



Red Hat Mobile Application Platform

ホスト型 3

クライアント API

Red Hat Mobile Application Platform ホスト型 3 向け

Red Hat Customer Content
Services

法律上の通知

Copyright © 2017 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

本ガイドでは、RHMAP クライアント API の参考情報を提供します。

目次

前書き	9
第1章 \$FH.AUTH	11
1.1. 例	13
1.2. セッションの確認	20
1.3. 独自認証プロバイダーの作成	23
第2章 \$FH.CLOUD	24
2.1. 例	24
第3章 \$FH.GETCLOUDURL	27
3.1. 例	27
第4章 \$FH.GETFHPARAMS	29
4.1. 例	29
第5章 \$FH.FORMS	31
5.1. SDK の構造	31
5.1.1. コア	31
5.2. \$FH.FORMS.INIT	31
5.2.1. 詳細	31
5.2.2. 例	32
5.3. \$FH.FORMS.GETFORMS	32
5.3.1. 詳細	32
5.3.2. 例	32
5.4. \$FH.FORMS.GETFORM	32
5.4.1. 詳細	32
5.4.2. 例	32
5.5. \$FH.FORMS.GETTHEME	33
5.5.1. 詳細	33
5.5.2. 例	33
5.6. \$FH.FORMS.GETSUBMISSIONS	33
5.6.1. 詳細	33
5.6.2. 例	33
5.7. \$FH.FORMS.DOWNLOADSUBMISSION	34
5.7.1. 詳細	34
5.7.2. Example: Callback Passed	34
5.7.3. 例: コールバックが渡されない場合	34
5.8. \$FH.FORMS.LOG	35
5.8.1. 詳細	35
5.8.2. 例	35
5.9. \$FH.FORMS.GETLOGS	35
5.9.1. 詳細	35
5.10. グローバルイベント	36
5.10.1. 詳細	36
5.10.2. \$fh.forms.on	36
5.10.3. \$fh.forms.once	36
5.11. \$FH.FORMS.MODELS.FORMS	36
5.11.1. 詳細	36
5.11.2. \$fh.forms.models.forms.clearAllForms	36
5.11.2.1. 詳細	37
5.11.2.2. 例	37
5.11.3. \$fh.forms.models.forms.isFormUpdated	37

5.11.3.1. 詳細	37
5.11.3.2. 例	37
5.11.4. \$fh.forms.models.forms.getFormMetaById	37
5.11.4.1. 詳細	37
5.11.4.2. 例	37
5.11.5. \$fh.forms.models.forms.refresh	38
5.11.5.1. 詳細	38
5.11.5.2. 例	38
5.11.6. \$fh.forms.models.forms.clearLocal	38
5.11.6.1. 詳細	38
5.11.6.2. 例	38
5.11.7. \$fh.forms.models.forms.getFormsList	39
5.11.7.1. 詳細	39
5.11.7.2. 例	39
5.11.8. \$fh.forms.models.forms.size	39
5.11.8.1. 詳細	39
5.11.8.2. 例	39
5.12. \$FH.FORMS.MODELS.FORM	39
5.12.1. 詳細	39
5.12.2. \$fh.forms.models.Form.constructor()	40
5.12.2.1. 詳細	40
5.12.2.2. 例	40
5.12.3. \$fh.forms.models.Form.getLastUpdate()	40
5.12.3.1. 詳細	40
5.12.3.2. 例	40
5.12.4. \$fh.forms.models.Form.getPageModelList()	40
5.12.4.1. 詳細	40
5.12.4.2. 例	41
5.12.4.3. \$fh.forms.models.Form.getRuleEngine()	41
5.12.4.4. 詳細	41
5.12.4.5. 例	41
5.12.5. \$fh.forms.models.Form.getName()	41
5.12.5.1. 詳細	41
5.12.5.2. 例	41
5.12.6. \$fh.forms.models.Form.getDescription()	42
5.12.6.1. 詳細	42
5.12.6.2. 例	42
5.12.7. \$fh.forms.models.Form.getFormId()	42
5.12.7.1. 詳細	42
5.12.7.2. 例	42
5.12.8. \$fh.forms.models.Form.getFieldModelById()	43
5.12.8.1. 詳細	43
5.12.8.2. 例	43
5.12.9. \$fh.forms.models.Form.newSubmission()	43
5.12.9.1. 詳細	43
5.12.9.2. 例	43
5.12.10. \$fh.forms.models.Form.removeFromCache()	43
5.12.10.1. 詳細	43
5.12.10.2. 例	44
5.12.11. \$fh.forms.models.Form.refresh()	44
5.12.11.1. 詳細	44
5.12.11.2. 例	44
5.12.12. \$fh.forms.models.Form.clearLocal()	44

5.12.12.1. 詳細	44
5.12.12.2. 例	44
5.13. \$FH.FORMS.MODELS.PAGE	44
5.13.1. 詳細	44
5.13.2. \$fh.forms.models.Page.setVisible()	45
5.13.2.1. 詳細	45
5.13.2.2. 例	45
5.13.3. \$fh.forms.models.Page.getName()	45
5.13.3.1. 詳細	45
5.13.3.2. 例	45
5.13.4. \$fh.forms.models.Page.getDescription()	45
5.13.4.1. 詳細	45
5.13.4.2. 例	45
5.13.5. \$fh.forms.models.Page.getFieldModelList()	45
5.13.5.1. 詳細	46
5.13.5.2. 例	46
5.13.6. \$fh.forms.models.Page.getFieldModelById()	46
5.13.6.1. 詳細	46
5.13.6.2. 例	46
5.14. \$FH.FORMS.MODELS.FIELD	46
5.14.1. 詳細	46
5.14.2. \$fh.forms.models.Field.isRequired()	46
5.14.2.1. 詳細	46
5.14.2.2. 例	47
5.14.3. \$fh.forms.models.Field.isRepeating()	47
5.14.3.1. 詳細	47
5.14.3.2. 例	47
5.14.4. \$fh.forms.models.Field.getType()	47
5.14.4.1. 詳細	47
5.14.4.2. 例	47
5.14.5. \$fh.forms.models.Field.getName()	47
5.14.5.1. 詳細	47
5.14.6. \$fh.forms.models.Field.getCode()	47
5.14.6.1. 詳細	47
5.14.7. \$fh.forms.models.Field.getHelpText()	48
5.14.7.1. 詳細	48
5.14.7.2. 例	48
5.14.8. \$fh.forms.models.Field.validate(inputValue)	48
5.14.8.1. 詳細	48
5.14.8.2. 例	48
5.14.9. \$fh.forms.models.Field.getRules()	48
5.14.9.1. 詳細	48
5.14.9.2. 例	48
5.14.10. \$fh.forms.models.Field.getCheckboxOptions()	49
5.14.10.1. 詳細	49
5.14.10.2. 例	49
5.14.11. \$fh.forms.models.Field.getRadioOption()	49
5.14.11.1. 詳細	49
5.14.11.2. 例	49
5.15. \$FH.FORMS.MODELS.SUBMISSIONS	49
5.15.1. \$fh.forms.models.submissions.getSubmissions()	49
5.15.1.1. 詳細	50
5.15.1.2. 例	50

5.15.2. \$fh.forms.models.submissions.getSubmissionMetaList()	50
5.15.2.1. 詳細	50
5.15.2.2. 例	50
5.15.3. \$fh.forms.models.submissions.findByFormId(formId)	50
5.15.3.1. 詳細	50
5.15.3.2. 例	50
5.15.4. \$fh.forms.models.submissions.clear(cb)	50
5.15.4.1. 詳細	51
5.15.4.2. 例	51
5.15.5. \$fh.forms.models.submissions.getDrafts()	51
5.15.5.1. 詳細	51
5.15.5.2. 例	51
5.15.6. \$fh.forms.models.submissions.getPending()	51
5.15.6.1. 詳細	51
5.15.6.2. 例	51
5.15.7. \$fh.forms.models.submissions.getSubmitted()	51
5.15.7.1. 詳細	51
5.15.7.2. 例	52
5.15.8. \$fh.forms.models.submissions.getError()	52
5.15.8.1. 詳細	52
5.15.8.2. 例	52
5.15.9. \$fh.forms.models.submissions.getInProgress()	52
5.15.9.1. 詳細	52
5.15.9.2. 例	52
5.15.10. \$fh.forms.models.submissions.getSubmissionByMeta(meta,cb)	52
5.15.10.1. 詳細	52
5.15.10.2. 例	52
5.15.11. \$fh.forms.models.submissions.getSubmissionByLocalId(localId,cb)	53
5.15.11.1. 詳細	53
5.15.11.2. 例	53
5.15.12. \$fh.forms.models.submissions.getSubmissionByRemoteId(remoteId,cb)	53
5.15.12.1. 詳細	53
5.15.12.2. 例	53
5.16. \$FH.FORMS.MODELS.SUBMISSION	53
5.16.1. 詳細	53
5.16.2. \$fh.forms.models.Submission.saveDraft(cb)	54
5.16.2.1. 詳細	54
5.16.2.2. 例	54
5.16.3. \$fh.forms.models.Submission.submit()	54
5.16.3.1. 詳細	54
5.16.3.2. 例	54
5.16.4. \$fh.forms.models.Submission.getStatus()	55
5.16.4.1. 詳細	55
5.16.4.2. 例	55
5.16.5. \$fh.forms.models.Submission.addComment()	55
5.16.5.1. 詳細	55
5.16.5.2. 例	55
5.16.6. \$fh.forms.models.Submission.getComments()	55
5.16.6.1. 詳細	55
5.16.6.2. 例	55
5.16.7. \$fh.forms.models.Submission.removeComment(timestamp)	55
5.16.7.1. 詳細	55
5.16.7.2. 例	56

