToolbar([flexSpace,button,flexSpace]);

// やや右寄りに配置する
var flexSpace = Titanium.UI.createButton({
  systemButton:Titanium.UI.iPhone.SystemButton.FLEXIBLE_SPACE,
});
var button = Titanium.UI.createButton({title:'Text'});
Titanium.UI.currentWindow.setToolbar([flexSpace,flexSpace,button,flexSpace]);
```

固定幅スペイサー

両端から少し間を持たせたいとき、 隣のボタンとの間を取りたいときなどに固定幅の特殊なボタンを配置することで解決します。

```
var fixedSpace = Titanium.UI.createButton({
  systemButton:Titanium.UI.iPhone.SystemButton.FIXED_SPACE,
  width: '40',
});
var button = Titanium.UI.createButton({title:'Text'});
Titanium.UI.currentWindow.setToolbar([fixedSpace,button]);
```

関連する API ドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.Button

システムボタンアイコンの使い方

iPhoneOS に組み込まれているボタンアイコンを使う事ができます。 むしろ、そういった挙動（たとえばカメラ撮影をするなど）をする場合はこれらのアイコンを使う事が推奨されていますので、機能が合致する場合は積極的に 使ったほうがよいでしょう。

ボタンのインスタンス作成時の引数として渡し て使います。

```
var button = Titanium.UI.createButton({
  systemButton:Titanium.UI.iPhone.SystemButton.CAMERA
});
```

ボタン一覧

- Titanium.UI.iPhone.SystemButton.ACTION
- Titanium.UI.iPhone.SystemButton.CAMERA
- Titanium.UI.iPhone.SystemButton.COMPOSE
- Titanium.UI.iPhone.SystemButton.BOOKMARKS
- Titanium.UI.iPhone.SystemButton.SEARCH
- Titanium.UI.iPhone.SystemButton.ADD
- Titanium.UI.iPhone.SystemButton.TRASH
- Titanium.UI.iPhone.SystemButton.REPLY
- Titanium.UI.iPhone.SystemButton.STOP
- Titanium.UI.iPhone.SystemButton.REFRESH
- Titanium.UI.iPhone.SystemButton.PLAY
- Titanium.UI.iPhone.SystemButton.PAUSE
- Titanium.UI.iPhone.SystemButton.FAST_FORWARD
- Titanium.UI.iPhone.SystemButton.REWIND
- Titanium.UI.iPhone.SystemButton.EDIT
- Titanium.UI.iPhone.SystemButton.CANCEL
- Titanium.UI.iPhone.SystemButton.SAVE
- Titanium.UI.iPhone.SystemButton.ORGANIZE
- Titanium.UI.iPhone.SystemButton.DONE
- Titanium.UI.iPhone.SystemButton.DISCLOSURE
- Titanium.UI.iPhone.SystemButton.CONTACT_ADD
- Titanium.UI.iPhone.SystemButton.SPINNER
- Titanium.UI.iPhone.SystemButton.INFO_DARK
- Titanium.UI.iPhone.SystemButton.INFO_LIGHT

関連する API ドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.iPhone.SystemIcon

ボタン形状の指定

iPhone の Native UI のボタンには 3 種類の形状が用意されています。 使うにはボタン生成時に以下のように引数として指定する必要があります。

```
var button = Titanium.UI.createButton({
 title:'Text',
 style:Titanium.UI.iPhone.SystemButtonStyle.DONE
});
```

引数の種類としては以下の３つから選びます。

- Titanium.UI.iPhone.SystemButtonStyle.PLAIN
- Titanium.UI.iPhone.SystemButtonStyle.BORDERED
- Titanium.UI.iPhone.SystemButtonStyle.DONE

関連する API ドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.iPhone.SystemButtonStyle

UI カタログ(コントロール) - TextField

言わずもがなのテキスト入力を行うためのコントロールです。

キーボードの制御設定やキーボードツールバーを設定できます。


```
var tf1 = Titanium.UI.createTextField({
    color:'#336699',
    height:35,
    top:10,
    left:10,
    width:250,
    height:40,
    hintText:'hintText',
    keyboardType:Titanium.UI.KEYBOARD_DEFAULT,
    returnKeyType:Titanium.UI.RETURNKEY_DEFAULT,
    borderStyle:Titanium.UI.INPUT_BORDERSTYLE_ROUNDED,
});
// TEXT FIELD EVENTS (return, focus, blur, change)
tf1.addEventListener('return',function(e){
    l.text = 'return received, val = ' + e.value;
    tf1.blur();
});
tf1.addEventListener('focus',function(e){
    l.text = 'focus received, val = ' + e.value;
});
tf1.addEventListener('blur',function(e){
```

```
    l.text = 'blur received, val = ' + e.value;
});
tf1.addEventListener('change', function(e){
    l.text = 'change received, event val = ' + e.value + '\nfield val = ' + tf1.value;

});
```

キーボードの種類

Titanium.UI.KEYBOARD_ASCII

Titanium.UI.KEYBOARD_URL

Titanium.UI.KEYBOARD_PHONE_PAD

Titanium.UI.KEYBOARD_NUMBERS_PUNCTUATION

Titanium.UI.KEYBOARD_NUMBER_PAD

Titanium.UI.KEYBOARD_EMAIL_ADDRESS

Titanium.UI.KEYBOARD_DEFAULT

Enter キーの種類 (returnKeyType)

組み込まれている確定キーのボタン種別としては以下のものがあります。

なお、任意のテキストは指定できません。

- Titanium.UI.RETURNKEY_GO
- Titanium.UI.RETURNKEY_JOIN
- Titanium.UI.RETURNKEY_NEXT
- Titanium.UI.RETURNKEY_SEARCH
- Titanium.UI.RETURNKEY_SEND
- Titanium.UI.RETURNKEY_DONE
- Titanium.UI.RETURNKEY_DEFAULT
- Titanium.UI.RETURNKEY_ROUTE
- Titanium.UI.RETURNKEY_YAHOO
- Titanium.UI.RETURNKEY_GOOGLE
- Titanium.UI.RETURNKEY_EMERGENCY_CALL

その他の動作や見え方の指定

Autocorrection

日本語入力しているとあまり使う局面がありませんので、false で指定しておいたほうが便利かもしれま せん。

テキストの表示位置

テキストの表示位置は textAlign プロパティに以下のいずれかの値を指定することで設定できます。

left	左寄せ
center	中央寄せ
right	右寄せ

縦位置については verticalAlign プロパティに次の値を指 定します。

top	上寄せ
middle	中央寄せ
bottom	下寄せ

初期値の設定

初 期値をあらかじめ表示しておく場合、value プロパティにセットします。

入力可能 状態の制御

enabled プロパティで状態を制御できます。

ヒント文

フィールドが選択されていない状態の時、表示されるヒント文 を hintText プロパティとして指定できます。フォーカスがテキストに来た際に表示がクリアされます。

枠の表示

フィールドの周辺の囲み線をどのように描画するかを borderStyle プロパティで指定します。

`Titanium.UI.INPUT_BORDERSTYLE_ROUNDED`	
`Titanium.UI.INPUT_BORDERSTYLE_LINE`	
`Titanium.UI.INPUT_BORDERSTYLE_BEZEL`	
`Titanium.UI.INPUT_BORDERSTYLE_NONE`	

色の制御

color ならびに backgroundColor プロパティを 16 進数指定することで表示色・背景色を指定できます。

パスワードマスク

パスワードを始めとした機密文字入 力で入力済みの文字をマスクするかどうかを passwordMask プロパティで指定します。

クリアのタイミング

入力前にクリアする場合は clearOnEdit プロパティを true に設定します。

クリアボタン の表示タイミング

入力文字列や初期値のクリアを行うボタンをいつ表示するか を clearButtonMode プロパティで指定できます。

`Titanium.UI.INPUT_BUTTONMODE_NEVER`	常に 表示しないというモードです。	
`Titanium.UI.INPUT_BUTTONMODE_ALWAYS`	常に 表示するというモードです。	
`Titanium.UI.INPUT_BUTTONMODE_ONFOCUS`	フォー カスの持っているときだけ表示します。	
`Titanium.UI.INPUT_BUTTONMODE_ONBLUR`	フォーカスを失っている間だけ表示します。	

左右ボタンの表示タイミ ング

クリアボタン同様に任意のボタンイメージを TextField の 両端に設定できます。

```
var leftButton = Titanium.UI.createButton({
    style:Titanium.UI.iPhone.SystemButton.DISCLOSURE
});
var rightButton = Titanium.UI.createButton({
    style:Titanium.UI.iPhone.SystemButton.INFO_DARK
});
leftButton.addEventListener('click',function()
{
    Titanium.UI.createAlertDialog({
        title:'Text Fields',
        message:'You clicked the left button'
    }).show();
});
rightButton.addEventListener('click',function()
{
    Titanium.UI.createAlertDialog({
        title:'Text Fields',
        message:'You clicked the right button'
```

```
    }).show();
});
var tf1 = Titanium.UI.createTextField({
    color:'#336699',
    height:35,
    top:10,
    left:10,
    width:250,
    borderStyle:Titanium.UI.INPUT_BORDERSTYLE_ROUNDED,
    leftButton:leftButton,
    rightButton:rightButton,
    leftButtonMode:Titanium.UI.INPUT_BUTTONMODE_ALWAYS,
    rightButtonMode:Titanium.UI.INPUT_BUTTONMODE_ONFOCUS
});
```

キーボードツールバーについて

ボタンや他の TextField を ツールバーの部品としてキーボード表示時に連動させることができます。

