DynamicApp JavaScript API Document

Ver.1.2

改編履歴

Version	内容	ページ	更新日付
1.0	新規作成	全て	2012/07/31
1.1	新規APIを追加 ・AddressBook ・Bluetooth ・Bluetooth4LE ・Peripheral ・Felica	P.59～P.86	2012/10/08
	新規APIを一覧に追加	P.3	2012/10/08
1.1.1	新規APIを追加 ・AppVersion	P.87	2012/10/15
	新規APIを一覧に追加	P.4	2012/10/15
1.2	新規APIを追加 ・Contacts ・PDFViewer ・Ad	P.89～P.104	2013/02/15
	新規APIを一覧に追加	P.4	2013/02/15
	Camera APIに動画撮影機能を追加	P.34, P.36	2013/02/15

目次

DynamicApp JavaScript API 一覧

No.	オブジェクト	概要
1.	Encryption	文字列の暗号化/復号化機能を提供します。
2.	File	ファイル操作（コピー、移動、削除）の機能を提供します
3.	FileReader	ファイル読み込み機能を提供します
4.	FileWriter	ファイル書き込み機能を提供します
5.	Cache	アプリケーションデータやリソースのキャッシュ機能を提供します
6.	Notification	デバイスのローカル通知機能を提供します
7.	Camera	デバイスのカメラ機能を提供します
8.	Sound	音声再生機能を提供します
9.	Movie	動画再生機能を提供します
10.	QRReader	QRリーダー機能を提供します
11.	LoadingScreen	Loading screen画面を表示します
12.	Database	データベース機能を提供します
13.	AddressBook	電話帳UIを表示します
14.	Bluetooth	Bluetoothによる通信をサポートします
15.	Bluetooth4LE	Bluetooth4LE対応デバイスとの通信をサポートします （※iOSのみ、ただしiPhone以前のiPhone、iPad2以前のiPadではご利用いただけません）
16.	Peripheral	Bluetooth4LEデバイスを表します （※iOSのみ、ただしiPhone以前のiPhone、iPad2以前のiPadではご利用いただけません）

No.	オブジェクト	概要
17.	Felica	Felicaカードへのリード/ライト機能を提供します （※Androidのみ）
18.	AppVersion	アプリバージョンを表示します
19.	Contacts	デバイス内の電話帳データ操作をサポートします
20.	PDFViewer	PDFビューワ機能を提供します （※iOSのみ）
21.	Ad	広告機能を提供します

Encryption

メソッド:
- encrypt
- decrypt

encrypt

文字列を暗号化します

```
Encryption.encrypt(string, successCallback, errorCallback);
```

string

暗号化対象の文字列

successCallback

暗号化が正常終了した場合に呼び出されます

```
function(encryptedString) {
    // Do something with the encryptedString.
}
```

パラメータ
- encryptedString : 暗号化された文字列

errorCallback

暗号化に失敗した際に呼び出されれます

```
function(message) {
    // Show a helpful message.
}
```

パラメータ

・message：ネイティブ側からのエラーメッセージ

decrypt

文字列の復号化を行います。

```
Encryption.decrypt(encryptedString, successCallback, errorCallback);
```

encryptedString

暗号化されている文字列

successCallback

復号化が正常終了した場合に呼び出されます

```
function(decryptedString) {
    // Do something with the decryptedString.
}
```

パラメータ
・decryptedString：復号化された文字列

errorCallback

復号化に失敗した場合に呼び出されます

```
function(message) {
    // Show a helpful message.
}
```

パラメータ
・message：ネイティブ側からのエラーメッセージ

File

プロパティ:
- name : ファイル名
- fullpath : フルパス
- type : ファイルタイプ
- size : ファイルサイズ

メソッド:
- コンストラクタ
- isFile
- isDirectory
- copy
- move
- remove

コンストラクタ

```
var file = new File(parentPath,  filename, successCallback, errrorCallback);
```

parentPath

このファイルが存在する親のパス

filename

ファイル名

successCallback

ファイルオブジェクトが正常に作成された場合に呼び出されます

```
function(metadata) {
    // Do something with the metadata.
}
```

パラメータ

・metadata：ファイルメタデータ

errorCallback

ファイルオブジェクト作成に失敗した場合に呼び出されます

```
function(errorCode) {
    // Do something in case of error.
}
```

パラメータ

・errorCode：エラーコード
エラーコードの詳細については、P.16 エラーコードを参照のこと

isFile

このファイルオブジェクトがファイルかどうかを返します

```
file.isFile();
```

isDirectory

このファイルオブジェクトがディレクトリかどうかを返します

```
file.isDirectory();
```

copy

ファイルのコピーを行います

```
file.copy(targetDirectory, newName, successCallback, errorCallback);
```

targetDirectory

コピー先ディレクトリ

successCallback

ファイルコピーが正常終了した場合に呼び出されます

```
function() {
    // Do something
}
```

errorCallback

ファイルコピーに失敗した場合に呼び出されます

```
function(errorCode) {
    // Do something in case of error.
}
```

パラメータ

・errorCode：エラーコード
エラーコードの詳細については、P.16 エラーコードを参照のこと

move

ファイルの移動を行います

```
file.move(targetDirectory, newName, successCallback, errorCallback);
```

targetDirectory

移動先ディレクトリ

newName

新しいファイル名

successCallback

ファイル移動が正常終了した場合に呼び出されます

```
function() {
    // Do something
}
```

errorCallback

ファイル移動に失敗した場合に呼び出されます

```
function(errorCode) {
    // Do something in case of error.
}
```

パラメータ

・errorCode：エラーコード
エラーコードの詳細については、P.16 エラーコードを参照のこと

remove

ファイルを削除します

```
file.remove(successCallback, errorCallback);
```

successCallback

ファイル削除が正常終了した場合に呼び出されます

```
function() {
    // Do something
}
```

errorCallback

ファイル削除に失敗した場合に呼び出されます

```
function(errorCode) {
    // Do something in case of error.
}
```

パラメータ

・errorCode：エラーコード
エラーコードの詳細については、P.16 エラーコードを参照のこと

エラーコード

エラーコード一覧

```
File.ERROR_NOT_FOUND = 1;
File.ERROR_SECURITY = 2;
File.ERROR_ABORT = 3;
File.ERROR_NOT_READABLE = 4;
File.ERROR_ENCODING = 5;
File.ERROR_NO_MODIFICATION_ALLOWED = 6;
File.ERROR_INVALID_STATE = 7;
File.ERROR_SYNTAX = 8;
File.ERROR_INVALID_MODIFICATION = 9;
File.ERROR_QUOTA_EXCEEDED = 10;
File.ERROR_TYPE_MISMATCH = 11;
File.ERROR_PATH_EXISTS = 12;
```