5.16.8. \$fh.forms.models.Submission.addInputValue(params,cb)	56
5.16.8.1. 詳細	56
5.16.8.2. 例	56
5.16.9. \$fh.forms.models.Submission.startInputTransaction()	56
5.16.9.1. 詳細	56
5.16.9.2. 例	56
5.16.10. \$fh.forms.models.Submission.endInputTransaction(isSucceed)	56
5.16.10.1. 詳細	56
5.16.10.2. 例	57
5.16.11. \$fh.forms.models.Submission.reset()	57
5.16.11.1. 詳細	57
5.16.11.2. 例	57
5.16.12. \$fh.forms.models.Submission.getForm(cb)	57
5.16.12.1. 詳細	57
5.16.12.2. 例	57
5.16.13. \$fh.forms.Submission.removeFieldValue(fieldId, [index])	58
5.16.13.1. 詳細	58
5.16.13.2. 例	58
5.16.14. \$fh.forms.Submission.getInputValueByFieldId(field,cb)	58
5.16.14.1. 詳細	58
5.16.15. 提出のイベント	58
5.16.15.1. inprogress: The Submission Is In The Process Of Uploading.	58
5.16.15.2. error: There Was An Error Uploading The Submission.	59
5.16.15.3. savedraft: The Submission Was Saved As A Draft	59
5.16.15.4. validationerror: There Was A Validation Error When Making A Submission.	59
5.16.15.5. submit: The Submission Is Valid And Can Now Be Uploaded.	59
5.16.15.6. submitted: The Submission Is Valid And Has Completed Uploading All Data.	60
5.16.15.7. queued: The Submission JSON Definition Has Been Uploaded. Proceeding To Upload Any Files.	60
5.16.15.8. progress: The Progress For A Submission Has Been Incremented.	60 60
5.16.15.8.1. progressJSON	60
5.16.15.9. downloaded: The Submission Has Completed Downloading (Only Used When Downloading Submissions.)	60
第6章 \$FH.HASH	62
6.1. 例	62
第7章 \$FH.INIT	63
7.1. 例	63
第8章 \$FH.MBAAS	66
8.1. 例	66
第9章 \$FH.PUSH	67
9.1. 例	67
第10章 \$FH.SEC	71
10.1. 例	71
第11章 \$FH.SYNC	73
11.1. \$FH.SYNC.INIT	73
11.1.1. 詳細	73
11.1.2. 例	73
11.2. \$FH.SYNC.NOTIFY	81
11.2.1. 詳細	81
11.2.2. 例	82

11.2.3. 同期通知	88
11.2.3.1. Sync Started	88
11.2.3.1.1. 通知コード	89
11.2.3.1.2. 通知の構造	89
11.2.3.2. Sync Complete	89
11.2.3.2.1. 通知コード	89
11.2.3.2.2. 通知の構造	89
11.2.3.3. Sync Failed	89
11.2.3.3.1. 通知コード	90
11.2.3.3.2. 通知の構造	90
11.2.3.4. Record Delta Received	90
11.2.3.4.1. 通知コード	90
11.2.3.4.2. 通知の構造	90
11.2.3.5. Delta Received	91
11.2.3.5.1. 通知コード	91
11.2.3.5.2. 通知の構造	91
11.2.3.6. Local Update Applied	91
11.2.3.6.1. 通知コード	91
11.2.3.6.2. 通知の構造	91
11.2.3.7. Remote Update Applied	92
11.2.3.7.1. 通知コード	92
11.2.3.7.2. 通知の構造	92
11.2.3.8. Remote Update Failed	92
11.2.3.8.1. 通知コード	93
11.2.3.8.2. 通知の構造	93
11.2.3.9. Collision Detected	93
11.2.3.9.1. 通知コード	93
11.2.3.9.2. 通知の構造	93
11.2.3.10. Offline Update	93
11.2.3.10.1. 通知コード	93
11.2.3.10.2. 通知の構造	93
11.2.3.11. Client Storage Failed	94
11.2.3.11.1. 通知コード	94
11.2.3.11.2. 通知の構造	94
11.3. \$FH.SYNC.MANAGE	94
11.3.1. 詳細	94
11.3.2. 例	94
11.4. \$FH.SYNC.DOLIST	97
11.4.1. 詳細	97
11.4.2. 例	97
11.5. \$FH.SYNC.DOCREATE	99
11.5.1. 詳細	99
11.5.2. 例	99
11.6. \$FH.SYNC.DOREAD	101
11.6.1. 詳細	101
11.6.2. 例	101
11.7. \$FH.SYNC.DOUPDATE	102
11.7.1. 詳細	102
11.7.2. 例	102
11.8. \$FH.SYNC.DODELETE	104
11.8.1. 詳細	104
11.8.2. 例	104
11.9. \$FH.SYNC.STARTSYNC	105

-----	---
11.9.1. 詳細	105
11.9.2. 例	105
11.10. \$FH.SYNC.STOPSYNC	106
11.10.1. 詳細	107
11.10.2. 例	107
11.11. \$FH.SYNC.DOSYNC	108
11.11.1. 詳細	108
11.11.2. 例	108
11.12. \$FH.SYNC.FORCESYNC	108
11.12.1. 詳細	108
11.12.2. 例	109
第12章 \$FH.ACT	110
12.1. 例	110

前書き

凡例:

C - Cordova

W1 - Web Apps

A2 - Android SDK

I1 - iOS Objective-C

I2 - SDK iOS

S - Swift SDK

W2 - Windows

X - Xamarin

API 名と説明	C	W1	A2	i1	i2	S	W2	X
\$fh.auth	y	y	y	y	y	y	y	y
\$fh.cloud	y	y	y	y	y	y	y	y
\$fh.getcloudurl	y	y	y	y	y	y	y	y
\$fh.getfhparams	y	y	y	y	y	y	y	y
\$fh.forms	y	y						
\$fh.hash	y	y						
\$fh.init	y	y	y	y	y	y	y	y
\$fh.mbaas		y						
\$fh.push	y	y	y	y	y	y	y	y

API 名と説明	C	W1	A2	i1	i2	S	W2	X
\$fh.sec	y	y						
\$fh.sync	y	y	y	y	y	y	y	y
\$fh.act	y	y	y	y	y	y	y	y

第1章 \$FH.AUTH

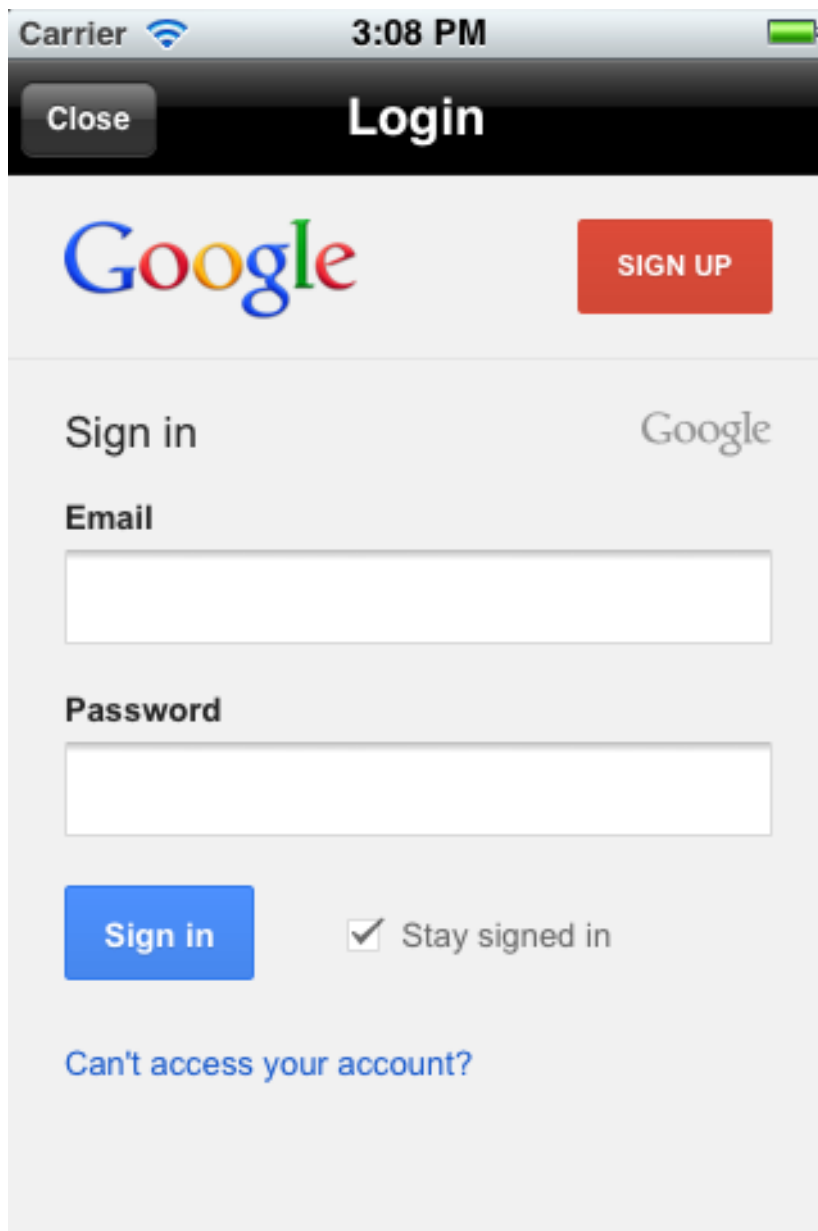
```
$fh.auth(options, success, failure);
```