```
var win = Titanium.UI.currentWindow;
var flexSpace = Titanium.UI.createButton({
    systemButton:Titanium.UI.iPhone.SystemButton.FLEXIBLE_SPACE
});
var tf = Titanium.UI.createTextField({
    height:32,
    backgroundImage:'../images/inputfield.png',
    width:200,
    font:{fontSize:13},
    color:'#777',
    paddingLeft:10,
    borderStyle:Titanium.UI.INPUT_BORDERSTYLE_NONE
});
var camera = Titanium.UI.createButton({
    backgroundImage:'../images/camera.png',
    height:33,
    width:33
})
camera.addEventListener('click', function()
{
    Titanium.UI.createAlertDialog({title:'Toolbar',message:'You clicked
camera!'}).show();
});
var send = Titanium.UI.createButton({
    backgroundImage:'../images/send.png',
    backgroundSelectedImage:'../images/send_selected.png',
    width:67,
    height:32,
});
send.addEventListener('click', function()
{
    Titanium.UI.createAlertDialog({title:'Toolbar',message:'You clicked send!'}).show();
});
var textfield = Titanium.UI.createTextField({
    color:'#336699',
    value:'Focus to see keyboard w/ toolbar',
    height:35,
```

```
    width:300,
    top:10,
    borderStyle:Titanium.UI.INPUT_BORDERSTYLE_ROUNDED,
    keyboardToolbar:[flexSpace,camera, flexSpace,tf,flexSpace, send,flexSpace],
    keyboardToolbarColor: '#999',
    keyboardToolbarHeight: 40,
});
win.add(textfield);
```

関連する API ドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.TextField

UI カタログ(コントロール)- TextArea

TextField と 異なり、こちらは複数行のテキスト入力を行うためのコントロールです。

TextField に 比べると若干設定可能な内容は減ります。


```
var ta1 = Titanium.UI.createTextArea({
    value:'I am a textarea',
    height:70,
```

```
    width:300,
    top:60,
    font:{fontSize:20,fontFamily:'Marker Felt', fontWeight:'bold'},
    color:'#888',
    textAlign:'left',
    appearance:Titanium.UI.KEYBOARD_APPEARANCE_ALERT,
    keyboardType:Titanium.UI.KEYBOARD_NUMBERS_PUNCTUATION,
    returnKeyType:Titanium.UI.RETURNKEY_EMERGENCY_CALL,
    borderWidth:2,
    borderColor:'#bbb',
    borderRadius:5
});
win.add(ta1);
// Text area events
ta1.addEventListener('change',function(e){
    l.text = 'change fired, value = ' + e.value + '\nfield value = ' + ta1.value;
});
ta1.addEventListener('return',function(e){
    l.text = 'return fired, value = ' + e.value;
});
ta1.addEventListener('blur',function(e){
    l.text = 'blur fired, value = ' + e.value;
});
ta1.addEventListener('focus',function(e){
    l.text = 'focus fired, value = ' + e.value;
});
```

関連する APIドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.TextArea

UI カタログ(コントロール) - Switch

オンとオフの状態切り替えを行うためだけの部品になります。 設定画面ぐらいでしか出番がないかもしれません。

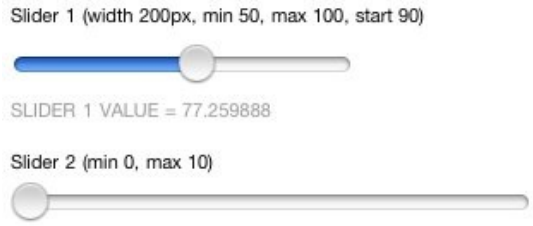

```
var s1 = Titanium.UI.createSwitch({
    value:false,
    top:30,
});
Ti.UI.currentWindow.add(s1);
// create a switch change listener
s1.addEventListener('change', function(e) {
  // e.value にはスイッチの新しい値が true もしくは false として設定されます。
});
```

関連する API ドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.Switch

UI カタログ(コントロール) - Slider

Slider は音量や数量を調整するのに使われるアナログ入力コントロールです。



上限・下限の範囲を設定し、スライダーのレバーとして値を表現します。

初期値・範囲についてはコントロール作成時にのみ指定できます。

```
// slider1 を作成し配置
var slider1 = Titanium.UI.createSlider({
  min:50,
  max:100,
  value:90,
  width:200,
  height:'auto',
  top: 30
});
Titanium.UI.currentWindow.add(slider1);
slider1.addEventListener('change',function(e) {
  // e.value に現在の値が入ります。
});
```

関連する API ドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.Slider

UI カタログ(コントロール) - Picker

コンボボックスに相当する複数の候補から項目をドラム式に選択させるコントロールで す。

項目が多すぎると正直使い勝手が悪いので、そういう ときは TableView などで外に出せないか考慮する必要があるかもしれません。

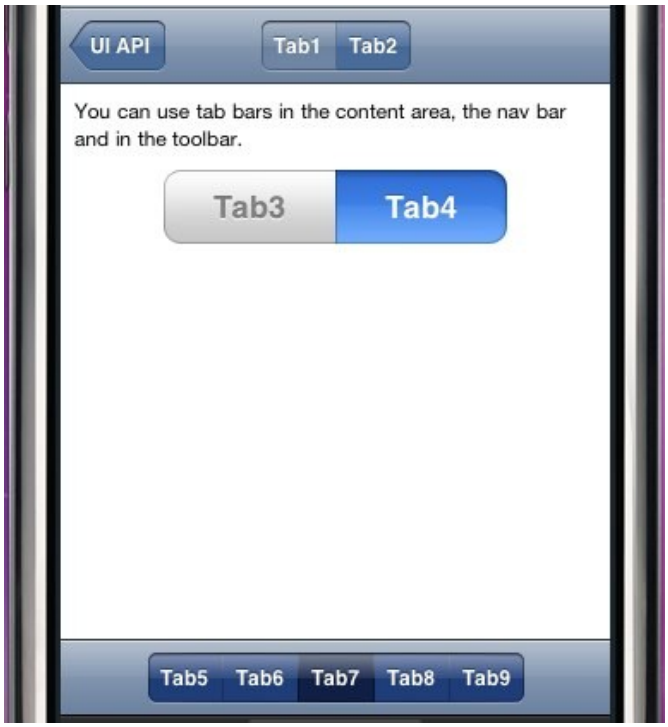

```
var picker = Ti.UI.createPicker();
var data = [];
data[0]=Ti.UI.createPickerRow({title:'Bananas',custom_item:'b'});
data[1]=Ti.UI.createPickerRow({title:'Strawberries',custom_item:'s'});
data[2]=Ti.UI.createPickerRow({title:'Mangos',custom_item:'m'});
data[3]=Ti.UI.createPickerRow({title:'Grapes',custom_item:'g'});
picker.add(data);
// 選択表示を有効にします（標準は無効）
picker.selectionIndicator = true;
// スタイルを指定します。
picker.type = Ti.UI.PICKER_TYPE_PLAIN;
/*
// iPhone では現在 PLAIN しか対応していません！
Ti.UI.PICKER_TYPE_COUNT_DOWN_TIMER
Ti.UI.PICKER_TYPE_DATE
Ti.UI.PICKER_TYPE_DATE_AND_TIME
Ti.UI.PICKER_TYPE_PLAIN
Ti.UI.PICKER_TYPE_TIME
*/
win.add(picker);
picker.addEventListener('change',function(e){
  // e.row, e.column として選択項目が取得できます
  // e.row.custom_item として各列のカスタムデータが帰ります
});
```

関連する API ドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.Picker

UI カタログ(コントロール) - TabbedBar (iPhone のみ)

TabbedBar と は次の画像の NavBar の 中央, コンテンツエリアの上段、ToolBar の それぞれに配置されているコントロールのことをさします。



こ のコントロールは複数の状態を切り替えるのに用います。単なる ON/OFF の場合は Switch が 適していますが、それ以外の状態の切り替えにはこれを利用したほうがよいでしょう。

上記の画面のコード例は以下のようになります。

```
// スペイサーとして用いる特殊なボタンを宣言しておく
var flexSpace = Titanium.UI.createButton({
  systemButton:Titanium.UI.iPhone.SystemButton.FLEXIBLE_SPACE
});
// NavBar に配置するインスタンス
var tabbar1 = Titanium.UI.createTabbedBar({
   labels: ['Tab1', 'Tab2'],
   index:1
});
// Toolbar に配置するインスタンス
var tabbar2 = Titanium.UI.createTabbedBar({
 labels:['Tab5', 'Tab6','Tab7', 'Tab8','Tab9'],
 backgroundColor:'#336699',
 index:2
});
// Navbar の中央に配置
Titanium.UI.currentWindow.setTitleControl(tabbar1);
// Toolbar の中央に配置
Titanium.UI.currentWindow.setToolbar([flexSpace, tabbar2, flexSpace]);
```