FileReader

メソッド:
- コンストラクタ
- read

コンストラクタ

```
var reader = new FileReader(file);
```

file

読み込み対象のファイルオブジェクト

read

読み込みを行います

```
reader.read(encoding, successCallback, errorCallback);
```

encoding

ファイルで使われているエンコーディング

successCallback

読み込みが正常終了した場合に呼び出されます

```
function(readData) {
    // Do something with the data.
}
```

パラメータ

・readData: ファイルから読み込んだ内容

errorCallback

読み込み時にエラーが発生した場合に呼び出されます

```
function(errorCode) {
    // Do something in case of error.
}
```

パラメータ

・errorCode : エラーコード

エラーコードの詳細については、P.16 エラーコードを参照のこと

FileWriter

プロパティ:
- position : ファイルポインタの現在位置

メソッド:
- コンストラクタ
- write
- seek

コンストラクタ

```
var writer = new FileWriter(file);
```

file

書き込み対象のファイルオブジェクト

write

書き込みを行います

```
writer.write(writeData, successCallback, errorCallback);
```

writeData

書き込みデータ

successCallback

書き込みが正常終了した場合に呼び出されます

```
function() {
    // Do something.
}
```

errorCallback

読み込み時にエラーが発生した場合に呼び出されます

```
function(errorCode) {
    // Do something in case of error.
}
```

パラメータ

・errorCode：エラーコード
　エラーコードの詳細については、P.16 エラーコードを参照のこと

seek

ファイルポインタを指定された位置に移動します。（バイト単位）

```
writer.seek(pos);
```

pos

ファイルポインタを移動させる位置

Cache

メソッド:

- addData
- getList
- removeData
- resetDataCache
- addResource
- updateResource
- getResourceList
- removeResource
- resetResourceCache

addData

データキャッシュにデータを追加します

```
Cache.addData(data, expiredDate, successCallback, errorCallback);
```

data

追加するデータ

expiredDate

有効期限（yyyy-MM-dd HH:mm:ss形式）

successCallback

データのキャッシュ保存が正常終了した場合に呼び出されます

```
function() {
    // Do something.
}
```

errorCallback

データ保存失敗時に呼び出されます

```
function() {
    // Do something in case of error.
}
```

getList

キャッシュに保存されているデータリストを返します

```
Cache.getList();
```

removeData

キャッシュからデータを削除します

```
Cache.removeData(id, successCallback, errorCallback);
```

id

データID

successCallback

データ削除が正常終了した場合に呼び出されます

```
function() {
    // Do something.
}
```

errorCallback

データ削除失敗時に呼び出されます

```
function() {
    // Do something in case of error.
}
```

resetDataCache

データキャッシュをクリアします

```
Cache.resetDataCache(successCallback, errorCallback);
```

successCallback

クリアが正常終了した場合に呼び出されます

```
function() {
    // Do something.
}
```

errorCallback

クリアに失敗した際に呼び出されます

```
function() {
    // Do something in case of error.
}
```

addResource

リソース（URL）データをキャッシュに追加します。
追加されたデータURLからリソースのダウンロードを行います。

```
Cache.addResource(remoteContentsPath, expiredDate, successCallback, errorCallback);
```

remoteContentsPath

リモートコンテンツパス

expiredDate

有効期限（yyyy-MM-dd HH:mm:ss形式）

successCallback

データ追加が正常終了した場合に呼び出されます

```
function(result) {
    // Do something with result.
}
```

パラメータ

・result：結果情報

result.id：ダウンロードしたデータID
result.expireDate：データの有効期限
result.fullpath：ダウンロードデータの保存先

errorCallback

データ追加失敗時に呼び出されます

```
function() {
    // Do something in case of error.
}
```

updateResource

リソースデータを更新します（ダウンロードも含む）

```
Cache.updateResource(id, nextexpiredDate, successCallback, errorCallback);
```

id

データID

nextexpiredDate

次の有効期限（yyyy-MM-dd HH:mm:ss形式）

successCallback

更新が正常終了した場合に呼び出されます

```
function(result) {
    // Do something.
}
```

パラメータ

・result：結果情報
result.id：ダウンロードしたデータID
result.expireDate：データの有効期限
result.fullpath：ダウンロードデータの保存先

errorCallback

更新失敗時に呼び出されます

```
function() {
    // Do something in case of error.
}
```

getResourceList

データリストを取得します

```
Cache.getResourceList();
```

removeResource

キャッシュからリソースデータを削除します

```
Cache.removeResource(id, successCallback, errorCallback);
```

id

データID

successCallback

データ削除が正常終了した場合に呼び出されます

```
function() {
    // Do something.
}
```

errorCallback

データ削除時にエラーが発生した場合に呼び出されます

```
function(errorCode) {
    // Do something in case of error.
}
```

resetResourceCache

リソースキャッシュをクリアします

```
Cache.resetResourceCache(successCallback, errorCallback);
```

successCallback

キャッシュクリアが正常終了した場合に呼び出されます

```
function() {
    // Do something.
}
```

errorCallback

キャッシュクリア時にエラーが発生した場合に呼び出されます（主にファイル削除失敗）

```
function(errorCode) {
    // Do something in case of error.
}
```

Notification

メソッド:
- notify
- cancelNotification

notify

指定された時間に、指定されたメッセージを通知します（ローカル通知）

```
Notification.notify( date, message, [options] );
```

date

通知を行う日付（yyyy-MM-dd HH:mm:ss形式）

message

通知メッセージ

options

通知関係のオプションパラメータ（iOSのみ有効）

```
{ badge : 1,
  hasAction : true,
  action : “起動” };
```

オプション

・budge: アプリケーションアイコンに表示するバッジ数
・hasAction: アクションボタンを表示するかどうか
・action: アクションボタンのタイトル

cancelNotification

通知をキャンセルします

```
Notification.cancelNotification(date, successCallback, errorCallback);
```

date

キャンセルする通知日付

successCallback

キャンセルが正常に行われた場合に呼び出されます

```
function() {
    // Do something.
}
```

errorCallback

キャンセルに失敗した場合に呼び出されます

```
function() {
    // Do something in case of error.
}
```

Camera

メソッド:
- getPicture
- recordVideo

getPicture

カメラ、もしくはライブラリから画像を取得します。
画像のBase64エンコード文字列か画像ファイルのURIを返します

```
Camera.getPicture( successCallback, errorCallback, [ options ] );
```

successCallback

画像の取得が正常に行われた場合に呼び出されます

```
function(imageData) {
    // Do something with the image.
}
```