認証を行うとともに、オプションでアクセス権限管理によってユーザー承認も行います。fh.auth を使用するには、ドメイン管理者のアカウントが必要になります。ユーザーレベルについては、[ユーザーの管理](#) を参照してください。oauth プロバイダーに対して \$fh.auth を使用する予定の場合は、[OAuth ポリシー](#) を参照してください。

一般的に、認証タイプが FeedHenry または LDAP の場合は、ユーザーが自分の認証情報を入力し、auth リクエストのインスタンスでそれらを設定できるようにするビューを構築するだけで十分です。

認証ポリシーのタイプが OAuth の場合、認証プロセスは複雑なものになります。ユーザーは OAuth プロバイダーのログインページにログインし、アプリケーションが自分の情報にアクセスできるようにするためにパーミッションを付与する必要があります。

このプロセスを簡単なものにするために、ネイティブの SDK (iOS、Android および .NET) が追加の UI コンポーネントを提供し、シームレスに処理できるようにします。UI コンポーネントが画面上に現れ、OAuth プロバイダーの認証ページを読み込みます。コンポーネントは OAuth プロセスが終了するとこれを検知して自動的に閉じ、結果を success または failure 関数に渡します。UI コンポーネントは以下のようなものです。



サポートされるプラットフォーム

- ✧ JavaScript SDK
 - Cordova
 - Web Apps
- ✧ Android SDK
- ✧ iOS Objective-C SDK
- ✧ iOS Swift SDK
- ✧ .NET SDK
 - Windows
 - Xamarin

詳細なバージョン情報については、[Supported Configurations](#) (英語) を参照してください。

1.1. 179

JavaScript

```

// LDAP or Platform User Example
$fh.auth({
  "policyId": "My LDAP Auth Policy", // name of auth policy to use - see
  link:{ProductFeatures}#administration[Auth Policies Administration] for
  details on how to configure an auth policy
  "clientToken": "myAppId", // Your App ID
  "endRedirectUrl": window.location.href, // The URL to return to after
  authentication. Optional
  "params": { // the parameters associated with the requested auth policy
  - see below for full details.
    "userId": "joe@bloggs.com", // LDAP or Platform username
    "password": "password" // LDAP or Platform password
  }
}, function (res) {
  // Authentication successful - store sessionToken in variable
  var sessionToken = res.sessionToken; // The platform session identifier
  var authResponse = res.authResponse; // The authentication information
  returned from the authentication service.
  // This may include things such as validated email address,
  // OAuth token or other response data from the authentication service
}, function (msg, err) {
  var errorMsg = err.message;
  /* Possible errors:
    unknown_policyId - The policyId provided did not match any defined
    policy. Check the auth policies defined. See link:
    {ProductFeatures}#administration[Auth Policies Administration]
    user_not_found - The auth policy associated with the policyId
    provided has been set up to require that all users authenticating exist
    on the platform, but this user does not exists.
    user_not_approved - - The auth policy associated with the policyId
    provided has been set up to require that all users authenticating are in
    a list of approved users, but this user is not in that list.
    user_disabled - The user has been disabled from logging in.
    user_purge_data - The user has been flagged for data purge and
    all local data should be deleted.
    device_disabled - The device has been disabled. No user or apps
    can log in from the requesting device.
    device_purge_data - The device has been flagged for data purge
    and all local data should be deleted.
  */
  if (errorMsg === "user_purge_data" || errorMsg === "device_purge_data") {
    // User or device has been black listed from administration console and
    all local data should be wiped
  } else {
    alert("Authentication failed - " + errorMsg);
  }
});

// OAuth 2.0 Example
// OAuth does not require any params, instead the "authCallback" param
should be set on the $fh.auth call.

```

```
// This should be a function name that you have defined, and will be
// called after Auth has completed.
$fh.auth({
  "policyId": "My OAuth Policy",
  "clientToken": "myAppId",
  "authCallback": "authLoginCallback",
  "endRedirectUrl": window.location.href
}, function () {
  //
}, function () {
  //
});

var authLoginCallback = function(err, res) {
  if (!err) {
    // Authentication successful - store sessionToken in variable
    var sessionToken = res.sessionToken;
  } else {
    alert("Authentication failed - " + err.message);
  }
}
```

Android (Java)

下記の例にあるようにビルトイン OAuth ハンドラーを使用するには、以下の設定をアプリケーションの **AndroidManifest.xml** ファイルの **application** 要素に追加する必要があります。

```
<application>
...
<activity android:name="com.feedhenry.sdk.oauth.FHOAuthIntent" />
</application>
```



注記

Auth コードを呼び出す前に、**FH.init** コードが初期化されていることを確認してください。例については [こちら](#) を参照してください。

```
//Example code to authenticate a user with username and password that are
//defined in an auth policy called "MyFeedHenryPolicy"
private void loginWithFh(){
  EditText userField = (EditText) findViewById(R.id.fh_login_user);
  EditText passField = (EditText) findViewById(R.id.fh_login_password);
  String userName = userField.getText().toString();
  String password = passField.getText().toString();
  if("").equals(userName){
    FhUtil.showMessage(this, "Error", "User name is empty");
    return;
  }
  if("").equals(password){
    FhUtil.showMessage(this, "Error", "Password is empty");
    return;
  }
  try{
```

```

    FHAuthRequest authRequest = FH.buildAuthRequest("MyFeedHenryPolicy",
        userName, password);
    authRequest.executeAsync(new FHActCallback() {

        @Override
        public void success(FHResponse resp) {
            Log.d("FHLoginActivity", "Login success");
        }

        @Override
        public void fail(FHResponse resp) {
            Log.d("FHLoginActivity", "Login fail");
        }
    });
} catch (Exception e) {
    e.printStackTrace();
}
}

```

認証ポリシーのタイプが OAuth の場合、Intent が起動して OAuth プロバイダーの認証ページが読み込まれます。OAuth プロセスが終了するとこれを検知して自動的に閉じ、結果を success または failure 関数に渡します。これを有効にするには、アプリケーションのコンテキストで FHAuthRequest インスタンスの **setPresentingActivity** メソッドを呼び出すだけです。

```

private void doOAuth(){
    try{
        FHAuthRequest authRequest = FH.buildAuthRequest();
        authRequest.setPresentingActivity(this);
        authRequest.setAuthPolicyId("MyGooglePolicy"); // "MyGooglePolicy" should
        be replaced with policy id you created
        authRequest.executeAsync(new FHActCallback() {

            @Override
            public void success(FHResponse resp) {
                Log.d("FHAAuthActivity", resp.getJson().toString());
            }

            @Override
            public void fail(FHResponse resp) {
                Log.d("FHAAuthActivity", resp.getErrorMessage());
            }
        });
    } catch (Exception e){
        Log.e("FHAAuthActivity", e.getMessage(), e);
    }
}

```

setPresentingActivity メソッドが呼び出されない場合は、自身のコードでこれを処理することができます。例を示します。

```

private void doOAuth(){
    try{
        FHAuthRequest authRequest = FH.buildAuthRequest();
        authRequest.setAuthPolicyId("MyGooglePolicy"); // "MyGooglePolicy"