```
// コンテンツエリアに配置
var tabbar = Titanium.UI.createTabbedBar({
  index:1,
  labels:['Tab3', 'Tab4'],
  top:50,
  style:Titanium.UI.iPhone.SystemButtonStyle.BAR,
  height:25,
  width:200
});
Titanium.UI.currentWindow.add(tabbar);
```

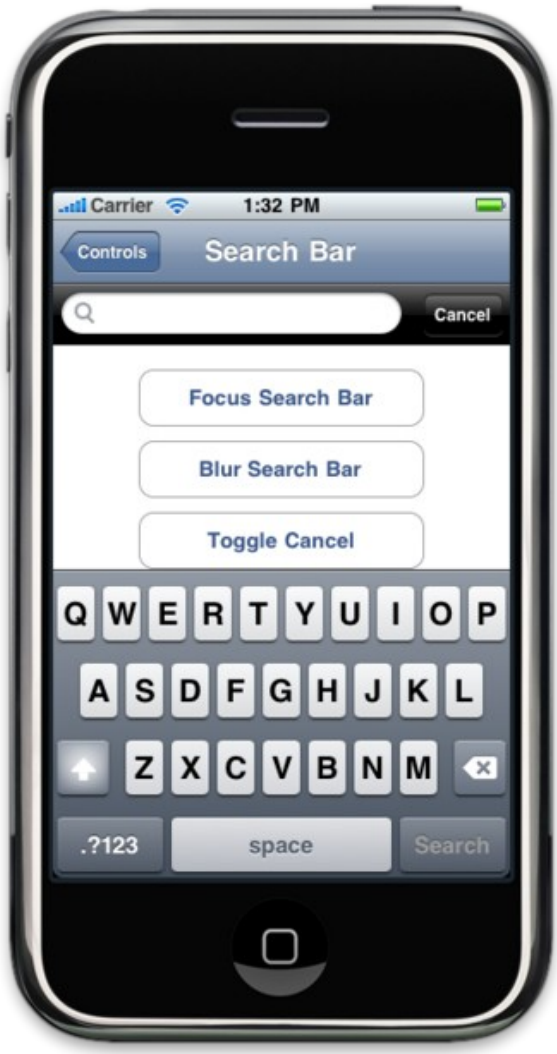
関連するAPIドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.TabbedBar

UIカタログ(コントロール) - SearchBar

検索用のツールバーです。

結果表示のための TableView や WebView, MapView な どとセットで登場するかたちになります。



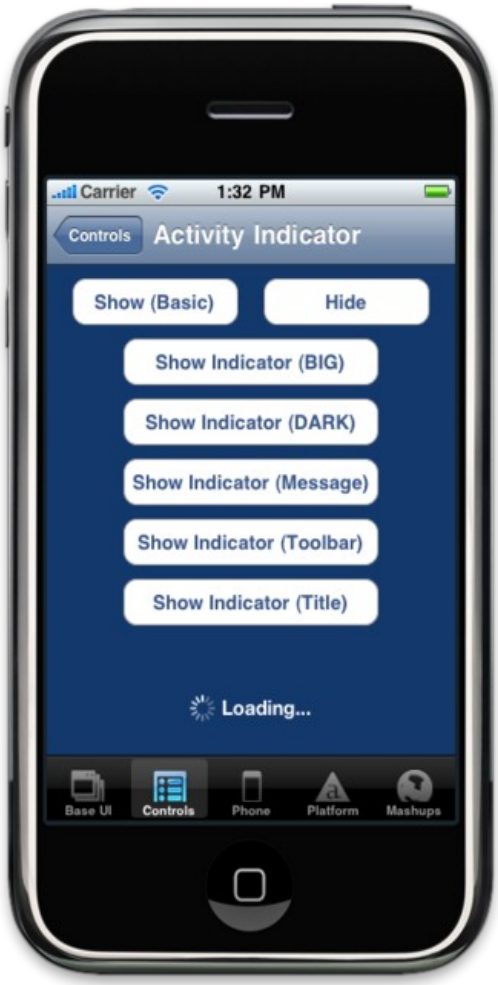
```
// コントロールの配置
var search = Titanium.UI.createSearchBar({
    barColor:'#000',
    showCancel:true,
    height:43,
    top:0,
});
Titanium.UI.currentWindow.add(search);
// SearchBar 自体のイベント
search.addEventListener('change', function(e){
    Titanium.API.info('search bar: you type ' + e.value + ' act val ' + search.value);


});
search.addEventListener('cancel', function(e){
    Titanium.API.info('search bar cancel fired');
    search.blur();
});
search.addEventListener('return', function(e){
    Titanium.UI.createAlertDialog({title:'Search Bar', message:'You typed ' +
e.value }).show();
    search.blur();
});
search.addEventListener('focus', function(e){
    Titanium.API.info('search bar: focus received')
});
search.addEventListener('blur', function(e){
    Titanium.API.info('search bar:blur received')
});
```

関連する API ドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.SearchBar

UI カタログ(コントロール) - ActivityIndicator

現在なんらかの処理中であることを示すアイコンとメッセージを表 示するコントロールです。




```
var actInd = Titanium.UI.createActivityIndicator({
    bottom:10,
    height:50,
    width:10,
    style:Titanium.UI.iPhone.ActivityIndicatorStyle.PLAIN
});
var button1 = Titanium.UI.createButton({
    title:'Show',
    height:35,
    width:130,
    top:10,
    left:20,
});
button1.addEventListener('click', function(){
    // 表示形式は以下の種類から選べます。
    /*
    actInd.style = Titanium.UI.iPhone.ActivityIndicatorStyle.PLAIN;
    actInd.style = Titanium.UI.iPhone.ActivityIndicatorStyle.DARK;
    actInd.style = Titanium.UI.iPhone.ActivityIndicatorStyle.BIG
    */
    // メッセージ関連の設定
    actInd.font = {fontFamily:'Helvetica Neue', fontSize:15,fontWeight:'bold'};
    actInd.color = 'white';
```

```
    actInd.message = 'Loading...';
    actInd.width = 210;
    // 表示します
    actInd.show();
});
```

関連する APIドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.ActivityIndicator

UI カタログ(コントロール) - ProgressBar

進捗表示を行うコントロールです。

進捗を返すイベントリスナとセットで利用することになるでしょう。

たとえば、ダウンロードやアップロードの状態を返すイベントなどが対象にな ります。


```
var ind=Titanium.UI.createProgressBar({
    width:150,
    min:0,
    max:10,
```

```
    value:0,
    height:70,
    color:'#888',
    message:'Downloading 0 of 10',
    font:{fontSize:14, fontWeight:'bold'},
    style:Titanium.UI.iPhone.ProgressBarStyle.PLAIN,
    top:60
});
var interval = setInterval(function(){
    if (ind.val == 10){
        clearInterval(interval);
        ind.hide();
        return;
    }
    ind.value = ind.value + 1;
    ind.message = 'Downloading ' + val + ' of 10';
},1000);
```

関連する APIドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.ProgressBar

UI カタログ(API) - アニメーション

Window, View, Control などの各オブジェクトには `animate` と いう関数が用意されており、対象となるオブジェクトに対するアニメーション描画を行うことができます。

その際に引き渡される引数が `Titanium.UI.Animation` オ ブジェクトになり、次のような多くのパラメータとなるプロパティを持っています。

名前	型	説 明
autoreverse	boolean	アニメーション完了後に元に戻るかどうかを指定
backgroundColor	string	色アニメーション：背景 色
bottom	float	移動アニメーション：bottom 位置
center	object	移動アニメーション：対 象オブジェクトの中心座標
color	string	色アニメーション：表示色
curve	int	変形アニメーション：曲 線の状態を指定。Ti.UI.ANIMATION_CURVE_EASE_IN, Ti.UI.ANIMATION_CURVE_EASE_IN_OUT, Ti.UI.ANIMATION_CURVE_EASE_OUT, Ti.UI.ANIMATION_CURVE_LINEAR
delay	float	開始まで遅延時間（単位：ミリ秒）
duration	float	アニメーションに掛ける 時を（単位：ミリ秒）
height	float	変形アニメーション：高さ
left	float	変形アニメーショ ン：left 位置

opacity	float	色アニメーション：透過度
opaque	boolean	色アニメーション：透 過・非透過の切替アニメーション
repeat	int	アニメーション回数
right	float	変形アニメーショ ン：right 位置
top	float	変形アニメーション：top 位置
transform	object	2DMatrix, 3DMatrix の値を設定し、変形アニメーション指定をする
transition	int	規定のパターンに基づくアニメーション
visible	boolean	表示・非表示の切替アニ メーション
width	float	変形アニメーション：幅
zIndex	int	移動アニメーショ ン：zIndex

例え ば次のようなコードが例ですが、もともと赤い背景色が 1000ms(1 秒)かけて黒にフェイドしていき、その後、さらに 1 秒かけてオレンジ色に変化してい くというものになります。

```
var view = Titanium.UI.createView({
  backgroundColor:'red'
});
// イベントリスナに登録するため、変数化している。
// 実際には JSON 形式で渡す省略記法もある（後述）
var animation = Titanium.UI.createAnimation();
animation.backgroundColor = 'black';
animation.duration = 1000;
animation.addEventListener('complete',function(){
   animation.removeEventListener('complete',this);
   animation.backgroundColor = 'orange';
   view.animate(animation);
});
// View に対してアニメーションを指示している。
view.animate(animation);
```

また animate には Animation オブジェクトに続いて、完了後のコールバック関 数も引数にすることができます。

次の例は 4 段階にアニメー ションが変化していくというものです。

```
// 枠線によって円形にしている View を配置し、
// これに対するアニメーションを指示する、というものです。
var circle = Titanium.UI.createView({
   height:100,
   width:100,
   borderRadius:50,
   backgroundColor:'#336699',
   top:10
});
//
// ここでは Titanium.UI.Animation ではなく、JSON 形式で直接指示している。
//
// STEP 1. 中心座標(100, 100)に移動
circle.animate({center:
```

```
{x:100,y:100},curve:Ti.UI.ANIMATION_CURVE_EASE_IN_OUT,duration:1000}, function(){
  // STEP 2. 中心座標(0, 200)に移動
  circle.animate({center:{x:0,y:200},duration:1000}, function(){
    // STEP 3. 中心座標(300, 300)に移動
    circle.animate({center:{x:300,y:300},duration:1000},function(){
      // STEP 4. 元の位置である中心座標(150, 60)に戻る
      circle.animate({center:{x:150,y:60, duration:1000}});
    });
  })
});
```

Transition アニメーション

Window や View に対してフリップ移動やカールするアニメーションがありますが、ああいうこと を行うための指定が transition アニメーションになります。

指 定できるのは次のとおり５つです。

- Ti.UI.iPhone.AnimationStyle?.CURL_UP
- Ti.UI.iPhone.AnimationStyle?.FLIP_FROM_LEFT
- Ti.UI.iPhone.AnimationStyle?.FLIP_FROM_RIGHT
- Ti.UI.iPhone.AnimationStyle?.CURL_DOWN
- Ti.UI.iPhone.AnimationStyle?.NONE