パラメータ
- imageData: 画像データ（Base64エンコード文字列）もしくは、画像ファイルのURI

errorCallback

画像取得に失敗した場合に呼び出されます

```
function(message) {
    // Show a helpful message.
}
```

パラメータ

・message: ネイティブ側からのエラーメッセージ

options

カスタマイズオプション

```
{ quality : 75,
  destinationType : Camera.DestinationType.DATA_URL,
  sourceType : Camera.PictureSourceType.CAMERA,
  encodingType: Camera.EncodingType.JPEG,
  targetWidth: 100,
  targetHeight: 100 };
```

オプション

・quality: 画質(0~100)
・destinationType: 戻り値のフォーマット
・sourceType: 画像ソース
・encodingType: 画像のエンコードタイプ
・targetWidth: 取得画像の幅
・targetHeight: 取得画像の高さ

recordVideo

動画の撮影を行います。

```
Camera.recordVideo(successCallback, errorCallback);
```

successCallback

動画撮影が正常に行われた場合に呼び出されます

```
function(videoData) {
    // Do something with the videoData.
}
```

パラメータ

・videoData: 撮影した動画ファイルへのURI

errorCallback

動画撮影に失敗した場合に呼び出されます

```
function(message) {
    // Show a helpful message.
}
```

パラメータ

・message: ネイティブ側からのエラーメッセージ

Sound

メソッド:
- コンストラクタ
- play
- pause
- stop
- release
- getCurrentPosition
- setCurrentPosition
- getDuration

コンストラクタ

```
var sound = new Sound(src, [successCallback], [errorCallback] );
```

src

サウンドコンテンツへのURI

successCallback

再生、一時停止、停止など全てのアクションが正常に終了した場合に呼び出されます

```
function() {
    // Do something
}
```

errorCallback

再生、一時停止、停止など全てのアクションに於いてエラーが発生した場合に呼び出されます

```
function(errorCode) {
    // Do something in case of error.
}
```

パラメータ

・errorCode: エラーコード

SOUND_ERR_ABORTED = 1
SOUND_ERR_NETWORK = 2
SOUND_ERR_DECODE = 3
SOUND_ERR_NONE_SUPPORTED = 4

play

サウンドを再生します

```
sound.play([numberOfLoops:] );
```

numberOfLoops

ループ回数

pause

サウンドの再生を一時停止します

```
sound.pause( );
```

stop

サウンドの再生を停止します

```
sound.stop( );
```

release

このオブジェクトで使用していたサウンドリソースを解放します

```
sound.release();
```

getCurrentPosition

現在の再生位置を取得します

```
sound.getCurrentPosition();
```

setCurrentPosition

現在の再生位置（秒）を設定します

```
sound.setCurrentPosition(pos, [func]);
```

pos

再生位置（秒）

func

再生位置変更後に行うファンクション

getDuration

再生時間を取得します

```
sound.getDuration();
```

Movie

メソッド:
- コンストラクタ
- play
- pause
- stop
- release
- getDuration
- getCurrentPosition
- setCurrentPosition
- getThumbnail

コンストラクタ

```
var movie = new Movie(src, [successCallback], [errorCallback], [options]);
```

src

動画コンテンツへのURI

successCallback

再生、一時停止、停止など全てのアクションが正常に終了した場合に呼び出されます

```
function() {
    // Do something
}
```

errorCallback

再生、一時停止、停止など全てのアクションに於いてエラーが発生した場合に呼び出されます

```
function(errorCode) {
    // Do something in case of error.
}
```

パラメータ

・errorCode：エラーコード

MOVIE_ERR_ABORTED = 1
MOVIE_ERR_NETWORK = 2
MOVIE_ERR_DECODE = 3
MOVIE_ERR_NONE_SUPPORTED = 4

options

ムービープレーヤーオプション

```
{ frame : {posX : 0, posY : 0, width : 100,  height : 100},
  scalingMode : Movie.SCALING_ASPECT_FIT,
  controlStyle : Movie.CONTROL_NONE };
```

オプション

・frame: ムービープレーヤー表示位置、サイズ
・scalingMode: スケーリングモード
・controlStyle: コントロールスタイル

play

ムービーの再生開始を行います

```
movie.play( [options]);
```

options

ムービープレーヤーオプション

```
{ frame : {posX : 0, posY : 0, width : 100,  height : 100},
  scalingMode : Movie.SCALING_ASPECT_FIT,
  controlStyle : Movie.CONTROL_NONE };
```

オプション

- frame: ムービープレーヤー表示位置、サイズ
- scalingMode: スケーリングモード
- controlStyle: コントロールスタイル

pause

ムービーの再生を一時停止します

```
movie.pause( );
```

stop

ムービーの再生を停止します

```
movie.stop( );
```

release

このオブジェクトで使用していたムービーリソースを解放します

```
movie.release();
```

getDuration

ムービーの再生時間を取得します

```
movie.getDuration();
```

getCurrentPosition

現在の再生位置（秒）を取得します

```
movie.getCurrentPosition();
```

setCurrentPosition

現在の再生位置（秒）を設定します

```
movie.setCurrentPosition(pos, [func]);
```

pos

再生位置（秒）

func

再生位置変更後に行うファンクション

getThumbnail

ムービーのサムネイルを取得します

```
movie.getThumbnail(successCallback, errorCallback, [options]);
```

successCallback

サムネイルの取得が正常に行われた場合に呼び出されます

```
function(data) {
    // Do something
}
```

パラメータ

・data: Base64 エンコーディングされたイメージデータ

errorCallback

サムネイル取得時にエラーが発生した場合に呼び出されます

```
function(errorCode) {
    // Do something in case of error.
}
```

パラメータ

・errorCode: エラーコード

MOVIE_ERR_NONE_SUPPORTED = 4

options

サムネイル画像取得に関するオプションを設定します

```
{ quality : 75
  width : 100,
  height : 100,
  offset : 0 };
```

オプション

- quality: サムネイル画像の画質（0~100）
- width: サムネイル画像の幅
- height: サムネイル画像の高さ
- offset: ムービー開始位置からのオフセット（どの位置のサムネイルを取得するか）※iOSのみ

QRReader

メソッド:
- read

read

QRコードを読み込みます

```
QRReader.read(successCallback , errorCallback, [options]);
```

successCallback

QRコード読み込みが正常終了した際に呼び出されます

```
function(dataString) {
    // Do something with the string.
}
```