```

```

should be replaced with policy id you created
authRequest.executeAsync(new FHActCallback() {

    @Override
    public void success(FHResponse resp) {
        Log.d("FHAAuthActivity", resp.getJson().toString());
        //because the setPresentingActivity method is not called, the
        reponse will contain a URL which should be used for user to login.
        Normally it should be loaded into a WebView
        String url = resp.getJson().getString("url");
        // load the url in a WebView, and then a series of redirects will
        happen
        // the last url will contain a string "status=complete"
        // and there will be a query parameter called "authResponse" in
        that url
        // the value of that parameter is the data returned from the OAuth
        provided (JSON stringified and URL encoded)
    }

    @Override
    public void fail(FHResponse resp) {
        Log.d("FHAAuthActivity", resp.getErrorMessage());
    }
});
} catch (Exception e) {
    Log.e("FHAAuthActivity", e.getMessage(), e);
}
}

```

iOS (Objective-C)

```

//Example to authenticate user using username and password
NSString* userName = self.usernameField.text;
if(!userName){
    return [self showMessage:@"Error" message:@"User Name field is
required"];
}
NSString* password = self.passwordField.text;
if(!password){
    return [self showMessage:@"Error" message:@"Password field is
required"];
}
FHAAuthRequeust* authRequest = [FH buildAuthRequest];
[authRequest authWithPolicyId:@"MyFeedHenryPolicy" UserId:userName
Password:password]; // "MyFeedHenryPolicy" should be replaced with policy
id you created
void (^success)(FHResponse *)=^(FHResponse * res){
    NSLog(@"parsed response %@ type=%@",res.parsedResponse,
[res.parsedResponse class]);
    if ([[res.parsedResponse valueForKey:@"status"]
isEqualToString:@"error"]){
        [self showMessage:@"Failed" message:%5Bres.parsedResponse
valueForKey:@"message"]];
    } else {

```

```

        [self showMessage:@"Success" message:res.rawValueAsString];
    }
};
void (^failure)(FHResponse *)=^(FHResponse* res){
    NSLog(@"parsed response %@ type=%@",res.parsedResponse,
[res.parsedResponse class]);
    [self showMessage:@"Failed" message:res.rawValueAsString];
};
[authRequest execAsyncWithSuccess:success AndFailure:failure];

```

認証ポリシーのタイプが OAuth の場合、UI コンポーネントが起動して OAuth プロバイダーの認証ページが表示されます。OAuth プロセスが終了するとこれを検知して自動的に閉じ、結果を success または failure 関数に渡します。これを有効にするには、アプリケーションの **UIViewController** インスタンスで **FHAuthRequest** インスタンスの **parentViewController** プロパティを設定します。

```

FHAuthReqeust * authRequest = [FH buildAuthRequest];
[authRequest authWithPolicyId:@"MyOAuthPolicy"]; //"MyOAuthPolicy" should
be replaced with policy id you created
authRequest.parentViewController = viewController; //Important, this will
enable the built-in OAuth hanlder
void (^success)(FHResponse *)=^(FHResponse * res){
    NSLog(@"parsed response %@ type=%@",res.parsedResponse,
[res.parsedResponse class]);
    if ([[res.parsedResponse valueForKey:@"status"]
isEqualToString:@"error"]) {
        [self showMessage:@"Failed" message:%5Bres.parsedResponse
valueForKey:@"message"]];
    } else {
        [self showMessage:@"Success" message:%5Bres.parsedResponse
JSONString]];
    }
};
void (^failure)(FHResponse *)=^(FHResponse* res){
    NSLog(@"parsed response %@ type=%@",res.parsedResponse,
[res.parsedResponse class]);
    [self showMessage:@"Failed" message:res.rawValueAsString];
};

[authRequest execAsyncWithSuccess:success AndFailure:failure]

```

parentViewController プロパティが設定されていない場合は、自身のコードでこれを処理することができます。例を示します。

```

FHAuthReqeust* authRequest = [FH buildAuthRequest];
[authRequest authWithPolicyId:@"MyOAuthPolicy"]; //"MyOAuthPolicy" should
be replaced with policy id you created
void (^success)(FHResponse *)=^(FHResponse * res){
    NSLog(@"parsed response %@ type=%@",res.parsedResponse,
[res.parsedResponse class]);
    //because the parentViewController is not set, the reponse will contain
a URL which should be used for user to login. Normally it should be
loaded into a WebView
    NSString* oauthUrl = [res.parsedResponse valueForKey:@"url"];
    NSURL* request = [NSURL URLWithString:oauthUrl];
    // load the url in a WebView, and then a series of redirects will

```

```

    happen
    // the last url will contain a string "status=complete"
    // and there will be a query parameter called "authResponse" in that
    url
    // the value of that parameter is the data returned from the OAuth
    provided (JSON stringified and URL encoded)
};
void (^failure)(FHResponse *)=^(FHResponse* res){
    NSLog(@"parsed response %@ type=%@", res.parsedResponse,
[res.parsedResponse class]);
    [self showMessage:@"Failed" message:res.rawResponseAsString];
};

[authRequest execAsyncWithSuccess:success AndFailure:failure];

```

iOS (Swift)

```

//Example to authenticate user using username and password
FH.auth("MyFeedHenryPolicy", userName: "me", password: "password",
completionHandler: { (response: Response, error: NSError?) -> Void in
    if let error = error {
        print("Error \(error)")
        return
    }
    if let response = response.parsedResponse as? [String: String]{
        if let status = response["status"] where status == "ok" {
            print("Response \(response)")
        } else if let status = response["status"] where status == "error" {
            let message = response["message"] ?? ""
            print("OAuth failed \(message)")
        }
    }
})

```

認証ポリシーのタイプが OAuth の場合、UI コンポーネントが起動して OAuth プロバイダーの認証ページが読み込まれます。OAuth プロセスが終了するとこれを検知して自動的に閉じ、結果を success または failure 関数に渡します。これを有効にするには、アプリケーションの UIViewController インスタンスで **AuthRequest** インスタンスの **parentViewController** プロパティを設定します。

```

let request = FH.authRequest("MyOAuthPolicy") // "MyOAuthPolicy" should be
replaced with policy id you created
request.parentViewController = viewController //Important, this will
enable the built-in OAuth handler
request.exec({ (response: Response, error: NSError?) -> Void in
    if let error = error {
        print("Error connecting \(error)")
        return
    }
    if let response = response.parsedResponse as? [String: String] {
        if let status = response["status"] where status == "ok" {
            print("Response \(response)")
        } else if let status = response["status"] where status == "error" {
            let message = response["message"] ?? ""

```

```

        print("OAuth failed \(message)")
    }
}
})

```

parentViewController プロパティが設定されていない場合は、自身のコードでこれ进行处理することができます。例を示します。

```

let request = FH.authRequest("MyOAuthPolicy") // "MyOAuthPolicy" should be replaced with policy id you created
request.exec({ (response: Response, error: NSError?) -> Void in
    if let error = error {
        print("Error connecting \(error)")
        return
    }
    if let response = response.parsedResponse as? [String: String] {
        if let status = response["status"] where status == "ok" {
            print("Response \(response)")
            // because the parentViewController is not set, the response will contain a URL which
            // should be used for user to login. Normally it should be loaded into a WebView
            if let urlString = response["url"] {
                let url = NSURL(string: urlString)
                // load the url in a WebView, and then a series of redirects will happen
                // the last url will contain a string "status=complete"
                // and there will be a query parameter called "authResponse" in that url
                // the value of that parameter is the data returned from the OAuth provided (JSON stringified and URL encoded).
            } else if let status = response["status"] where status == "error" {
                let message = response["message"] ?? ""
                print("OAuth failed \(message)")
            }
        }
    }
})

```

.NET (C#)

```

//Example to authenticate user using username and password
string authPolicy = "MyFeedHenryPolicy"; // "MyFeedHenryPolicy" should be replaced with policy id you created
string username = this.usernameField.Text;
string password = this.passwordField.Text;

FHResponse authRes = await FH.Auth(authPolicy, username, password);
if (null == authRes.Error)
{
    //user successfully logged in
}
else

```



```
{
    //login failed, show error
    ShowMessage(authRes.Error.Message);
}
```

認証ポリシーのタイプが OAuth の場合、UI コンポーネントが起動して OAuth プロバイダーの認証ページが読み込まれます。OAuth プロセスが終了するとこれを検知して自動的に閉じ、結果を success または failure 関数に渡します。

```
string authPolicy = "TestGooglePolicy"; //"TestGooglePolicy" should be
replaced with policy id you created
//When next line is executed, the user will be prompted with a new view
to
//allow them enter their credentials on the OAuth provider's login page,
//and the result will be returned in FHResponse
FHResponse res = await FH.Auth(authPolicy);
if (null == res.Error)
{
    //user successfully logged in
}
else
{
    //login failed, show error
    ShowMessage(res.Error.Message);
}
```

SDK が提供するデフォルトの OAuth ログインハンドラーを使用したくない場合は、独自の実装を提供することが可能です。**IOAuthClientHandlerService** インターフェース用の実装を作成し、以下のように使用します、

```
//create a new instance of the custom IOAuthClientHandlerService
IOAuthClientHandlerService authHandler = new MyOAuthHandler();
//create a new auth request
FHAuthRequest authRequest = new FHAuthRequest();
authRequest.SetAuthPolicyId(policyId);
//set the request to use the custome oauth handler
authRequest.SetOAuthHandler(authHandler);
FHResponse res = await authRequest.execAsync();
```