```
var button = Titanium.UI.createButton({
  title:'Animate Me',
  width:300,
  height:40,
  top:10
});
button.addEventListener('click', function(){
  button.animate({
    transition:Ti.UI.iPhone.AnimationStyle.FLIP_FROM_LEFT
  });
});
```

https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.iPhone.AnimationStyle

2DMatrix, 3DMatrix による変形アニメーション

2DMatrix, 3DMatrix は変形内容の指示を構成するオブジェクトで、Animation オブジェクトに与えることを目的としています。

さっそく比較的理解しやすい平面変形をする例を見てみましょう。

```
// 画像を表示した view を変形させます
var cloud = Titanium.UI.createView({
  backgroundImage:'../images/cloud.png',
  height:178,
  width:261,
  top:10
});
var button = Titanium.UI.createButton({
  title:'Animate',
  width:200,
```

```
  height:40,
  bottom:20
});
button.addEventListener('click', function(){
  // 先ほどの画像を変形アニメーションします。
  var t = Ti.UI.create2DMatrix();
  t = t.rotate(20);
  t = t.scale(1.5);
  var animation = Titanium.UI.createAnimation();
  animation.transform = t;
  animation.duration = 3000;
  animation.autoreverse = true;
  animation.repeat = 3;
  // 上記の設定は以下のようなものです。
  // ---------------------------------
  // 「全体的に 20 度右回転(t.rotate)し、1.5 倍に拡大(t.scale)する」変形(transform)を
  // 3000 ミリ秒(duration)かけて行い、その後同じ時間かけて元に戻す(autoreverse)
  // 処理を 3 回繰り返す(repeat)
  cloud.animate(animation);
```

このように 2DMatrix オブジェクトにエフェクトを重ねていき、 Animation オブジェクトに設定するだけで変形を行えます。

回転エフェクトをする場合、回転する UI 部品の anchorPoint プ ロパティに設定することで回転軸をどこにするか設定できます。

```
var v = Titanium.UI.createView({
  backgroundColor:'#336699',
  top:10,
  left:220,
  height:50,
  width:50,
  anchorPoint:{x:0,y:0}
});
var t = Ti.UI.create2DMatrix();
t = t.rotate(90);
var a = Titanium.UI.createAnimation();
a.transform = t;
a.duration = 1000;
a.autoreverse = true;
v.animate(a);
```

この場合、左上を中心に９０度回転します。

anchorPoint プロパティは x, y で回転軸を指定します。右下が `{x:1, y:1}` となります。中心では `{x:0.5, y:0.5}` です。

続いて 3DMatrix による３次元 変形の例です。

```
// ここから先の解説は API リファレンス待ちです (>_<)
var button = Titanium.UI.createButton({
  title:'Animate Me',
  width:300,
  height:40,
  top:10
});
```

```
button.addEventListener('click', function(){
  var t = Titanium.UI.create3DMatrix();
  t = t.rotate(200, 0, 1, 1);
  t = t.scale(3);
  t = t.translate(20, 50, 170);
  t.m34 = 1.0/-2000;
  button.animate({
    transform:t,
    duration:1000,
    autoreverse:true
  });
});
```

関連するAPIドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.Animation

3rd Step : API カタログ

API カタログ(ネットワーク編) - ネットワークの状態

コード例・解説

```
// 通信状態じゃないと使えないアプリ(AppStoreとかもそうですが)がありますが、
// まずネットワークの状態を判断できる必要がありますね。
// 今現状のネットワーク状態を取得する
if(Titanium.Network.online){
  // boolean値で接続状態が返ります。
  // さらにどんなネットワークにつながっているのかを判断できます。
  var nt = Titanium.Network.networkType;
  /*
   候補として以下のいずれかの値になります。
   ・Titanium.Network.NETWORK_LAN
   ・Titanium.Network.NETWORK_MOBILE
   ・Titanium.Network.NETWORK_WIFI
   ・Titanium.Network.NETWORK_NONE
   ・Titanium.Network.NETWORK_UNKNOWN
   文字列として判断したい場合は Titanium.Network.networkTypeName を使ってください。
   */
}
// 接続状態が変わる度にイベントを発生させるリスナーです。
Titanium.Network.addEventListener('change', function(e) {
   var online = e.online;
   var type = e.networkType;
   var networkTypeName = e.networkTypeName;
});
```

関連する APIドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.Network

API カタログ(ネットワーク編) - HTTPClient による通信

HTTPClient は AJAX でおなじみの XMLHttpClient(xhr)と同じ仕組みで動くので、prototype.js や jQuery などのラッパーなしで素の AJAX をしていた向きには理解しやすいかもしれません。

基本構文

```
// オフラインなら処理しないようにしたほうがいいですよね！
if(Titanium.Network.online == false){
   // エラー表示
   return;
}
// オブジェクトを生成します。
var xhr = Titanium.Network.createHTTPClient();
```

```
// 第一引数は HTTP Method(GET か POST がほとんどだと思いますが)
// 第二引数は URI です。
xhr.open('GET','http://search.twitter.com/search.json?q=%23titanium');
// レスポンスを受け取るイベント
xhr.onload = function(){
  alert(this.responseText);
  /*
  // これと同義
  xhr.onreadystatechange = function(){
    if(this.readyState == xhr.DONE){
      alert(this.responseText);
    }
  };
  */
};
// エラー発生時のイベント
xhr.onerror = function(error){
  // error にはエラー事由の文字列オブジェクトが入ってくる。
};
// リクエスト送信します。(引数として JSON 値を入れるとパラメータ化される)
xhr.send();
/*
 xhr.send({
   q : 'querystring',
   param_name : 'param_value'
 });
 */
```

基本的に onload でレスポンスを操作するという流れです。

responseText のほかに DOMParser に予め通した形で受け取る responseXML というプロパティもあります。

JSON の取得

JSON 化する場合は次のようにメソッドを 使います。

```
xhr.onload = function(){
  var json = JSON.parse(this.responseText);
  // 後続処理
};
```

バイナリデータの取得

上の例ではテキスト データの取得ですが、画像や PDF などをダウンロードする場合は次のように記述します。

```
xhr.onload = function(){
  var f = Ti.Filesystem.getFile(Ti.Filesystem.applicationDataDirectory, 'tmp.png');
  f.write(this.responseData);
  // 画像を表示する場合
  // imageView.url = f.nativePath;
  //
  // PDF を表示する場合
  // webView.url = f.nativePath;
```

```
};
```

写真を POST する例 + 進捗表示

カメラ撮影をしたものを TwitPic サーバにアップロードする例で す。

```
Titanium.Media.showCamera({
  success : function(event){
    try{
      var xhr = Titanium.Network.createHTTPClient();
      xhr.onload = function(){
        var xml = this.responseXML;
        var url = xml.documentElement.getElementsByTagName("mediaurl")[0].nodeValue;
        // ...
      };
      // 送信時処理(進捗表示など)
      xhr.onsendstream = function(e){
        Ti.API.info(e.progress);
      };
      xhr.open("POST","http://twitpic.com/api/upload");
      xhr.send({media:event.media, username:'username', password:'password'});
    }
    catch(error){
      // ...
    }
  }
});
```

上 記で書いているアップロード進捗の反対、ダウンロードの進捗も別のイベントでフォローされています。

```
xhr.ondatastream = function(e){
  // e.progress で進捗を取得できる
};
```

標準認証

実はあまりこの HTTPClient、あまりスマートにでき ていません。

標準認証時は次のようにしろとドキュメントに書かれていて思わず脱力してしまいま す。

```
try{
  var xhr = Titanium.Network.createHTTPClient();
  xhr.onload = function() {
    //do work on "this.responseXML"
  };
  xhr.open("GET","https://"+username + ":" + password +
"@twitter.com/account/verify_credentials.xml");
  xhr.send();
}
catch(error){
  Titanium.UI.createAlertDialog({
    title: "Error",
    message: String(error),
```

```
    buttonNames: ['OK']
  }).show();
}
```

URL に直接ユーザ名とパスワードを埋め込む方式です。

このあたりは 今後改善されていくと信じたいところです。

リクエストヘッダの追加

HTTP リクエストヘッダに対する操作も可能です。

```
// UA の指定
xhr.setRequestHeader('User-Agent','Mozilla/5.0 (iPhone; U; CPU like Mac OS X; en)
AppleWebKit/420+ (KHTML, like Gecko) Version/3.0 Mobile/1A537a Safari/419.3');
// 認証系情報
xhr.setRequestHeader('Authorization','Basic
'+Ti.Utils.base64encode(username.value+':'+password.value));
```

関連する API ドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.Network.HTTPClient

API カタログ(I/O 編) - アプリケーションプロパティ

アプリケーショ ン上での設定を保存するためのの仕掛けが用意されています。

`Titanium.App.Properties` には set〜と get〜という関数が型単位で用意されています。

型	get〜	set〜
string	getString	setString
int	getInt	setInt
boolean	getBool	setBool
double	getDouble	setDouble
Array	getList	setList

第一引数はプロパティ名で、set〜の場合、第二引数にプロパティに設定す る値をセットします。

プロパティを削除する 場合は第二引数に `null` を設定してください。

```
// 出力
Titanium.App.Properties.setString('String','I am a String Value ');
Titanium.App.Properties.setInt('Int',10);
Titanium.App.Properties.setBool('Bool',true);
Titanium.App.Properties.setDouble('Double',10.6);
Titanium.App.Properties.setList('MyList',array);
// 入力（表示はログレベル info で）
Titanium.API.info('String: '+ Titanium.App.Properties.getString('String'));
Titanium.API.info('Int: '+ Titanium.App.Properties.getString('Int'));
Titanium.API.info('Bool: '+ Titanium.App.Properties.getString('Bool'));
Titanium.API.info('Double: '+ Titanium.App.Properties.getString('Double'));
Titanium.API.info('List:');
```

また現在のプロパティ一覧の取得やプロパティの存在確認を行うためのメ ソッドもあります。