パラメータ
- dataString: QRコードデータ

errorCallback

QRコード読み込み時に失敗した際に呼び出されます

```
function() {
    // Do something in case of error.
}
```

options

QRコード読み込み時のオプションを設定します

```
{source: QRReader.SOURCE_CAMERA};
```

オプション

・source: どこからQRコードを読み込むか

LoadingScreen

メソッド:
- startLoad
- stopLoad

startLoad

ローディングビューの表示を開始します

```
LoadingScreen.startLoad( successCallback, errorCallback, [ options ] );
```

successCallback

ローディングスクリーン表示が正常に開始された場合に呼び出されます

```
function() {
    // Do something.
}
```

errorCallback

ローディングスクリーン表示時にエラーが発生した場合に呼び出されます

```
function() {
    // Do something in case of error.
}
```

options

表示オプションを設定します

```
{label:  "Loading...",
 style:LoadingScreen.STYLEWHITELARGE,
 isBlackBackground:true};
```

オプション

- label: ローディングスクリーンに表示する文字列
- style: ローディングスクリーンのスタイル
- isBlackBackgroud: バックグラウンドが黒かどうか

stopLoad

ローディングスクリーンの表示を停止します

```
LoadingScreen.stoptLoad(successCallback, errorCallback);
```

successCallback

ローディングスクリーン停止が正常に完了した際に呼び出されます

```
function() {
    // Do something.
}
```

errorCallback

ローディングスクリーン停止時にエラーが発生した場合に呼び出されます

```
function() {
    // Do something in case of error.
}
```

Database

メソッド:
- コンストラクタ
- executeSQL
- beginTrans
- commit
- rollback

コンストラクタ

```
var db = new Database(dbName, successCallback, errorCallback);
```

dbName

データベース名

successCallback

データベースオブジェクトが正常に作成された場合に呼び出されます

```
function() {
    // Do something.
}
```

errorCallback

データベースオブジェクト作成時にエラーが発生した場合に呼び出されます

```
function(errorCode) {
    // Do something in case of error.
}
```

パラメータ

・errorCode: エラーコード
エラーコードの詳細については、P.59 エラーコードを参照のこと

executeSQL

SQLクエリを実行します

```
db.executeSQL(query, args, successCallback, errorCallback);
```

query

クエリ文字列

args

クエリがSELECT文かどうかを指定する
※次期バージョン以降、プリペアードステートメントに対応する予定

successCallback

クエリ実行が正常終了した場合に呼び出されます
※SELECT文の場合、該当データがJSONオブジェクトで返されます

```
function(result) {
    // Do something.
}
```

errorCallback

クエリー実行時に失敗した場合に呼び出されます

```
function(errorCode) {
    // Do something in case of error.
}
```

パラメータ

・errorCode: エラーコード
エラーコードの詳細については、P.59 エラーコードを参照のこと

beginTrans

トランザクションを開始します

```
db.beginTrans(successCallback, errorCallback);
```

successCallback

トランザクション開始が正常終了した場合に呼び出されます

```
function() {
    // Do something.
}
```

errorCallback

トランザクション開始に失敗した場合に呼び出されます

```
function(errorCode) {
    // Do something in case of error.
}
```

パラメータ

・errorCode: エラーコード
エラーコードの詳細については、P.59 エラーコードを参照のこと

commitTrans

トランザクションをコミットします

```
db.commitTrans(successCallback, errorCallback);
```

successCallback

コミットに成功した場合に呼び出されます

```
function() {
    // Do something.
}
```

errorCallback

コミットに失敗した場合に呼び出されます

```
function(errorCode) {
    // Do something in case of error.
}
```

パラメータ

・errorCode: エラーコード
エラーコードの詳細については、P.59 エラーコードを参照のこと

rollback

トランザクションをロールバックします

```
db.rollback(successCallback, errorCallback);
```

successCallback

ロールバックに成功した場合に呼び出されます

```
function() {
    // Do something.
}
```

errorCallback

ロールバックに失敗した場合に呼び出されます

```
function(errorCode) {
    // Do something in case of error.
}
```

パラメータ

・errorCode: エラーコード
エラーコードの詳細については、P.59 エラーコードを参照のこと

エラーコード

エラーコード一覧

```
Database.SQLITE_ERROR       = 1;   // SQL error or missing database
Database.SQLITE_INTERNAL    = 2;   // Internal logic error in SQLite
Database.SQLITE_PERM        = 3;   // Access permission denied
Database.SQLITE_ABORT       = 4;   // Callback routine requested an abort
Database.SQLITE_BUSY        = 5;   // The database file is locked
Database.SQLITE_LOCKED      = 6;   // A table in the database is locked
Database.SQLITE_NOMEM       = 7;   // A malloc() failed
Database.SQLITE_READONLY    = 8;   // Attempt to write a readonly database
Database.SQLITE_INTERRUPT   = 9;   // Operation terminated by sqlite3_interrupt()
Database.SQLITE_IOERR       = 10;  // Some kind of disk I/O error occurred
Database.SQLITE_CORRUPT     = 11;  // The database disk image is malformed
Database.SQLITE_NOTFOUND    = 12;  // Unknown opcode in sqlite3_file_control()
Database.SQLITE_FULL        = 13;  // Insertion failed because database is full
Database.SQLITE_CANTOPEN    = 14;  // Unable to open the database file
Database.SQLITE_PROTOCOL    = 15;  // Database lock protocol error
Database.SQLITE_EMPTY       = 16;  // Database is empty
Database.SQLITE_SCHEMA      = 17;  // The database schema changed
Database.SQLITE_TOOBIG      = 18;  // String or BLOB exceeds size limit
Database.SQLITE_CONSTRAINT  = 19;  // Abort due to constraint violation
Database.SQLITE_MISMATCH    = 20;  // Data type mismatch
Database.SQLITE_MISUSE      = 21;  // Library used incorrectly
Database.SQLITE_NOLFS       = 22;  // Uses OS features not supported on host
Database.SQLITE_AUTH        = 23;  // Authorization denied
Database.SQLITE_FORMAT      = 24;  // Auxiliary database format error
Database.SQLITE_RANGE       = 25;  // 2nd parameter to sqlite3_bind out of range
Database.SQLITE_NOTADB      = 26;  // File opened that is not a database file
Database.SQLITE_ROW         = 100; // sqlite3_step() has another row ready
Database.SQLITE_DONE        = 101; // sqlite3_step() has finished executing
```

AddressBook

メソッド:
- show
- showForSelection

show

電話帳UIを表示します。

```
AddressBook.show(successCallback, errorCallback);
```

successCallback

電話帳UIが正常に表示された場合に呼び出されます。

```
function() {
    // Do something.
}
```

errorCallback

電話帳UI表示時にエラーが発生した場合に呼び出されます。

```
function(message) {
    // Do something in case of error.
}
```