1.2. セッションの確認

重要

この機能を使用する際には、クライアントとクラウド SDK で以下のバージョンが使用されていることを確認してください。

- ✧ fh-js-sdk: >= 2.6.0
- ✧ fh-ios-sdk: >= 2.2.8
- ✧ fh-android-sdk: >= 2.2.0
- ✧ fh-dotnet-sdk: >= 1.2.0
- ✧ fh-mbaas-api: >=4.10.0

認証 API が返す **sessionToken** はデバイス上で保持され、後で自動的に全 **cloud API** コールに追加されます。

クライアント側では、新規 API がセッションの操作をサポートするために追加されます。

JavaScript

```
//To check if user is already authenticated
$fh.auth.hasSession(function(err, exist){
  if(err) {
    console.log('Failed to check session');
    return;
  }
  if(exist){
    //user is already authenticated
    //optionally we can also verify the session is actually valid from
    client. This requires network connection.
    $fh.auth.verify(function(err, valid){
      if(err){
        console.log('failed to verify session');
        return;
      }
      if(valid){
        console.log('session is valid');
      } else {
        console.log('session is not valid');
      }
    });
  } else {
    //user is not authenticated
  }
});

//When the user is logging out, the session should be cleared
$fh.auth.clearSession(function(err){
  });
```

Java

```
//To check if user is already authenticated
boolean exists = FHAuthSession.exists();
if (exists) {
    //user is already authenticated
    //optionally we can also verify the session is actually valid from
    client. This requires network connection.
    FHAuthSession.verify(new FHAuthSession.Callback() {
        @Override
        public void handleSuccess(final boolean isValid) {
            if (isValid) {
                //The session is valid, notify the application
                //You may now access the session token using
                FHAuthSession.getToken()
            } else {
                //The session is not valid. Clear the application's
                //state and authenticate again
            }
        }
    })

    @Override
    public void handleError(FHResponse resp) {
        //Something went wrong with the network call.
    }
}, false);

} else {
    //Not logged in, notify the application.
}
```

プラットフォームからクライアントをログアウトするには、**FHAuthSession.clear(boolean synchronous)** を使用します。このメソッドはネットワークアクセスを実行することに留意してください。メインの looper から呼び出す場合は、**synchronous** 引数を **false** に設定して、ネットワーク操作が looper スレッドをブロックしないようにし、Android が **NetworkOnMainThreadException** をスローすることを防ぎます。

```
FHAuthSession.clear(false);
```

```
//To check if user is already authenticated
BOOL hasSession = [FH hasAuthSession];
if(hasSession) {
    //optionally we can also verify the session is acutally valid from
    client. This requires network connection.
    [FH verifyAuthSessionWithSuccess:nil AndFailure:nil];
}
//When the user is logging out, the session should be cleared
[FH clearAuthSessionWithSuccess:nil AndFailure:nil];
```

.NET (C#)

```
//To check if user is already authenticated
```

```

FHAuthSession session = FH.GetAuthSession();
Boolean exists = session.Exists();
//optionally we can also verify the session is actually valid from client. This requires network connection.
if(exists) {
    bool valid = await session.Verify();
}

//When the user is logging out, the session should be cleared
session.Clear();

```

fh-mbaas-api モジュールでは、リクエストの **sessionToken** を検証するミドルウェアも提供されています。これは、リクエストが認証済みユーザーからのものかどうかの判断を容易にします。必要となるのは、以下のものだけです。

```

var mbaasExpress = require('fh-mbaas-api').mbaasExpress();
var express = require('express');
var router = new express.Router();
//This will protect the router and only accept requests from authenticated users.
//If a sessionToken is valid, you can choose to cache it so that it doesn't need to be checked again.
router.use(mbaasExpress.fhauth({cache: true, expire: 60*60}));

```

1.3. 独自認証プロバイダーの作成

mBaaS 認証ポリシータイプ が導入されたことで、独自の認証プロバイダーを作成することが可能になっています。

これは、以下を実行する **mBaaS サービス** を作成することで可能になります。

※ 認証の実行

※ **sessionToken** キーを含む JSON 応答の回答

上記の説明にあったように、クライアント SDK により(**X-FH-SESSIONTOKEN** ヘッダー経由で) **sessionToken** がすべてのクラウドコールに追加されます。ただし、**sessionToken** は mBaaS サービスが生成していることから、fh-mbaas-api で提供されているセッション確認のミドルウェアは機能しなくなります。**sessionToken** の値を確認するために独自のミドルウェアを提供する必要があります。しかし、これは、リクエストオブジェクトにユーザー情報を設定するなど、確認プロセス中にリクエストにさらなる情報を追加できることにもなります。

第2章 \$FH.CLOUD

```
$fh.cloud(options, success, failure);
```

AJAX を使用してクラウドアプリ内に定義した **すべての** クラウド URL を呼び出します。例えば、Express を使用してクラウドアプリ内にエンドポイントを定義する場合、\$fh.act ではなくこの API を使用してください。

サポートされるプラットフォーム

- ✧ JavaScript SDK
 - Cordova
 - Web Apps
- ✧ Android SDK
- ✧ iOS Objective-C SDK
- ✧ iOS Swift SDK
- ✧ .NET SDK
 - Windows
 - Xamarin

詳細なバージョン情報については、[Supported Configurations](#) (英語) を参照してください。

2.1. 例

JavaScript

```
$fh.cloud({
  "path": "/api/v1/user/create", //only the path part of the url, the
  // host will be added automatically
  "method": "POST", //all other HTTP methods are supported as well. For
  // example, HEAD, DELETE, OPTIONS
  "contentType": "application/json",
  "data": { "username": "testuser"}, //data to send to the server
  "timeout": 25000 // timeout value specified in milliseconds. Default:
  // 60000 (60s)
}, function(res) {
  // Cloud call was successful. Alert the response
  alert('Got response from cloud:' + JSON.stringify(res));
}, function(msg,err) {
  // An error occurred during the cloud call. Alert some debugging
  // information
  alert('Cloud call failed with error message:' + msg + '. Error
  properties:' + JSON.stringify(err));
});
```

Android (Java)

```

//build the request object with request path, method, headers and data
Header[] headers = new Header[1];
headers[0] = new BasicHeader("contentType", "application/json");
//The request should have a timeout of 25 seconds, 10 is the default
FHHttpClient.setTimeout(25000);
FHCloudRequest request = FH.buildCloudRequest("/api/v1/user/create",
"POST", headers, new JSONObject().put("username", "testuser"));
//the request will be executed asynchronously
request.executeAsync(new FHActCallback() {
    @Override
    public void success(FHResponse res) {
        //the function to execute if the request is successful
        try{
            //process response data
        } catch(Exception e){
            Log.e(TAG, e.getMessage(), e);
        }
    }

    @Override
    public void fail(FHResponse res) {
        //the function to execute if the request is failed
        Log.e(TAG, res.getErrorMessage(), res.getError());
    }
});

```

iOS (Objective-C)

```

NSDictionary * headers = [NSDictionary
dictionaryWithObject:@"application/json" forKey:@"contentType"];
NSDictionary * data = [NSDictionary dictionaryWithObject:@"testuser"
forKey:@"username"];
FHCloudRequest * action = (FHCloudRequest *) [FH
buildCloudRequest:@"/api/v1/user/create" WithMethod:@"POST"
AndHeaders:headers AndArgs:data];
// change timeout (default value: 60s)
action.requestTimeout = 25.0;
[action execAsyncWithSuccess:^(FHResponse * actRes){
    //the actRes will contain 10 tweets about "feedhenry"
    //the JSON response from the cloud will be parsed to NSDictionary
    automatically
    NSDictionary * resData = actRes.parsedResponse;
    // ...
} AndFailure:^(FHResponse * actFailRes){
    //if there is any error, you can check the rawResponse string
    NSLog(@"Failed to read tweets. Response = %@",
actFailRes.rawResponse);
}
];

```

iOS (Swift)

```

FH.cloud("/api/v1/user/create",
args: ["testuser": "username"],

```

```

        completionHandler: {(resp: Response, error: NSError?) -> Void in
            if let error = error {
                print("Cloud Call Failed, \(error)")
                return
            }
            print("Success \(resp.parsedResponse)")
        })
    })

```

.NET (C#)

```

var headers = new Dictionary<string, string> {{"contentType",
"application/json"}};
var data = new Dictionary<string, object> {"username", "testuser"}};

//change the timeout to 60 seconds, default is 30 seconds
FH.Timeout = TimeSpan.FromSeconds(60);
var response = await FH.Cloud("/api/v1/user/create", "POST", headers,
data);
if(null == response.Error)
{
    //no error occurred, the request is successful
    var rawResponseData = response.RawResponse;
    //you can get it as JObject (require Json.Net library)
    var resJson = response.GetResponseAsJsonObject();
    //process response data
}
else
{
    //error occurred during the request, deal with it.
    //More information can be access from response.Error.InnerException
}