```
// プロパティの存在確認をします
var bool = Titanium.App.Properties.hasProperty("prop_name");
// プロパティ名の配列を返します。
var list = Titanium.App.Properties.listProperties()
```

関連する API ドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.App.Properties

API カタログ(I/O 編) - ファイルシステム

ファイルシステムの操作には今のところプロパティ取得と内容の読取りしかないので、 次のような一覧表でのご紹介に留めさせていただきます。

パス関連情報の取得

API	内容	実行例
Titanium.Filesystem.resourcesDirectory	Resources Directory	/Users/username/Library/Application Support/iPhone Simulator/User/Applications/94646704-B9B9?-432F-A8CE-640A89A6B55C/KitchenSink.app
Titanium.Filesystem.tempDirectory	Temp Directory	/ var/folders/fy/fyJLG5EWEcSF52GCubOqmU+++TI/-Tmp-/
Titanium.Filesystem.applicationDirectory	Application Directory	/Users/username/Library/Application Support/iPhone Simulator/User/Applications/94646704-B9B9?-432F-A8CE-640A89A6B55C/Applications
Titanium.Filesystem.applicationDataDirectory	Application Data Directory	/Users/username/Library/Application Support/iPhone Simulator/User/Applications/94646704-B9B9?-432F-A8CE-640A89A6B55C/Documents
Titanium.Filesystem.isExteralStoragePresent	External Storage Available	false
Titanium.Filesystem.separator	Path Separator	/
Titanium.Filesystem.lineEnding	Line Ending	

ファイル情報の取得

次の処理で取得されたファイルオブジェクトのプロパティとして取得します。

```
var file = Titanium.Filesystem.getFile(Titanium.Filesystem.resourcesDirectory,
'text.txt');
var contents = file.read();
```

API	内容	実行例
file	file	/Users/username/Library/Application Support/iPhone Simulator/User/Applications/94646704-B9B9?-432F-A8CE-640A89A6B55C/KitchenSink.app/text.txt
contents	contents blob object	TiBlob?
contents.text	contents	Hello World. this is a filesystem read test.
contents.mimeType	mime type	text/plain
file.nativePath	nativePath	/Users/username/Library/Application Support/iPhone Simulator/User/Applications/94646704-B9B9?-432F-A8CE-640A89A6B55C/KitchenSink.app/text.txt
file.exists()	exists	true
file.size	size	44
file.readonly	readonly	true
file.symbolicLink	symbolicLink	false
file.executable	executable	false
file.hidden	hidden	false
file.writable	writable	false
file.name	name	text.txt
file.extension()	extension	txt
file.resolve()	resolve	/Users/username/Library/Application Support/iPhone Simulator/User/Applications/94646704-B9B9?-432F-A8CE-640A89A6B55C/KitchenSink.app/text.txt

ディレクトリ情報の取得

ファイル同様、ディレクトリオブジェクトからのプロパティ取得となります。

```
var dir = Titanium.Filesystem.getFile(Titanium.Filesystem.resourcesDirectory);
```

API	内容	実行例
dir.getDirectoryListing()	directoryListing	cricket.wav,Default.png,default_app_logo.png,dependencies.map,Entitlements.plist,images,Info.plist,KitchenSink,MainWindow?.nib,modules,movie.mp4,PkgInfo?,pop.caf,Settings.bundle,testdb.db,text.txt
dir.getParent()	getParent	/Users/username/Library/Application Support/iPhone Simulator/User/Applications/94646704-B9B9?-432F-A8CE-640A89A6B55C
dir.spaceAvailable()	spaceAvailable	true

関連するAPIドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.Filesystem

API カタログ(I/O 編) - データベース

デバイス上の SQLite データベースへの接続をするをするた めの機能が用意されています。

OR マッピング的なものはないので、自前で SQL の操作をする必要があるため、SQL の知識が必要になってきます。

```
// データベースファイルを開きます（ない場合、作成されます）
var db = Titanium.Database.open('mydb');
// DB 内にテーブルが無い場合、定義に基づいてテーブルを作成します。
db.execute('CREATE TABLE IF NOT EXISTS DATABASETEST  (ID INTEGER, NAME TEXT)');
// ちなみに TABLE を削除するときは DROP TABLE DATABASETEST みたいにします
// テストなので全行消してから、テストデータを追加→更新＋削除します。
// プレースホルダはありますが、名前ベースじゃなくて?の発生順での置換です。
db.execute('DELETE FROM DATABASETEST');
db.execute('INSERT INTO DATABASETEST (ID, NAME ) VALUES(?,?)',1,'Name 1');
db.execute('INSERT INTO DATABASETEST (ID, NAME ) VALUES(?,?)',2,'Name 2');
db.execute('INSERT INTO DATABASETEST (ID, NAME ) VALUES(?,?)',3,'Name 3');
db.execute('INSERT INTO DATABASETEST (ID, NAME ) VALUES(?,?)',4,'Name 4');
db.execute('UPDATE DATABASETEST SET NAME = ? WHERE ID = ?', 'I was updated', 4);
db.execute('UPDATE DATABASETEST SET NAME = "I was updated too" WHERE ID = 2');
db.execute('DELETE FROM DATABASETEST WHERE ID = ?',1);
// 直前の実行に伴う影響行数と最後に追加された rowId を取得
Titanium.API.info('JUST INSERTED, rowsAffected = '    + db.rowsAffected);
Titanium.API.info('JUST INSERTED, lastInsertRowId = ' + db.lastInsertRowId);
// 照会は普通に SELECT 文を実行します。結果は ResultSet オブジェクトとして返ります。
var rows = db.execute('SELECT * FROM DATABASETEST');
Titanium.API.info('ROW COUNT = ' + rows.getRowCount());
// ResultSet はカーソル的な挙動をしますので、こういったループで走査していきます。
while(rows.isValidRow()){
  // rows.field(field_no)で値を取得します。
  // カラム名ベースでも取れるみたいです。
  Titanium.API.info('ID: ' + rows.field(0) + ' NAME: ' + rows.fieldByName('name'));
  rows.next();
}
// 走査が終わったら ResultSet を閉じておきます。
rows.close();
```

既存の SQLiteDB の取込み

あらかじめ用意しておいた SQLite のデータベースファイルを取り込むことができます。

```
// ../testdb.db の内容を quotes データベースに取り込み、開きます。
var db = Titanium.Database.install('../testdb.db','quotes');
// あとの流れは同じですね。
var rows = db.execute('SELECT * FROM TIPS');
while (rows.isValidRow()){
  Titanium.API.info(rows.field(1) + '\n' + rows.field(0));
  rows.next();
}
rows.close();
```

関連する API ドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.Database
- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.Database.open

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.Database.install
- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.Database.DB
- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.Database.ResultSet

API カタログ(メディア編) - カメラ撮影・フォトギャラリーからの取得・スクリーンショット

カメラでの撮影、フォトギャラリーからのデータ取得はいずれも非同期に行われる ため、コールバック関数によって取得するかたちになります。そこで引き渡される画像・サムネイルのデータ、補助情報を利用し、プレビューしたりアップロー ドしたりします。

ちなみに allowImageEditing プロパティを true と指定した場合は拡大や移動をするための画面を一段挟み、その結果が succuess プロパティで呼び出される関数に引き渡されま す。

カメラ撮影

```
btnCamera.addEventListener('click', function(){
  Titanium.Media.showCamera({
    // JSON形式の引数です
    success:function(event){
      // cropRectにはx,y,height,widthといったデータがはいる。
      var cropRect = event.cropRect;
      // 画像イメージ(Blobデータ)が返される
      var image = event.media;
      // サムネイルイメージ(Blobデータ)が返される
      var thumbnail = event.thumbnail;

      // 撮影データを表示する
      var imageView = Ti.UI.createImageView({image:event.media});
      Ti.UI.currentWindow.add(imageView);
    },
    cancel:function(){
      //
    },
    error:function(error){
      // errorとしてカメラがデバイスにないようなケース(iPod touchなどがそうでしょうか)では
      // error.code が Titanium.Media.NO_CAMERA として返ります。
    },
    allowImageEditing:true,
    saveToPhotoGallery: false
  });
});
```

撮影成功時に本体フォトギャラリーに画像を保存するだけの場合は、showCamera の引数として saveToPhotoGallery を true として設定するだけです。

ファイルへの書 き出しをする場合、succuess 内で次のように出力処理を記述します。

```
// アプリケーションデータディレクトリに camera_photo.png として出力する。
var f = Titanium.Filesystem.getFile(Titanium.Filesystem.applicationDataDirectory,
'camera_photo.png');
f.write(event.media);
```

```
// 現在のウィンドウ背景画像としてそのまま使う場合は次のようにする
Titanium.UI.currentWindow.backgroundImage = f.nativePath;
```

フォトギャラリー側から写真選択

撮影済みのデータから処理対象を選択する場合、次のように記述します。

結果が event.media として返ってくるので、それをアップロードしたりする感じになります。

```
Titanium.Media.openPhotoGallery({
  success: function(event) {
    // カメラロールで写真を選択した時の挙動(カメラと同様)
  },
  error: function(error) {
    // notify(e.message);
  },
  cancel: function() {
    // キャンセル時の挙動
  },
  allowImageEditing: true
});
```

スクリーンショットの取得

使いどこ ろが難しいのですが、デバイス画面をカメラ同様に取得することができます。

```
Titanium.Media.takeScreenshot(function(event){
  var image = event.media;
  var imageView = Ti.UI.createImageView({image:event.media});
  Ti.UI.currentWindow.add(imageView);
});
```

関連する APIドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.Media

API カタログ(メディア編) - 動画再生・録画

動画再生