パラメータ
- message: ネイティブ側からのエラーメッセージ

showForSelection

連絡先情報取得UIを表示します。

```
AddressBook.showForSelection(successCallback, errorCallback);
```

successCallback

連絡先が正常に取得された場合に呼び出されます。

```
function(data) {
    // Do something.
}
```

パラメータ

・data: 選択したアドレス情報

errorCallback

連絡先取得時にエラーが発生した場合に呼び出されます。

```
function(message) {
    // Do something in case of error.
}
```

パラメータ

・message: ネイティブ側からのエラーメッセージ

プロパティ:
- peerList
- onRecvCallback

メソッド:
- discover
- connect
- disconnect
- send

discover

デバイスの検索を開始します。

```
Bluetooth.discover();
```

connect

デバイスに接続します。

```
Bluetooth.connect(peerData, successCallback, errorCallback);
```

peerData

デバイス情報

successCallback

接続が正常に取得された場合に呼び出されます。

```
function() {
    // Do something.
}
```

errorCallback

接続に失敗した発生した場合に呼び出されます。

```
function() {
    // Do something in case of error.
}
```

disconnect

デバイスの接続を解除します。

```
Bluetooth.disconnect(peerData, successCallback, errorCallback);
```

peerData

デバイス情報

successCallback

接続解除が正常に取得された場合に呼び出されます。

```
function() {
    // Do something.
}
```

errorCallback

接続解除に失敗した発生した場合に呼び出されます。

```
function() {
    // Do something in case of error.
}
```

send

デバイスへデータ送信を行います。

```
Bluetooth.send(peerData, sendData, successCallback, errorCallback);
```

peerData

デバイス情報

sendData

送信データ（String）

successCallback

正常にデータ送信された場合に呼び出されます。

```
function() {
    // Do something.
}
```

errorCallback

データ送信に失敗した発生した場合に呼び出されます。

```
function() {
    // Do something in case of error.
}
```

※ご注意※
Androidでは現在ご利用頂けません。
iPhone4以前のiPhone、iPad2以前のiPadではご利用頂けません。

定数:

Services

- Bluetooth4LE.SERVICE_ALERT_NOTIFICATION = '1811'
- Bluetooth4LE.SERVICE_BATTERY_SERVICE = '180F'
- Bluetooth4LE.SERVICE_BLOOD_PRESSURE = '1810'
- Bluetooth4LE.SERVICE_CURRENT_TIME = '1805'
- Bluetooth4LE.SERVICE_CYCLINGSPEED_AND_CADENCE = '1816'
- Bluetooth4LE.SERVICE_DEVICE_INFORMATION = '180A'
- Bluetooth4LE.SERVICE_GENERIC_ACCESS = '1800'
- Bluetooth4LE.SERVICE_GENERIC_ATTRIBUTE = '1801'
- Bluetooth4LE.SERVICE_GLUCOSE = '1808'
- Bluetooth4LE.SERVICE_HEALTH_THERMOMETER = '1809'
- Bluetooth4LE.SERVICE_HEART_RATE = '180D'
- Bluetooth4LE.SERVICE_HUMAN_INTERFACE_DEVICE = '1812'
- Bluetooth4LE.SERVICE_IMMEDIATE_ALERT = '1802'
- Bluetooth4LE.SERVICE_HUMAN_LINK_LOSS = '1803'
- Bluetooth4LE.SERVICE_NEXT_DST_CHANGE = '1807'
- Bluetooth4LE.SERVICE_PHONE_ALERT_STATUS = '180E'
- Bluetooth4LE.SERVICE_REFERENCE_TIME_UPDATE = '1806'
- Bluetooth4LE.SERVICE_SCAN_PARAMETERS = '1813'
- Bluetooth4LE.SERVICE_TX_POWER = '1804'

Characteristics

- Bluetooth4LE.CHARACTERISTICS_ALERT_CATEGORY_ID = '2A43'
- Bluetooth4LE.CHARACTERISTICS_ALERT_CATEGORY_ID_BIT_MASK = '2A42'
- Bluetooth4LE.CHARACTERISTICS_ALERT_LEVEL = '2A06'
- Bluetooth4LE.CHARACTERISTICS_ALERT_NOTIFICATION_CONTROL_POINT = '2A44'

・Bluetooth4LE.CHARACTERISTICS_ALERT_STATUS = '2A3F'
・Bluetooth4LE.CHARACTERISTICS_APPEARANCE = '2A01'
・Bluetooth4LE.CHARACTERISTICS_BATTERY_LEVEL = '2A19'
・Bluetooth4LE.CHARACTERISTICS_BLOOD_PRESSURE_FEATURE = '2A49' ・
・Bluetooth4LE.CHARACTERISTICS_BLOOD_PRESSURE_MEASUREMENT = '2A35'
・Bluetooth4LE.CHARACTERISTICS_BODY_SENSOR_LOCATION = '2A38'
・Bluetooth4LE.CHARACTERISTICS_BOOT_KEYBOARD_INPUT_REPORT = '2A22'
・Bluetooth4LE.CHARACTERISTICS_BOOT_KEYBOARD_OUTPUT_REPORT = '2A32'
・Bluetooth4LE.CHARACTERISTICS_BOOT_MOUSE_INPUT_REPORT = '2A33'
・Bluetooth4LE.CHARACTERISTICS_CURRENT_TIME = '2A2B'
・Bluetooth4LE.CHARACTERISTICS_DATE_TIME = '2A08'
・Bluetooth4LE.CHARACTERISTICS_DAY_DATE_TIME = '2A0A'
・Bluetooth4LE.CHARACTERISTICS_DAY_OF_WEEK = '2A09'
・Bluetooth4LE.CHARACTERISTICS_DEVICE_NAME = '2A00'
・Bluetooth4LE.CHARACTERISTICS_DST_OFFSET = '2A0D'
・Bluetooth4LE.CHARACTERISTICS_EXACT_TIME_256 = '2A0C'
・Bluetooth4LE.CHARACTERISTICS_FIRMWARE_REVISION_STRING = '2A26'
・Bluetooth4LE.CHARACTERISTICS_GLUCOSE_FEATURE = '2A51'
・Bluetooth4LE.CHARACTERISTICS_GLUCOSE_MEASUREMENT = '2A18'
・Bluetooth4LE.CHARACTERISTICS_GLUCOSE_MEASUREMENT_CONTEXT = '2A18'
・Bluetooth4LE.CHARACTERISTICS_HARDWARE_REVISION_STRING = '2A27'
・Bluetooth4LE.CHARACTERISTICS_HEART_RATE_CONTROL_POINT = '2A39'
・Bluetooth4LE.CHARACTERISTICS_HEART_RATE_MEASUREMENT = '2A37'
・Bluetooth4LE.CHARACTERISTICS_HID_CONTROL_POINT = '2A4C'
・Bluetooth4LE.CHARACTERISTICS_HID_INFORMATION = '2A4A'
・Bluetooth4LE.CHARACTERISTICS_IEEE_11073-20601_REGULATORY_CERTIFICATION_DATA_LIST = '2A2A'
・Bluetooth4LE.CHARACTERISTICS_INTERMEDIATE_BLOOD_PRESSURE = '2A36'
・Bluetooth4LE.CHARACTERISTICS_INTERMEDIATE_TEMPERATURE = '2A1E'
・Bluetooth4LE.CHARACTERISTICS_LOCAL_TIME_INFORMATION = '2A0F'
・Bluetooth4LE.CHARACTERISTICS_MANUFACTURER_NAME_STRING = '2A29'
・Bluetooth4LE.CHARACTERISTICS_MEASUREMENT_INTERVAL = '2A21'