```

第3章 \$FH.GETCLOUDURL

```
$fh.getCloudURL();
```

現行のクライアントアプリの通信先となっているクラウドアプリの URL を取得します。SDK が初期化された後に使用してください。

クラウドアプリの URL を取得したら、他の HTTP/AJAX クライアントを使用してクラウドアプリと通信することができます。ただし、この方法では一部のデータがリクエスト内にないため、プラットフォームが提供するアナリティクスサービスをアプリで使用できないというマイナス面があります。また、エンドポイントが確保された場合は、API キーを提供しないと呼び出すことができません。

これを容易にするため、SDK は API がリクエストのパラメーターまたはヘッダーとしてメタデータを提供するようにしています。このデータを取得し、リクエストのボディまたはヘッダーとしてリクエストに追加するだけで済みます。詳細は、[\\$fh.getFHParams](#) を参照してください。

サポートされるプラットフォーム

JavaScript SDK

- Cordova
- Web Apps

Android SDK

iOS Objective-C SDK

iOS Swift SDK

.NET SDK

- Windows
- Xamarin

詳細なバージョン情報については、[Supported Configurations](#) (英語) を参照してください。

3.1. 例

JavaScript

```
var cloud_url = $fh.getCloudURL();
```

Android (Java)

```
String cloudAppHost = FH.getCloudHost();
```

iOS (Objective-C)

```
NSString * cloudAppHost = [FH getCloudHost];
```

.NET (C#)

```
string cloudAppHost = FH.GetCloudHost();
```


第4章 \$FH.GETFHPARAMS

```
$fh.getFHParams();
```

このメソッドは、各クラウドリクエストについて FH SDK が追加したメタデータを返します。クラウドアプリとの通信に別のライブラリーを使用することを選択した場合は、このデータをリクエストボディまたはヘッダーに追加してください。クラウドアプリの url を取得する方法については、[\\$fh.getCloudURL](#) を参照してください。

通常は、メタデータを修正する必要は全くありません。参考のために、メタデータに含まれているキーを値を以下に示します。

- ✧ appid: アプリの id
- ✧ appkey: アプリの api キー
- ✧ projectid: プロジェクトの id
- ✧ cuid: クライアント用に生成された一意の id
- ✧ destination: iOS、Android、ウェブなど、クライアントアプリが稼働しているプラットフォーム
- ✧ sdk_version: sdk のバージョン
- ✧ connectiontag: アプリの connectiontag

メタデータがリクエストボディで送信される場合は、特別キー "__fh" の値にしてください。下記の例を参照してください。

SDK によってはリクエストヘッダーとしてメタデータを提供するものもあります。この場合は、各ヘッダー名は "X-FH-appid" のように "X-FH-<meta data name>" という形式になります。

サポートされるプラットフォーム

- ✧ JavaScript SDK
 - Cordova
 - Web Apps
- ✧ Android SDK
- ✧ iOS Objective-C SDK
- ✧ iOS Swift SDK
- ✧ .NET SDK
 - Windows
 - Xamarin

詳細なバージョン情報については、[Supported Configurations](#) (英語) を参照してください。

4.1. 例

JavaScript

■

```
var fhparams = $fh.getFHParams();  
//then it should be added to your request body under the key "__fh"  
body.__fh = fhparams;
```

Android (Java)

JSON オブジェクトとしてリクエストのメタデータを取得するには、以下を使用します。

```
JSONObject fhParams = FH.getDefaultParams();
```

HTTP ヘッダーとしてメタデータを取得することもできます。

```
Header[] fhParamHeaders = FH.getDefaultParamsAsHeaders(null);
```

iOS (Objective-C)

NSDictionary としてリクエストのメタデータを取得するには、以下を使用します。

```
NSDictionary * fhParams = [FH getDefaultParams];
```

NSDictionary の HTTP ヘッダーとしてメタデータを取得することもできます。

```
NSDictionary * fhParamHeaders = [FH getDefaultParamsAsHeaders];
```

.NET (C#)

Dictionary としてリクエストのメタデータを取得するには、以下を使用します。

```
IDictionary <string, object> fhParams = FH.GetDefaultParams();
```

Dictionary の HTTP ヘッダーとしてメタデータを取得することもできます。

```
IDictionary <string, string> fhParamHeaders =  
FH.GetDefaultParamsAsHeaders();
```

第5章 \$FH.FORMS

サポートされるプラットフォーム

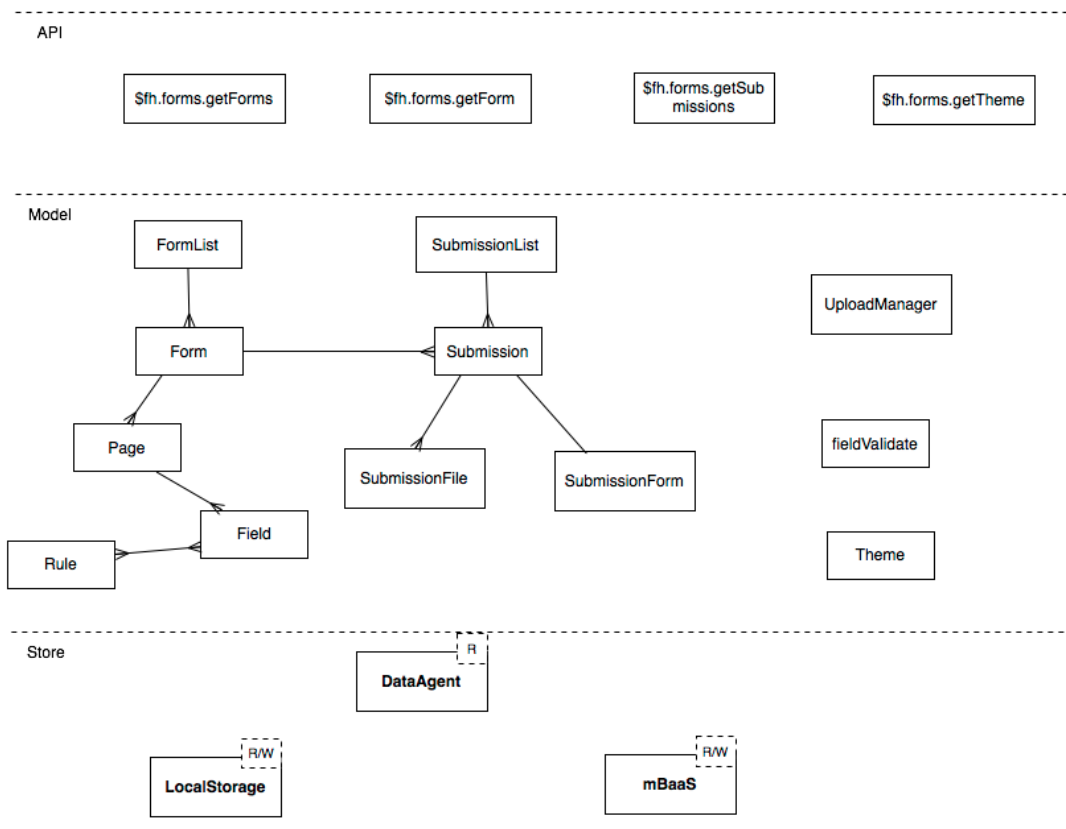
JavaScript SDK

- Cordova
- Web Apps

詳細なバージョン情報については、[Supported Configurations](#) (英語) を参照してください。

5.1. SDK の構造

5.1.1. コア



5.2. \$FH.FORMS.INIT

```
$fh.forms.init(params, callback);
```

5.2.1. 詳細

appForm SDK を初期化します。現時点では `params` は使用されないなので、空の JSON オブジェクトとして渡します。初期化が完了すると、`callback` が呼び出されます。

5.2.2. 例

```
var params = {};
$fh.forms.init(params, function(err) {
  if (err) console.error(err);

  // Forms initialised ok if no error
});
```

5.3. \$FH.FORMS.GETFORMS

```
$fh.forms.getForms(params, callback);
```

5.3.1. 詳細

フォームモデルの配列を取得します。'fromRemote' パラメーターが true に設定されている場合はサーバーから、false に設定されている場合はローカルストレージから読み込むことができます。

5.3.2. 例

```
var params = {
  "fromRemote": true
};

$fh.forms.getForms(params, function(err, forms){
  if(err) console.error(err);

  // forms is an instance of $fh.forms.models.forms
  // See Models section for details of its API
  // for example, getFormsList() returns an array of Forms models

  var formsList = forms.getFormsList();
  console.log(formsList);
});
```