```
var video = Titanium.Media.createVideoPlayer({
  movieControlMode:Titanium.Media.VIDEO_CONTROL_DEFAULT,
  scalingMode:Titanium.Media.VIDEO_SCALING_ASPECT_FILL,
  //contentURL:movieFile.nativePath
  media:movieFile // note you can use either contentURL to nativePath or the file object
});
video.play();
```

再生時のアスペクト比の指定

VIDEO_SCALING_ASPECT_FILL	constant for video aspect where the movie will be scaled until the movie fills the entire screen. Content at the edges of the larger of the two dimensions is clipped so that the other dimension fits the screen exactly. The aspect ratio of the movie is preserved.
VIDEO_SCALING_ASPECT_FIT	constant for video aspect fit where the movie will be scaled until one dimension fits on the screen exactly. In the other dimension, the region between the edge of the movie and the edge of the screen is filled with a black bar. The aspect ratio of the movie is preserved.
VIDEO_SCALING_MODE_FILL	constant for video aspect where the movie will be scaled until both dimensions fit the screen exactly. The aspect ratio of the movie is not preserved.
VIDEO_SCALING_NONE	constant for video scaling where the scaling is turn off. The movie will not be scaled.

動画ストリーミング再生

```
var video = Titanium.Media.createVideoPlayer({
  // リモート再生も可能です
  //  contentURL:'http://movies.apple.com/media/us/ipad/2010/tours/apple-ipad-video-us-20100127_r848-9cie.mov'
  contentURL : "movie.mp4"
});
video.addEventListener("complete", function() {
  video.removeEventListener('complete', listenerId);
  var dialog = Titanium.UI.createAlertDialog({
    'title' : '再生終了',
    'message' : 'video completed',
    'buttonNames' : [ 'OK' ]
  });
  dialog.show();
});
video.play();
```

録画

静止画撮影同様に Titanium.Media.showCamera を 用いますが、引数で `mediaTypes: Titanium.Media.MEDIA_TYPE_VIDEO` と指定する点が異なり ます。

```
Titanium.Media.showCamera({
  success: function(event){
    var video = event.media;
    movieFile = Titanium.Filesystem.getFile(Titanium.Filesystem.applicationDataDirectory,'mymovie.mov');
    movieFile.write(video);
  },
  cancel:function(){
  },
  error:function(error){
    // create alert
    var a = Titanium.UI.createAlertDialog({title:'Video'});
    // set message
    if (error.code == Titanium.Media.NO_VIDEO){
      a.setMessage('Device does not have video recording capabilities');
    }
```

```
    else{
      a.setMessage('Unexpected error: ' + error.code);
    }
    a.show();
  },
  mediaTypes: Titanium.Media.MEDIA_TYPE_VIDEO,
  videoMaximumDuration:10000,
  videoQuality:Titanium.Media.QUALITY_HIGH
});
```

録画品質

QUALITY_HIGH	media type constant to use high-quality video recording. Recorded files are suitable for on-device playback and for wired transfer to the Desktop using Image Capture; they are likely to be too large for transfer using Wi-Fi.
QUALITY_LOW	media type constant to use use low-quality video recording. Recorded files can usually be transferred over the cellular network.
QUALITY_MEDIUM	media type constant to use medium-quality video recording. Recorded files can usually be transferred using Wi-Fi. This is the default video quality setting.

関連する API ド キュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.Media

API カタログ(メディア編) - 音声再生・録音

音声再生

ループ再生とかボリュームコントロールなども API もありま すが、とりあえず再生機能のサンプルをどうぞ。

```
var sound = Titanium.Media.createSound({
  // リモート URL も指定できます
  // url : "http://www.nch.com.au/acm/8kmp38.wav"
  url: '../cricket.wav'
});
sound.addEventListener('complete', function() {
  sound.removeEventListener('complete', listenerId);
  var dialog = Titanium.UI.createAlertDialog({
    'title' : 'Sound Complete',
    'message' : 'sound completed',
    'buttonNames' : [ 'OK' ]
  });
  dialog.show();
});
sound.play();
```

音声ストリーム再生

音声のストリーミング再生の 場合は start メソッドでの実行となります。

```
// 適当な位置に進捗ラベルを表示します(^^;
var progressLabel = Titanium.UI.createLabel();
Ti.UI.currentWindow.add(progressLabel);
var stream = Titanium.Media.createSound({
  url: 'http://202.6.74.107:8060/triplej.mp3'
});
stream.addEventListener('progress', function(e) {
  progressLabel.text = 'Time Played: ' + Math.round(e.progress) + ' seconds';
});
sound.start();
```

録音(作業中)

```
var recording = Ti.Media.createAudioRecorder();
var file = null;
// default compression is Ti.Media.AUDIO_FORMAT_LINEAR_PCM
// default format is Ti.Media.AUDIO_FILEFORMAT_CAF
recording.compression = Ti.Media.AUDIO_FORMAT_ULAW;
recording.format = Ti.Media.AUDIO_FILEFORMAT_WAVE;
// 5秒間の録音
recording.start();
Ti.Media.startMicrophoneMonitor();
setTimeout(function(e){
  file = recording.stop();
  Ti.Media.stopMicrophoneMonitor();

  var sound = Titanium.Media.createSound({sound:file});
  sound.addEventListener('complete', function()       {
    sound = null;
  });
  sound.play();
}, 5000);
```

音声ファイルのフォーマット

- AUDIO_FILEFORMAT_3GP2
- AUDIO_FILEFORMAT_3GPP
- AUDIO_FILEFORMAT_AIFF
- AUDIO_FILEFORMAT_AMR
- AUDIO_FILEFORMAT_CAF
- AUDIO_FILEFORMAT_MP3
- AUDIO_FILEFORMAT_MP4
- AUDIO_FILEFORMAT_MP4A
- AUDIO_FILEFORMAT_WAVE

圧縮形式

- AUDIO_FORMAT_AAC
- AUDIO_FORMAT_ALAW
- AUDIO_FORMAT_APPLE_LOSSLESS
- AUDIO_FORMAT_ILBC
- AUDIO_FORMAT_IMA4
- AUDIO_FORMAT_LINEAR_PCM
- AUDIO_FORMAT_ULAW

ボリューム・録音レベルに関するプロパティ

- Ti.Media.volume
- Ti.Media.peakMicrophonePower
- Ti.Media.averageMicrophonePower

関連する API ドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.Media

API カタログ(デバイスハードウェア編) - 電源状態

バッテリ容量・充電状況についての状態を取得するプロパティとその変化を通知 するイベントがあります。

充電状況

Titanium.Platform.BATTERY_STATE_UNKNOWN	不明
Titanium.Platform.BATTERY_STATE_UNPLUGGED	放電中
Titanium.Platform.BATTERY_STATE_CHARGING	充電中
Titanium.Platform.BATTERY_STATE_FULL	フル

コード例・解説

```
// 電源状態の取得
var bs = Titanium.Platform.batteryState;
// バッテリ残量の取得
var bl = Titanium.Platform.batteryLevel;
// バッテリ状況変化を検知するイベント
Titanium.Platform.addEventListener('battery', function(e){
  // 状態と残量が以下のように返されます。
  var state = e.state;
  var level = e.level;
});
```

関連する API ドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.Platform

API カタログ(デバイスハードウェア編) - 加速度センサ

Titanium では加速度センサについての詳細情報を取得するための低級 API とスマートにそれらの情報を扱えるようにイベント化された高級 API が提供されています。厳密な傾き判定などを行う場合は低級 API が必要になって きますが、「縦横の向きが変わった」ことや「シェイクしている」といったことを判断するだけなら高級 API だけで事足ります。

高級API

提供されているイベントは"shake" と"orientationchange"の二つだけです。

イベント名	説明
shake	２～３回デバイスを振る とイベントが発生します。
orientationchange	縦横の向きを変えるとイベントが発生します。

```
// shake イベント
Ti.Gesture.addEventListener("shake", function(){
  var dialog = Titanium.UI.createAlertDialog();
  dialog.setTitle('shake event');
  dialog.setMessage('鮭じゃないよ、シェイクだよ。');
  dialog.setButtonNames(['OK']);
  dialog.show();
});
// orientationchange イベント
Ti.Gesture.addEventListener('orientationchange',function(e) {
  // イベントから取得する場合は次のように取得
  var o = e.orientation;
  // デバイスの状態を取るには次のように
  var o2 = Titanium.Gesture.orientation;
  /*
   状態の種類は次のとおり
   ・Titanium.UI.PORTRAIT
   ・Titanium.UI.UPSIDE_PORTRAIT
   ・Titanium.UI.LANDSCAPE_LEFT
   ・Titanium.UI.LANDSCAPE_RIGHT
   ・Titanium.UI.FACE_UP
   ・Titanium.UI.FACE_DOWN
   ・Titanium.UI.UNKNOWN
  */
});
```

画面の方向は強制的に変更も可能です。

```
Titanium.UI.orientation = Titanium.UI.LANDSCAPE_LEFT;
```

低級API

x, y, z 軸の情報を取得する API です。

取得自体は次のようなコードで随時イベントが発生する感じです。

```
Ti.Accelerometer.addEventListener('update',function(e){
  document.getElementById('x').innerHTML = e.x;
  document.getElementById('y').innerHTML = e.y;
  document.getElementById('z').innerHTML = e.z
});
```

XYZ の軸に関する状態について口で説明するよりもこれを見たほうが一発なので、こちらをご覧下さい。

- Digital Agua's Blog : Accelerometer x,y,z based on iPhone position

- http://blog.digitalagua.com/2008/07/15/accelerometer-xyz-based-on-iphone-position/

関連する APIドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.Gesture
- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.Accelerometer

API カタログ(デバイスハードウェア編) - 位置測定・電子コンパス

利用可能か判断する

位置測定とコンパス機能が利用できるかどうかを判断するプロパティがあります。 まずそちらでイベントリスナへの登録を制御するようにしましょう。

```
// 位置測定機能の有効状態を取得するプロパティ
if(Titanium.Geolocation.locationServicesEnabled){
  // 位置測定機能が必要な処理
}
// コンパスの有効状態を取得するプロパティ(iPhone 3G以前はfalseになる感じ)
if(Titanium.Geolocation.hasCompass){
  // 電子コンパスが必要な処理
}
```

測定方法

GPS、電子コンパスから現在状態を測定するため、2つの方 法が提供されています。

端的に今どこにいるのかを取得する一度きりの処理、もうひとつは方向や場 所の変化を検知し継続的に情報を更新し続ける処理になります。

一度きりの処理 (Titanium.Geolocation.getCurrent)

いかにも値を返しそうなメソッド名なんですが非 同期処理のため、実際の座標取得、エラー処理はコールバック関数で行います。

GPS(Titanium.Geolocation.getCurrentPosition)