- Bluetooth4LE.CHARACTERISTICS_MODEL_NUMBER_STRING = '2A24'
- Bluetooth4LE.CHARACTERISTICS_NEW_ALERT = '2A46'
- Bluetooth4LE.CHARACTERISTICS_PERIPHERAL_PREFERRED_CONNECTION_PARAMETERS = '2A04'
- Bluetooth4LE.CHARACTERISTICS_PERIPHERAL_PRIVACY_FLAG = '2A02'
- Bluetooth4LE.CHARACTERISTICS_PNP_ID = '2A50'
- Bluetooth4LE.CHARACTERISTICS_PROTOCOL_MODE = '2A4E'
- Bluetooth4LE.CHARACTERISTICS_RECONNECTION_ADDRESS = '2A03'
- Bluetooth4LE.CHARACTERISTICS_RECORD_ACCESS_CONTROL_POINT = '2A52'
- Bluetooth4LE.CHARACTERISTICS_REFERENCE_TIME_INFORMATION = '2A14'
- Bluetooth4LE.CHARACTERISTICS_REPORT = '2A4D'
- Bluetooth4LE.CHARACTERISTICS_REPORT_MAP = '2A4B'
- Bluetooth4LE.CHARACTERISTICS_RINGER_CONTROL_POINT = '2A40'
- Bluetooth4LE.CHARACTERISTICS_RINGER_SETTING = '2A41'
- Bluetooth4LE.CHARACTERISTICS_SCAN_INTERVAL_WINDOW = '2A4F'
- Bluetooth4LE.CHARACTERISTICS_SCAN_REFRESH = '2A31'
- Bluetooth4LE.CHARACTERISTICS_SERIAL_NUMBER_STRING = '2A25'
- Bluetooth4LE.CHARACTERISTICS_SERVICE_CHANGED = '2A05'
- Bluetooth4LE.CHARACTERISTICS_SOFTWARE_REVISION_STRING = '2A28'
- Bluetooth4LE.CHARACTERISTICS_SUPPORTED_NEW_ALERT_CATEGORY = '2A47'
- Bluetooth4LE.CHARACTERISTICS_SUPPORTED_UNREAD_ALERT_CATEGORY = '2A48'
- Bluetooth4LE.CHARACTERISTICS_SYSTEM_ID = '2A23'
- Bluetooth4LE.CHARACTERISTICS_TEMPERATURE_MEASUREMENT = '2A1C'
- Bluetooth4LE.CHARACTERISTICS_TEMPERATURE_TYPE = '2A1D'
- Bluetooth4LE.CHARACTERISTICS_TIME_ACCURACY = '2A12'
- Bluetooth4LE.CHARACTERISTICS_TIME_SOURCE = '2A13'
- Bluetooth4LE.CHARACTERISTICS_TIME_UPDATE_CONTROL_POINT = '2A16'
- Bluetooth4LE.CHARACTERISTICS_TIME_UPDATE_STATE = '2A17'
- Bluetooth4LE.CHARACTERISTICS_TIME_WITH_DST = '2A11'
- Bluetooth4LE.CHARACTERISTICS_TIME_ZONE = '2A0E'
- Bluetooth4LE.CHARACTERISTICS_TX_POWER_LEVEL = '2A07'
- Bluetooth4LE.CHARACTERISTICS_UNREAD_ALERT_STATUS = '2A45'

Descriptors

・Bluetooth4LE.DESCRIPTORS_CHARACTERISTIC_AGGREGATE_FORMAT = '2905'
・Bluetooth4LE.DESCRIPTORS_CHARACTERISTIC_EXTENDED_PROPERTIES = '2900'
・Bluetooth4LE.DESCRIPTORS_CHARACTERISTIC_PRESENTATION_FORMAT = '2904'
・Bluetooth4LE.DESCRIPTORS_CHARACTERISTIC_USER_DESCRIPTION = '2901'
・Bluetooth4LE.DESCRIPTORS_CLIENT_CHARACTERISTIC_CONFIGURATION = '2902'
・Bluetooth4LE.DESCRIPTORS_EXTERNAL_REPORT_REFERENCE = '2907'
・Bluetooth4LE.DESCRIPTORS_REPORT_REFERENCE = '2908'
・Bluetooth4LE.DESCRIPTORS_SERVER_CHARACTERISTIC_CONFIGURATION = '2903'
・Bluetooth4LE.DESCRIPTORS_VALID_RANGE = '2906'

ValueType

・Bluetooth4LE.VALUE_TYPE_UUID = 0
・Bluetooth4LE.VALUE_TYPE_BOOLEAN = 1
・Bluetooth4LE.VALUE_TYPE_2BIT = 2
・Bluetooth4LE.VALUE_TYPE_NIBBLE = 3
・Bluetooth4LE.VALUE_TYPE_8BIT = 4
・Bluetooth4LE.VALUE_TYPE_16BIT = 5
・Bluetooth4LE.VALUE_TYPE_24BIT = 6
・Bluetooth4LE.VALUE_TYPE_32BIT = 7
・Bluetooth4LE.VALUE_TYPE_UINT8 = 8
・Bluetooth4LE.VALUE_TYPE_UINT12 = 9
・Bluetooth4LE.VALUE_TYPE_UINT16 = 10
・Bluetooth4LE.VALUE_TYPE_UINT24 = 11
・Bluetooth4LE.VALUE_TYPE_UINT32 = 12
・Bluetooth4LE.VALUE_TYPE_UINT40 = 13
・Bluetooth4LE.VALUE_TYPE_UINT48 = 14