5.4. \$FH.FORMS.GETFORM

```
$fh.forms.getForm(params, callback);
```

5.4.1. 詳細

指定された ID に基づいてフォームを取得します。

5.4.2. 例

```
var params = {
  "fromRemote" : true,
```

```

    "formId" : "1234"
  };

  $fh.forms.getForm(params, function(err, form){
    if(err) console.error(err);

    var formName = form.getName();
    var formDesc = form.getDescription();
    console.log('Form Name: ', formName, 'Form Desc: ', formDesc);
  });

```

5.5. \$FH.FORMS.GETTHEME

```
$fh.forms.getTheme(params, callback);
```

5.5.1. 詳細

テーマを取得します。'css' パラメーターが true に設定されている場合は、css が返されます。false に設定されている場合は、Theme オブジェクトが返されます。

5.5.2. 例

```

var params = {
  "css" : true //if set to true, returns css.
};

$fh.forms.getTheme(params, function(err, theme){
  if(err) console.error(err);

  console.log(theme);
});

```

5.6. \$FH.FORMS.GETSUBMISSIONS

```
$fh.forms.getSubmissions(options, callback);
```

5.6.1. 詳細

正常に完了した提出の一覧を返します。

5.6.2. 例

```

var params = {};

$fh.forms.getSubmissions(params, function (err, submissions) {
  if (err) console.error(err);

  console.log('Array of completed submissions', submissions);
});

```

5.7. \$FH.FORMS.DOWNLOADSUBMISSION

5.7.1. 詳細

提出のダウンロードを開始します。

この API には 2 つの形式があります。

5.7.2. Example: Callback Passed

この例では、**\$fh.forms.downloadSubmission** 関数へのコールバックを渡すと、全ファイルを含む提出のダウンロードが完了した時点でのみ、コールバックが呼び出されるようになります。

```
var params = {
  'submissionId': "<< ID of the submission stored in the cloud >>"
};

//Downloading the submission.
//If the submission has already been downloaded to local memory, it will
be loaded from there instead of downloading from the cloud.
$fh.forms.downloadSubmission(params, function(err, downloadedSubmission){
  if(err){
    return console.error(err);
  }

  //The form that the submission was submitted against is sent back with
the submission from the cloud.
  var formSubmittedAgainst =
downloadedSubmission.getFormSubmittedAgainst();
  var formId = downloadedSubmission.getFormId();
});
```

5.7.3. 例: コールバックが渡されない場合

この例では、コールバックを渡さないため、提出のダウンロードがキューに登録されます。その後、グローバルイベントリスナーによってダウンロードの進捗が監視されます。

```
var params = {
  'submissionId': "<< ID of the submission stored in the cloud >>"
};

//A global event listener for all submission 'downloaded' events
$fh.forms.on('submission:downloaded', function(remoteSubmissionId){
  //The form that the submission was submitted against is sent back with
the submission from the cloud.
  var formSubmittedAgainst = this.getFormSubmittedAgainst();
  var formId = this.getFormId();
});

//A global event listener for all submission 'error' events
$fh.forms.on('submission:error', function(errorMessage){
  var localId = this.getLocalId();

  console.error(errorMessage + " for submission with local ID " +
```

```

    localId);
  });

  //Downloading the submission.
  //If the submission has already been downloaded to local memory, it will
  be loaded from there instead of downloading from the cloud.
  $fh.forms.downloadSubmission(params);

```

5.8. \$FH.FORMS.LOG

```

$fh.forms.log.e(message); //logs an error
$fh.forms.log.w(message); //logs a warning
$fh.forms.log.d(message); //logs a debug message
$fh.forms.log.l(message); //logs success

```

5.8.1. 詳細

以下の 4 タイプのログがあります。

- ✧ エラー: `$fh.forms.log.e('error');`
- ✧ 警告: `$fh.forms.log.w('warning');`
- ✧ デバッグ: `$fh.forms.log.d('debug');`
- ✧ ログ: `$fh.forms.log.l('successful log');`

5.8.2. 例

```

var params = {
  'fromRemote' : true
};

$fh.forms.getForms(params, function(err, formsList){
  if(error) $fh.forms.log.e(err); //log error

  console.log('Lit of forms: ', formsList.getFormsList());
  $fh.forms.log.l('Forms loaded successfully'); //log successfully loading
  of forms
});

```

5.9. \$FH.FORMS.GETLOGS

```

$fh.forms.log.getLog();

```

5.9.1. 詳細

ログ情報を返します。

```

$fh.forms.getSubmissions({}, function (err, submissions) {
  if (err) {

```

```

    console.log('Error loading submissions', err);
  } else {
    $fh.forms.log.l('Array loaded successfully') //Recorded log
    console.log('Array of completed submissions', submissions);

    var logRecords = $fh.forms.log.getLog();
    console.log('Log Record', logRecords); //Prints out an array of all
logs
  }
});

```

5.10. グローバルイベント

5.10.1. 詳細

\$fh.forms API は、\$fh.forms API で発生する全イベントにグローバルイベントエミッターを提供します。

イベント名は、**model:eventname** という形式になります。例えば、**Submission** モデルは **submitted** イベントを生成し、グローバルイベント名は **submission:submitted** となります。このイベントは、提出が提出されてアップロードされると起動します。

5.10.2. \$fh.forms.on

```

$fh.forms.on("submission:progress", function(progressJSON) {
    //See Submission Model Progress For progressJSON Definition
});

```

5.10.3. \$fh.forms.once

```

$fh.forms.once("submission:submitted", function(submissionId) {
    //Note: this refers to the individual submission that emitted the
submitted event.
    assert.equal(this.getRemoteSubmissionId(), submissionId);
});

```

5.11. \$FH.FORMS.MODELS.FORMS

5.11.1. 詳細

これは、\$fh.forms.getForms() が返すフォームモデルの一覧です。以下の関数は、返されるフォームモデルのエイで呼び出すことができます。

5.11.2. \$fh.forms.models.forms.clearAllForms

```

$fh.forms.models.forms.clearAllForms(cb);

```


5.11.2.1. 詳細

すべてのローカルのフォームをクリアします。



注記

この関数は、現在実装中です。

5.11.2.2. 例

```
formsList.clearAllForms(function(err) {
  if (err) {
    console.log('Error deleting forms', err);
  } else {
    console.log('Reloading forms list', formsList.getFormsList());
    //will return empty list
  }
});
```

5.11.3. \$fh.forms.models.forms.isFormUpdated

```
$fh.forms.models.forms.isFormUpdated(formModel);
```

5.11.3.1. 詳細

特定のフォームモデルが最新のものが確認します。

5.11.3.2. 例

```
var model = new $fh.forms.models.Form();
var updated = $fh.forms.models.forms.isFormUpdated(model);
console.log(updated);
```

5.11.4. \$fh.forms.models.forms.getFormMetaById

```
$fh.forms.models.forms.getFormMetaById(formId)
```

5.11.4.1. 詳細

formId によるメタオブジェクトを取得します。

5.11.4.2. 例

```
var exampleFormId = '1234';

var params = {
  'fromRemote' : true
};
```

```
$fh.forms.getForms(params, function(err, formsList){
  if(err) console.error(err);

  var formDetails = formsList.getFormMetaById(exampleFormId); //gets meta
  object from forms list based on id
  console.log(formDetails);
});
```

5.11.5. \$fh.forms.models.forms.refresh

```
$fh.forms.models.forms.refresh(fromRemote, cb);
```

5.11.5.1. 詳細

ローカルまたはリモートからフォーム一覧モデルを読み取ります。モデルがそれまでに存在していない場合は、自動的にローカルストレージに保存します。

5.11.5.2. 例

```
formsList.refresh(true, function (err) { //if fromRemote == true, forms
are read from server.
// If false, reads from local storage
  if (err) {
    console.log('Error refreshing form', err);
  } else {
    console.log('Refreshed form list', formsList); //prints newly
refreshed list
  }
});
```

5.11.6. \$fh.forms.models.forms.clearLocal

```
$fh.forms.models.forms.clearLocal(cb);
```

5.11.6.1. 詳細

モデルをローカルストレージから削除しますが、RAM からは削除しません。

5.11.6.2. 例

```
formsList.clearLocal(function (err) {
  if (err) {
    console.log('Error occurred clearing forms from local storage', err);
  } else {
    console.log('Reloading forms', formsList.getFormsList());
  }
});
```

5.11.7. \$fh.forms.models.forms.getFormsList

```
$fh.forms.models.forms.getFormsList();
```

5.11.7.1. 詳細

フォームのメタデータを含むアレイを取得します。

5.11.7.2. 例

```
var params = {
  'fromRemote': true
};

$fh.forms.getForms(params, function (err, formsList) {
  if (err) console.error(err);

  var forms = formsList.getFormsList();
  console.log(forms);
});
```

5.11.8. \$fh.forms.models.forms.size

```
$fh.forms.models.forms.size();
```

5.11.8.1. 詳細

保存されているフォームの数を取得します。

5.11.8.2. 例

```
var params = {
  "fromRemote": true
};

$fh.forms.getForms(params, function (err, formsList) {
  if (err) console.error(err);

  var numOfForms = formsList.size();
  console.log(numOfForms);
});
```