```
Titanium.Geolocation.getCurrentPosition(function(e)
  // エラー時はコールバック関数の引数のerrorプロパティがセットされます
  if(e.error){
    Ti.API.error(e.error);
    return;
  }
  // 状態取得時の処理
  var coords = e.corrds;
  /*
  // 中身はこんなデータです。
  {
     // 緯度
     "latitude":37.331689,
     // 経度
```

```
    "longitude":-122.030731,
    // 高度
    "altitude":0,
    // 平面（水平方向）の精度
    "accuracy":100,
    // 垂直方向の精度
    "altitudeAccuracy":-1,
    // 方向
    "heading": -1,
    // 速度
    "speed": -1,
    // 取得時刻
    "timestamp":274737055043
  }
  */
);
```

電子コンパス (Titanium.Geolocation.getCurrentHeading()

```
Titanium.Geolocation.getCurrentHeading(function(e)
  // エラー時はコールバック関数の引数の error プロパティがセットされます
  if(e.error){
    Ti.API.error(e.error);
    return;
  }
  // 状態取得時の処理
  // X-Y-Z 軸の磁束密度（単位：マイクロテスラ）
  var x = e.heading.x;
  var y = e.heading.y;
  var z = e.heading.z;
  // 磁北と真北に対する向き（単位：度）
  var magneticHeading = e.heading.magneticHeading;
  var trueHeading = e.heading.trueHeading;
  // 精度
  var accuracy = e.heading.accuracy;
  // 取得時刻
  var timestamp = e.heading.timestamp;
);
```

継続検知するイベント (Titanium.Geolocation.addEventListener)

イベントのコールバック関数になるだけで、一度きりの取得と内容的には同じになります。

GPS(location イベント)

```
Titanium.Geolocation.addEventListener("location", function(e)
  // エラー時はコールバック関数の引数の error プロパティがセットされます
  if(e.error){
    Ti.API.error(e.error);
    return;
```

```
  }
  // 状態取得時の処理
  var coords = e.corrds;
  /*
  // 中身はこんなデータです。
  {
     // 緯度
     "latitude":37.331689,
     // 経度
     "longitude":-122.030731,
     // 高度
     "altitude":0,
     // 平面（水平方向）の精度
     "accuracy":100,
     // 垂直方向の精度
     "altitudeAccuracy":-1,
     // 方向
     "heading": -1,
     // 速度
     "speed": -1,
     // 取得時刻
     "timestamp":274737055043
  }
  */
);
```

電子コンパス(heading イベント)

```
Titanium.Geolocation.addEventListener("heading", function(e)
  // エラー時はコールバック関数の引数の error プロパティがセットされます
  if(e.error){
    Ti.API.error(e.error);
    return;
  }
  // 状態取得時の処理
  // X-Y-Z 軸の磁束密度（単位：マイクロテスラ）
  var x = e.heading.x;
  var y = e.heading.y;
  var z = e.heading.z;
  // 磁北と真北に対する向き（単位：度）
  var magneticHeading = e.heading.magneticHeading;
  var trueHeading = e.heading.trueHeading;
  // 精度
  var accuracy = e.heading.accuracy;
  // 取得時刻
  var timestamp = e.heading.timestamp;
);
```

その他

磁場干渉があった場合のメッセージ表示を行うかどうか設定できま す。

```
// これはオフにする
Titanium.Geolocation.showCalibration = false;
```

電子コンパスの方向検知をする際にどの程度変化があったらイベントを発生させるかを設定します。（方向検知の「遊び」ですね）

```
// 90度以上方向を変えないとイベントが発生しません。
Titanium.Geolocation.headingFilter = 90;
```

同様にGPSで移動検知イベントを発生させる距離をメートル単位で指定します。

```
// 10m移動するごとにイベント発生する。
Titanium.Geolocation.distanceFilter = 10;

// 精度についての設定です。（追記予定）
//  SET ACCURACY - THE FOLLOWING VALUES ARE SUPPORTED
// Titanium.Geolocation.ACCURACY_BEST
// Titanium.Geolocation.ACCURACY_NEAREST_TEN_METERS
// Titanium.Geolocation.ACCURACY_HUNDRED_METERS
// Titanium.Geolocation.ACCURACY_KILOMETER
// Titanium.Geolocation.ACCURACY_THREE_KILOMETERS
//
Titanium.Geolocation.accuracy = Titanium.Geolocation.ACCURACY_BEST;
```

関連するAPIドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.Geolocation

API カタログ(プラットフォーム編) - アプリケーションバッジ (iPhone のみ)

ア プリケーションバッジとは以下の図の右についている状態のものを指します。

表示できるのはタブバッジとは異なり数値のみとなりますので、未読数などを表示 する以外になかなか使い道がなさそうです。


```
// 20 と表示する。（ホーム画面に戻らないと確認できない）
Titanium.UI.iPhone.appBadge = 20;
// バッジを外す
Titanium.UI.iPhone.appBadge = null;
```

関連する APIドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.UI.iPhone

API カタログ(プラットフォーム編) - 環境情報取得

デバイスの OS や解像度などの情報は Titanium.Platform のプロパティとして取得できます。

プロパティ名	型	説 明
address	string	IP アドレス
architecture	string	CPU のアーキテク チャ(ARM など)

availableMemory	double	使用可能なメモリ
batteryLevel	float	バッテリレベル→詳 細
batteryMonitoring	boolean	バッテリ監視状態→詳 細
batteryState	int	バッテリ状態→詳 細
displayCaps.density	string	the density property of the display device.
displayCaps.dpi	int	ディスプレイの DPI
displayCaps.platformHeight	float	スクリーンの高さ
displayCaps.platformWidth	float	スクリーンの幅
id	string	the unique id of the device
locale	string	ユーザが設定している言語
macaddress	string	MAC アドレス
model	string	デバイスのモデル (iPhone 3G とか 3GS とか)
name	string	デバイス名(iPhone とか)
osname	string	OS 名(iPhone な ら"iPhone", iPad なら"iPad", Android なら"android")
ostype	string	OS のアーキテク チャ(32bit など)
processorCount	int	プロセッサ数
username	string	デバイスに設定されてい るユーザ名
version	string	デバイスプラットフォームのバージョン(3.1.3 など)

またアプリケーションに関する情報は `Titanium.App` の プロパティとして取得できます。

プロパティ名	**型**	**説 明**
copyright	string	著作権者
description	string	アプリケーションの説明
guid	string	GUID
id	string	アプリケーション ID
idleTimerDisabled	boolean	property for controlling whether the phone screen will be locked on idle time. Can be set to true to disable the timer
name	string	アプリケーション名
proximityDetection	boolean	a boolean to indicate whether proximity detection is enabled
proximityState	int	the state of the device's proximity detector
publisher	string	アプリケーションの発行 者
url	string	アプリケーションに関する URL

version	string	アプリケーションのバー ジョン

関連する API ドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.Platform
- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.App

API カタログ(プラットフォーム編) - 他アプリケーション連携(OpenURL)

SMS/MMS

SMS/MMS アプリケーションの起動ですが、残念ながら個人（ひとり）への送信しか対応していません。カンマを入れた りしようとしても「カンマを含んだひとつのアドレス」として認識される iPhone OS 側の制限でできないようになっています。 spam 対策でしょうから致し方ありませんね。

```
// 電話番号 080-1234-5678 なひとに SMS を送る（実在していたらすいません）
Titanium.Platform.openURL('sms:08012345678');
```

電話

これも単独宛先になります。

```
// 電話番号 080-1234-5678 なひとに電話をかける（実在していたらすいません）
Titanium.Platform.openURL('tel:08012345678');
```

Web ページを Safari で開く

これもまあ想像通りだと思いますが。

```
// Google にアクセス！
Titanium.Platform.openURL('http://www.google.co.jp/');
```

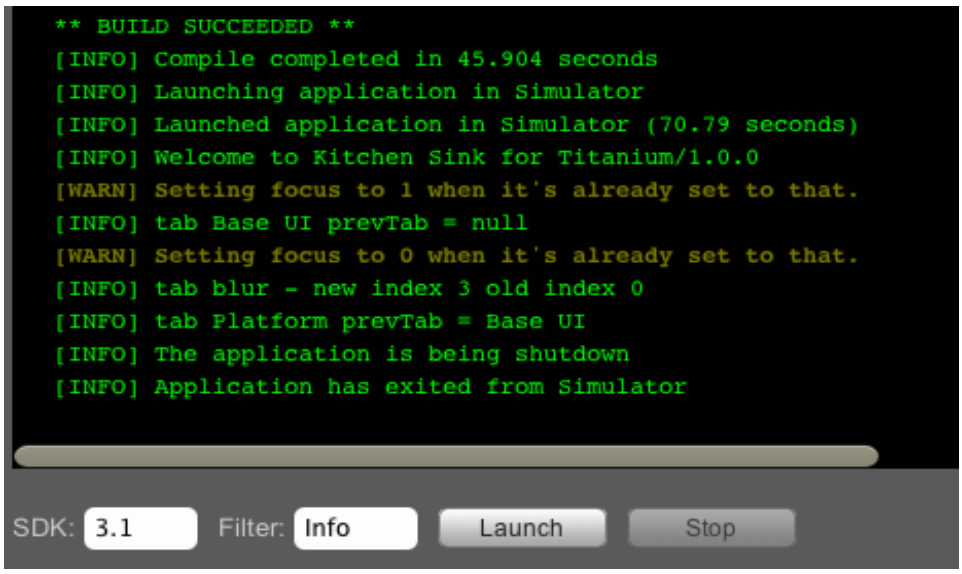
関連する API ドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.Platform.openURL

API カタログ(ユーティリティ編) - ログ出力

その名の通り、ログ出力します。

シミュレータ実行時にこんな表示がされるあれです。



ログ出力系フレームワークをお使いならおなじみだと思 いますが、ログレベルという概念があり、設定したログレベル以上のものに絞り込まれて表示や出力がされます。

高	error
↑	warn
↓	info
低	debug

たとえば、info に設定していると debug レベルの内容は表示されません。

これという 明確な線引きはありませんが、致命的なものが error、警告レベルが warn という感じでしょうか。