・Bluetooth4LE.VALUE_TYPE_UINT64 = 15
・Bluetooth4LE.VALUE_TYPE_UINT128 = 16
・Bluetooth4LE.VALUE_TYPE_SINT8 = 17
・Bluetooth4LE.VALUE_TYPE_SINT12 = 18
・Bluetooth4LE.VALUE_TYPE_SINT16 = 19
・Bluetooth4LE.VALUE_TYPE_SINT24 = 20
・Bluetooth4LE.VALUE_TYPE_SINT32 = 21
・Bluetooth4LE.VALUE_TYPE_SINT48 = 22
・Bluetooth4LE.VALUE_TYPE_SINT64 = 23
・Bluetooth4LE.VALUE_TYPE_SINT128 = 24
・Bluetooth4LE.VALUE_TYPE_FLOAT32 = 25
・Bluetooth4LE.VALUE_TYPE_FLOAT64 = 26
・Bluetooth4LE.VALUE_TYPE_SFLOAT = 27
・Bluetooth4LE.VALUE_TYPE_FLOAT = 28
・Bluetooth4LE.VALUE_TYPE_DUNIT16 = 29
・Bluetooth4LE.VALUE_TYPE_UTF8S = 30
・Bluetooth4LE.VALUE_TYPE_UTF16S = 31

プロパティ:

- peripheralList

- scanDevices
- connect
- disconnect

scanDevices

デバイスのスキャンを開始します。

```
Bluetooth4LE.scanDevices(services);
```

services

検索対象サービス(Array)

connect

デバイスへ接続します。

```
Bluetooth4LE.connect(peripheral, successCallback, errorCallback);
```

peripheral

デバイス情報

successCallback

接続が正常に取得された場合に呼び出されます。

```
function() {
    // Do something.
}
```

errorCallback

接続に失敗した発生した場合に呼び出されます。

```
function() {
    // Do something in case of error.
}
```

disconnect

デバイスとの接続を解除します。

```
Bluetooth4LE.disconnect(peripheral, successCallback, errorCallback);
```

peripheral

デバイス情報

successCallback

接続解除が正常に取得された場合に呼び出されます。

```
function() {
    // Do something.
}
```

errorCallback

接続解除に失敗した発生した場合に呼び出されます。

```
function() {
    // Do something in case of error.
}
```

※ご注意※
Androidでは現在ご利用頂けません。
iPhone4以前のiPhone、iPad2以前のiPadではご利用頂けません。

プロパティ:
- deviceName : デバイス名
- id : デバイスID（iOS:UUID/Android:MAC Address）

メソッド:
- writeValueForCharacteristic
- readValueForCharacteristic
- writeValueForDescriptor
- readValueForDescriptor

writeValueForCharacteristic

キャラクタリスティクに値を書き込みます。

```
peripheral.writeValueForCharacteristic(value, valueType, characteristic,
                                       writeType, successCallback, errorCallback);
```

value

書き込む値

valueType

書き込む値のタイプ

service

対象のサービス

characteristic

対象のキャラクタリスティック

writeType

書き込みタイプ
- WriteWithResponse : 0
- WriteWithoutResponse : 1

successCallback

書き込みが正常終了した場合に呼び出されます。

```
function() {
    // Do something.
}
```

errorCallback

書き込みに失敗した発生した場合に呼び出されます。

```
function() {
    // Do something in case of error.
}
```

readValueForCharacteristic

キャラクタリスティクから値を読み込みます。

```
peripheral.readValueForCharacteristic(service, characteristic, valueType, successCallback, errorCallback);
```

service

対象のサービス

characteristic

対象のキャラクタリスティック

valueType

読み込む値のタイプ

successCallback

読み込みが正常に取得された場合に呼び出されます。

```
function(value) {
    // Do something.
}
```

パラメータ

・value: 読み込んだ値

errorCallback

読み込みに失敗した発生した場合に呼び出されます。

```
function() {
    // Do something in case of error.
}
```

writeValueForDescriptor

ディスクリプタに値を書き込みます。

```
peripheral.writeValueForDescriptor(value, valueType, service, characteristic, descriptor,
                                                                    successCallback, errorCallback);
```

value

書き込む値

valueType

書き込む値のタイプ

service

対象のサービス

characteristic

対象のキャラクタリスティック

descriptor

対象のディスクリプタ

successCallback

書き込みが正常に取得された場合に呼び出されます。

```
function() {
    // Do something.
}
```

errorCallback

書き込みに失敗した発生した場合に呼び出されます。

```
function() {
    // Do something in case of error.
}
```

readValueForDescriptor

ディスクリプタから値を読み込みます。

```
peripheral.readValueForDescriptor(service, characteristic, descriptor, valueType,
                                                                successCallback, errorCallback);
```

service

対象のサービス

characteristic

対象のキャラクタリスティック

descriptor

対象のディスクリプタ

successCallback

読み込みが正常終了した場合に呼び出されます。

```
function(value) {
   // Do something with the value.
}
```

パラメータ

・value: 読み込んだ値

errorCallback

読み込みに失敗した発生した場合に呼び出されます。

```
function() {
    // Do something in case of error.
}
```

※ご注意※
iOSではご利用頂けません。
またAndroidでご使用の際はNFC機能を搭載した端末が必要です。

メソッド:
- getIDm
- getPMm
- requestSystemCode
- requestServiceCode
- requestResponse
- readWithoutEncryption
- writeWithoutEncryption

getIDm

IDmを取得します。

```
Felica.getIDm(successCallback, errorCallback);
```

successCallback

IDmが正常に取得された場合に呼び出されます。

```
function(IDm) {
    // Do something with the IDm.
}
```

パラメータ
- IDm: 取得したIDm(String)

errorCallback

取得に失敗した発生した場合に呼び出されます。

```
function() {
    // Do something in case of error.
}
```

getPMm

PMmを取得します。

```
Felica.getPMm(successCallback, errorCallback);
```

successCallback

PMmが正常に取得された場合に呼び出されます。

```
function(PMm) {
    // Do something with the PMm.
}
```

パラメータ

・PMm: 取得したPMm(String)

errorCallback

取得に失敗した発生した場合に呼び出されます。

```
function() {
    // Do something in case of error.
}
```

requestSystemCode

システムコード取得要求を送信します。

```
Felica.getPMm(successCallback, errorCallback);
```

successCallback

システムコードが正常に取得された場合に呼び出されます。

```
function(systemcodes) {
    // Do something with the system codes.
}
```

パラメータ

・systemcodes: 取得したシステムコード(Array)

errorCallback

取得に失敗した発生した場合に呼び出されます。

```
function() {
    // Do something in case of error.
}
```

requestResponse

応答要求を行います

```
Felica.requestResponse(successCallback, errorCallback);
```

successCallback

応答要求が正常に行われた場合に呼び出されます。

```
function(mode) {
    // Do something with the mode.
}
```