```
Ti.API.error('致命的なエラー');
Ti.API.warn('警告');
Ti.API.info('情報');
Ti.API.debug('デバッグ情報');
```

関連する API ドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.API

API カタログ(ユーティリティ編) - タイマー処理

Titanium には `JavaScript` の `setTimeout` と `setInterval` を 使って、タイマー処理をします。

`setTimeout` は 指定した時間後に処理を実行させるのに対し、`setInterval` は一定間隔で処理を繰り返し実行します。

```
//setTimeout の例
var labelTimeout = Ti.UI.createLabel({
  text: 'setTimeout',
  textAlign:'center',
  width:'auto',
  height: 20,
  top:20
});
Ti.UI.currentWindow.add(labelTimeout);
// 3000ms 後に描画する。
setTimeout(function(){
    labelTimeout.text = "3 sec timer fired!!";
}, 3000);

// setInterval の例
var count = 0;
var labelInterval = Ti.UI.createLabel({
  text: '0',
  textAlign:'center',
  width:'auto',
  height: 20,
  top:100
});
Ti.UI.currentWindow.add(labelInterval);
// 10ms ごとにカウントアップし、描画する。
setInterval(function(){
  count++;
  labelInterval.text = "Interval fired " + count;
}, 10);
```

API カタログ(ユーティリティ編) - カスタムイベント

カスタムイベントを設定し、それを任意のタイミングで fire することにより 多彩な処理を行う事ができます。

以下では app.js で宣言したメッセージ表示用イベントをグローバル関数的に使う手法をサンプルに挙げます。

```
//-------------------------------------------
// app.js
//-------------------------------------------
//メッセージ表示用の Window - View - Label のセットを作っておく
var messageWin = Titanium.UI.createWindow({
    height:30,
    width:250,
    bottom:70,
    borderRadius:10,
    touchEnabled:false
});
var messageView = Titanium.UI.createView({
    height:30,
    width:250,
    borderRadius:10,
```

```
    backgroundColor:'#000',
    opacity:0.7,
    touchEnabled:false
});
var messageLabel = Titanium.UI.createLabel({
    text:'',
    color:'#fff',
    width:250,
    height:'auto',
    font:{
        fontFamily:'Helvetica Neue',
        fontSize:13
    },
    textAlign:'center'
});
messageWin.add(messageView);
messageWin.add(messageLabel);
// カスタムイベント「show_message」を設定しておきます。
Titanium.App.addEventListener('show_message', function(e){
  // 内容としてはメッセージ表示して 1 秒後に close するというシンプルなものです
  messageLabel.text = e.message;
  messageWin.open();
  setTimeout(function(){
    messageWin.close({opacity:0,duration:500});
  },1000);
});
```

Titanium.App に対するイベントリスナの登録なので、グローバルに 適用されます。次のボタンをクリックすると「サンプル」というメッセージが表示される寸法です。

```
// sample.js
var b = Titanium.UI.createButton({
  title:'Fire Event ',
  width:200,
  height:40,
  top:60
});
b.addEventListener('click', function(){
  Titanium.App.fireEvent('show_message', {message: 'サンプル', arg1: 'hogehoge'});
});
```

関連する API ドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.App

API 案内 - ユーティリティ編 - 外部 JavaScript 取込み

配列処理をスマートに実現したり、あるいは毎度同じように書 いている twitter への post 処理などをライブラリ化して読み込ませたいといったことも多々あると思います。

C における `#include` のような機能 が `Titanium.include` 関数で実現します。

読み込まれた js ファイルは同じコンテキストで処理されるた め、外部ファイル側で定義された変数などがそ

のまま使えます。

```
// 複数の js ファイルを "," 区切りで列挙できる
Titanium.include('../my_js_include.js', '../my_js_include_2.js', 'local_include.js');
// 中で使われているそれぞれの変数は外部ファイルで定義されているものと考えてください
Ti.UI.createAlertDialog({
  title:'JS Includes',
  message:'first name: ' + myFirstName + ' middle name: ' + myMiddleName +' last name: '
+ myLastName
}).show();
```

Titanium.include は使用したコンテキスト に対して指定したスクリプトを当てはめて評価されるため、同じ名前のオブジェクト(function や class も含む)はすべて上書きされます。

意図しない挙動の原因にもなるので名前空間の汚染をしない ように常々心がけ、また命名には細心の注意を払う必要があるでしょう。

API カタログ(ユーティリティ編) - XML DOM Parser

XML ド キュメントは HTTPClient の responseXML として取得するか、Ti.XML.parseString(str)することで DOM オブジェ クトとして操作することができます。

一般的な DOM オブジェクトとしての操作が可能です。

https://developer.mozilla.org/ja/Gecko_DOM_Reference

```
var xmlstr2 = "<?xml version=\"1.0\" encoding=\"utf-8\"?>"+
"<FooBarResponse>"+
"<FooBarResult>"+
"<ResponseStatus>"+
"<Status>"+
"<PassFail>Pass</PassFail>"+
"<ErrorCode />"+
"<MessageDetail />"+
"</Status>"+
"</ResponseStatus>"+
"<FooBar>true</FooBar>"+
"</FooBarResult>"+
"</FooBarResponse>";
var xml2 = Ti.XML.parseString(xmlstr2);
fooBarList = xml2.documentElement.getElementsByTagName("FooBar");
result = fooBarList!=null && fooBarList.length == 1 && fooBarList.item(0).text=="true";
result = result && fooBarList.item(0).nodeName=="FooBar";
if (xml2.evaluate) {
    // test XPath against Document
    result2 = xml2.evaluate("//FooBar/text()");
    result = result && result2.item(0).nodeValue == "true";

    // test XPath against Element
    result2 = xml2.documentElement.evaluate("//FooBar/text()");
    result = result && result2.item(0).text == "true";

    // test XPath against Element
    result2 = fooBarList.item(0).evaluate("text()");
    result = result && result2.item(0).text == "true";
} else {
    result = false;
```

```
}
```

関連する APIドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.XML
- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.XML.DOMDocument

API カタログ(ユーティリティ編) - 文字列変換

引数に文字列を渡すと、規定の変換処理をする関数がいくつか用意されています。

Titanium.Utils に属するもの

BASE64 のエンコード・デコード、MD5 のハッシュ作成が用意されています。

- base64decode
- base64encode
- md5HexDigest

Titanium.Network に属するもの

URI エンコード・デコードを行う関数が用意されています。

- decodeURIComponent
- encodeURIComponent

関連する APIドキュメント

- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.Utils
- https://developer.appcelerator.com/apidoc/mobile/1.0/Titanium.Network

4th Step : アプリケーション設定

tiapp.xml, manifest について

tiapp.xml

tiapp.xml ファイル はプロジェクトルートにあり、mafinest ファイルと並んで、アプリケーションに関する情報を保存するためのファイルとなっています。

```
<?xml version="1.0" encoding="UTF-8"?>
<ti:app xmlns:ti="http://ti.appcelerator.org">
 <id>jp.hsj.test100</id>
 <name>Test100</name>
 <version>1.0</version>
 <publisher>donayama</publisher>
 <url>http://twitter.com/donayama/</url>
 <description>No description provided</description>
 <copyright>2010 by donayama</copyright>
 <icon>appicon.png</icon>
 <persistent-wifi>false</persistent-wifi>
 <prerendered-icon>false</prerendered-icon>
 <statusbar-style>default</statusbar-style>
 <statusbar-hidden>false</statusbar-hidden>
 <analytics>true</analytics>
 <guid>10eef80833b94e88ba0a455981620606</guid>
</ti:app>
```

要素名	説明
id	アプリケーションの ID(App ID)
name	アプリケーション名
version	アプリケーションのバージョン
publisher	アプリケーション配布者 （開発者）名
url	会社や個人の URL を設定します
description	アプリケーションの説明
copyright	著作権表示
icon	アプリケーションアイコ ンのパスを指定します。
persistent-wifi	WIFI 接続を必須とするか(true/false)
statusBarStyle	最上段のステータスバー の表示形式を指定します。選択は grey (デフォルト), opaque_black , translucent_black の 3 種類。
statusbar-hidden	ステータスバーを非表示 にする場合 true を指定する。

analytics	分析機能を有効にする場合 true とします。
guid	自動生成される guid です。基本的に手をつけません。

manifest

このファイルは Titanium Developer 上で設定されたプロジェクトの情報を保存しておくファイルであり、基本的に変更することはありません。

```
#appname: Test100
#publisher: donayama
#url: http://twitter.com/donayama/
#image: appicon.png
#appid: jp.hsj.test100
#desc: undefined
#type: mobile
#guid: 1bef9b04-0362-4bfb-a68a-1a3635d5734a
```

起動時画像(スプラッシュスクリーン)の変更方法

プロジェクト生成時にも少し触れましたが、project/Resorces/iphone フォルダにある Default.png というファイルが起動時にされる画像になります。お好きな画像に差し替えると気分が変わりますし…そもそもブランディングの観点からも差替えないとだめだと思います(^^;

ちなみに 画面中央部にはローディングインジケータが表示されるので干渉しないような画像にしておくと少し幸せになれると思いますよ。

ち なみにこのファイルを消すとどうなるのかというと、真っ暗な画面にローディングインジケータが表示されます。

一度でもデフォルトの画像が ある状態でコンパイルしていると、/build/iPhone/Resources/Default.png が残ってしまうので、意図的にこうしたい場合 は削除する必要があります。