パラメータ

・mode: システムの最新状態

errorCallback

取得に失敗した発生した場合に呼び出されます。

```
function() {
    // Do something in case of error.
}
```

readWithoutEncryption

認証を必要としないサービスからブロックデータを読み出します。

```
Felica.readWithoutEncryption(serviceCode, addr, successCallback, errorCallback);
```

serviceCode

サービスコード

addr

読み込みを開始するブロックデータアドレス

successCallback

読み込みが正常に行われた場合に呼び出されます。

```
function(blockdata) {
    // Do something with the block data.
}
```

パラメータ

・blockdata: 読み込まれたブロックデータ

errorCallback

読み込みに失敗した発生した場合に呼び出されます。

```
function() {
    // Do something in case of error.
}
```

writeWithoutEncryption

認証を必要としないサービスへブロックデータを書き込みます。

```
Felica.writeWithEncryption(serviceCode, addr, buff, successCallback, errorCallback);
```

serviceCode

サービスコード

serviceCode

サービスコード

addr

書き込みを開始するブロックデータアドレス

buff

書き込みデータ

successCallback

書き込みが正常に行われた場合に呼び出されます。

```
function() {
  // Do something.
}
```

errorCallback

書き込みに失敗した発生した場合に呼び出されます。

```
function() {
   // Do something in case of error.
}
```

AppVersion

メソッド:
- **get**

get

アプリバージョンを取得します。

```
AppVersion.get(callback);
```

callback

アプリバージョンが取得されます。

```
function(version) {
    // Do something with App Version.
}
```

パラメータ
- version: アプリバージョン(String)
  iOSの場合：info.plistに設定したbundle version
  Androidの場合：AndroidManifest.xmlに設定したversionName

Contacts

メソッド:

- create
- search

create

新しいコンタクトオブジェクトを作成します。

```
var contact = Contacts.create();
```

search

デバイス内の電話帳データベースから検索を行います

```
Contacts.search(conditions, orderby, successCallback, errorCallback);
```

conditions

検索条件

orderby

取得順

successCallback

連絡先情報が正常に取得された場合に呼び出されます。

```
function(contacts) {
  // Do something.
}
```

errorCallback

連絡先情報取得に失敗した発生した場合に呼び出されます。

```
function(message) {
  //  Show a helpful message.
}
```

パラメータ

・message: ネイティブ側からのエラーメッセージ

サンプル

```
var conditions = new Array();
conditions.push("displayName = 'Test'");
conditions.push("addresses != ''");
Contacts.search(conditions, Contacts.SEARCH_ORDER_ASC, successCallback,
                errorCallback);
```

Contact

プロパティ:
- id: ID.
- displayName: 表示名
- name: 名前
- nickname: ニックネーム
- phoneNumbers: 電話番号（Array）
- emails: メールアドレス（Array）
- addresses: 住所（Array）
- ims: IMアドレス（Array）
- organizations: 組織（Array）
- birthday: 誕生日
- note: 備考
- photos: 連絡先写真（Array）
- categories: カテゴリ（Array）
- urls: ホームページ（Array）

メソッド:
- remove
- save

remove

連絡先情報を削除します。

```
Contact.remove(successCallback, errorCallback);
```

successCallback

連絡先情報が正常に削除された場合に呼び出されます。

```
function() {
  // Do something.
}
```

errorCallback

連絡先情報の削除に失敗した場合に呼び出されます。

```
function(message) {
  //  Show a helpful message.
}
```

パラメータ

・message: ネイティブ側からのエラーメッセージ

save

連絡先情報を保存します。

```
Contact.save(successCallback, errorCallback);
```

successCallback

連絡先情報保存が正常に削除された場合に呼び出されます。

```
function() {
  // Do something.
}
```

errorCallback

連絡先情報の保存に失敗した場合に呼び出されます。

```
function(message) {
  //  Show a helpful message.
}
```

パラメータ

・message: ネイティブ側からのエラーメッセージ

ContactPhoneNumber

プロパティ:
- type: 電話番号タイプ
- number: 電話番号

ContactAddress

プロパティ:
- type: アドレスタイプ
- formatted:表示用にフォーマットされたフルアドレス
- streetAddress:ストリートアドレス
- locality:市、もしくは町名
- region:地方名
- postalCode:郵便番号
- country:国

ContactEmailAddress

プロパティ:
- type: メールアドレスタイプ
- address:メールアドレス

ContactIMAddress

プロパティ:
- type: アドレスタイプ
- address:IMアドレス

ContactPhoto

プロパティ:

・uri:フォト画像へのURI

ContactCategory

プロパティ:

・category:カテゴリ名

ContactURL

プロパティ:

- type:URLタイプ
- url:URL

ContactOrganization

プロパティ:

- type:組織タイプ
- name:組織名
- department:部門
- title:タイトル

PDFViewer

メソッド:

- showWithFileObject
- showWithFilePath
- showWithURL

showWithFileObject

ファイルオブジェクトを使用してPDFを表示します

```
PDFViewer.showWithFileObject(file);
```

file

PDFファイルへのファイルオブジェクト

showWithFilePath

ファイルパスで指定されたPDFを表示します

```
PDFViewer.showWithFilePath(filePath);
```

filePath

PDFファイルへのファイルパス

showWithURL

URLで指定されたPDFを表示します

```
PDFViewer.showWithURL(url);
```

url

PDFファイルへのURL
※PDFを表示する前に、該当ファイルをダウンロードします。

Ad

メソッド:
- create
- show
- hide

create

広告表示用ビューを作成し、広告をロードします。

```
Ad.create(position, successCallback, errorCallback, [options]);
```

position

広告表示用ビューを表示する位置を指定します。指定値は以下の通り
AD.POSITION_TOP : 0
AD.POSITION_BOTTOM : 1

successCallback

広告表示用ビューが作成され、広告が正常に読み込まれた場合に呼ばれます

```
function() {
  // Do something.
}
```

errorCallback

広告表示用ビューの作成に失敗した、もしくは広告の読み込みに失敗した場合に呼ばれます

```
function() {
    // Do something in case of error.
}
```

options

オプションを設定します

```
{
 id : a03ce87ff88282d3a7799fe89de30b  // 例) パブリッシャーid : a14e62fe30c05d7
};
```

オプション

・id: Admob用のパブリッシャーIDを指定します。指定する文字列はEncryptorオブジェクトを使用して暗号化された文字列を指定してください。（※Androidのみ）