5.12. \$FH.FORMS.MODELS.FORM

5.12.1. 詳細

これはフォームモデルです。forms.getFormsList() を呼び出すとフォームモデル一覧が返されます。

5.12.2. \$fh.forms.models.Form.constructor()

```
$fh.forms.models.Form(params, cb);
```

5.12.2.1. 詳細

フォームオブジェクトを構築します。フォームの定義が読み込まれると、コールバックします。

5.12.2.2. 例

```
var params = {
  "formId": "1234",
  "fromRemote": true
};

$fh.forms.getForm(params, function (err, form) {
  if (err) console.error(err);

  // new form model
  console.log(form);
});
```

5.12.3. \$fh.forms.models.Form.getLastUpdate()

```
form.getLastUpdate();
```

5.12.3.1. 詳細

サーバーの最新のタイムスタンプ更新を取得します。

5.12.3.2. 例

```
var params = {
  "formId": "1234",
  "fromRemote": true
};

$fh.forms.getForm(params, function (err, form) {
  if (err) console.error(err);

  var lastUpdate = form.getLastUpdate();
  console.log(lastUpdate);
});
```

5.12.4. \$fh.forms.models.Form.getPageModelList()

```
form.getPageModelList();
```

5.12.4.1. 詳細

このフォームに関連付けられているページモデルのエイを取得します。

5.12.4.2. 例

```
var params = {
  "formId": "1234",
  "fromRemote": true
};

$fh.forms.getForm(params, function (err, form) {
  if (err) console.error(err);

  var pageList = form.getPageModelList();
  console.log('Array of pages associated with this form', pageList);
});
```

5.12.4.3. fh.forms.models.Form.getRuleEngine()

```
form.getRuleEngine();
```

5.12.4.4. 詳細

form() にアタッチされているルールエンジンを取得します。

5.12.4.5. 例

```
var params = {
  "formId": 1234,
  "fromRemote": true
};

$fh.forms.getForm(params, function (err, form) {
  if (err) console.error(err);

  var ruleEngine = form.getRuleEngine();
  console.log(ruleEngine);
});
```

5.12.5. \$fh.forms.models.Form.getName()

```
form.getName();
```

5.12.5.1. 詳細

フォーム名を取得します。

5.12.5.2. 例

```
var params = {
```

```
    "formId": "1234",
    "fromRemote": true
  };

  $fh.forms.getForm(params, function (err, form) {
    if (err) console.error(err);

    var formName = form.getName();
    console.log(formName);
  });
```

5.12.6. \$fh.forms.models.Form.getDescription()

```
form.getDescription();
```

5.12.6.1. 詳細

フォームの説明を取得します。

5.12.6.2. 例

```
var params = {
  "formId": "1234",
  "fromRemote": true
};

$fh.forms.getForm(params, function (err, form) {
  if (err) console.error(err);

  var formDescription = form.getDescription();
  console.log(formDescription);
});
```

5.12.7. \$fh.forms.models.Form.getFormId()

```
form.getFormId();
```

5.12.7.1. 詳細

フォームの ID を取得します。

5.12.7.2. 例

```
var params = {
  "formId": "1234",
  "fromRemote": true
};

$fh.forms.getForm(params, function (err, form) {
  if (err) console.error(err);
```

```
var formId = form.getId();  
console.log(formId);  
});
```

5.12.8. \$fh.forms.models.Form.getFieldModelById()

```
form.getFieldModelById(fieldId);
```

5.12.8.1. 詳細

フォーム ID ごとのフィールドモデルを取得します。

5.12.8.2. 例

```
var params = {  
  "formId": "1234",  
  "fromRemote": true,  
  "fieldId": "123"  
};  
  
$fh.forms.getForm(params, function (err, form) {  
  if (err) console.error(err);  
  
  var fieldModel = form.getFieldModelById(fieldId);  
  console.log(fieldModel);  
});
```

5.12.9. \$fh.forms.models.Form.newSubmission()

```
form.newSubmission();
```

5.12.9.1. 詳細

このフォームの新規提出モデルを初期化します。

5.12.9.2. 例

```
form.newSubmission(); //creates a new submission
```

5.12.10. \$fh.forms.models.Form.removeFromCache()

```
form.removeFromCache();
```

5.12.10.1. 詳細

メモリーキャッシュ (singleton) から現行のフォームモデルを削除します。

5.12.10.2. 例

```
form.removeFromCache();
```

5.12.11. \$fh.forms.models.Form.refresh()

```
form.refresh([fromRemote], cb);
```

5.12.11.1. 詳細

ローカルまたはリモートからフォームモデルを読み取ります。モデルがそれまでに存在していない場合は、自動的にローカルストレージに保存します。

5.12.11.2. 例

```
form.refresh(true, function (err) {  
  if (err) {  
    console.log('Error refreshing', err);  
  } else {  
    console.log('Refreshed page');  
    //refresh successful  
  }  
});
```

5.12.12. \$fh.forms.models.Form.clearLocal()

```
form.clearLocal(cb)
```

5.12.12.1. 詳細

ローカルに保存されたフォームを削除します。

5.12.12.2. 例

```
foundForm.clearLocal(function (err) {  
  if (err) {  
    console.log('Error removing form');  
  } else {  
    //form cleared successfully  
  }  
});
```

5.13. \$FH.FORMS.MODELS.PAGE

5.13.1. 詳細

以下はページモデルで呼び出し可能な関数です。

5.13.2. \$fh.forms.models.Page.setVisible()

```
page.setVisible(isVisible);
```

5.13.2.1. 詳細

このページモデルを表示するかどうかを設定します。'isVisible' のブール値が true または false に設定されているかによって 'visible' または 'hidden' を生成します。

5.13.2.2. 例

```
page.setVisible(true) //Boolean value to determine whether page is set to visible or not.
```

5.13.3. \$fh.forms.models.Page.getName()

```
page.getName();
```

5.13.3.1. 詳細

ページ名を取得します。

5.13.3.2. 例

```
var pageList = foundForm.getPageModelList(); //Iterates through all pages of a returned form and prints out page names
for (var page = 0; page < pageList.length; page++) {
  var currentPage = pageList[page];
  console.log('Name of current page is: ', currentPage.getName());
}
```

5.13.4. \$fh.forms.models.Page.getDescription()

```
page.getDescription();
```

5.13.4.1. 詳細

ページの説明を取得します。

5.13.4.2. 例

```
page.getDescription();
```

5.13.5. \$fh.forms.models.Page.getFieldModelList()

```
page.getFieldModelList();
```

5.13.5.1. 詳細

この ページに関連付けられているフィールドモデルを取得します。

5.13.5.2. 例

```

var pageList = form.getPageModelList(); //Retrieves all pages of a form

for (var page = 0; page < pageList.length; page++) { //Iterates through all pages
    var currentPage = pageList[page];
    var pageFields = currentPage.getFieldModelList(); //Retrieves all fields on a page
    console.log(pageFields); //Lists all fields
}

```

5.13.6. \$fh.forms.models.Page.getFieldModelById()

```

page.getFieldModelById(fieldId);

```

5.13.6.1. 詳細

特定のフィールドモデルを取得します。このページにフィールドモデルがある必要はありません。Form.getFieldModelById(fieldId) のエイリアス。

5.13.6.2. 例

```

var fieldId = '1234';

page.getFieldModelById(fieldId); //Returns Field model

```

5.14. \$FH.FORMS.MODELS.FIELD

5.14.1. 詳細

ページモデル上で getFieldModelList() 関数を呼び出すと、フィールドオブジェクト一覧が返されます。

```

pageOne.getFieldModelList();

```

以下はフィールドモデルの各属性にアクセスするために呼び出しが可能な関数です。

5.14.2. \$fh.forms.models.Field.isRequired()

```

currentField.isRequired();

```

5.14.2.1. 詳細

フィールドが必須の場合、true が返されます。

5.14.2.2. 例

```
currentField.isRequired(); //will return true if the field is required
```

5.14.3. \$fh.forms.models.Field.isRepeating()

```
field.isRepeating();
```

5.14.3.1. 詳細

フィールドが繰り返しフィールドの場合は true が、繰り返しではない場合は false が返されます。

5.14.3.2. 例

```
field.isRepeating();
```

5.14.4. \$fh.forms.models.Field.getType()

```
field.getType();
```

5.14.4.1. 詳細

フィールドのタイプを返します。

5.14.4.2. 例

```
field.getType();
```

5.14.5. \$fh.forms.models.Field.getName()

```
field.getName();
```

5.14.5.1. 詳細

フィールド名を返します

```
console.log(field.getName()); //prints name of field
```

5.14.6. \$fh.forms.models.Field.getCode()

```
field.getCode();
```

5.14.6.1. 詳細

存在する場合は、フィールドの Field Code を返します。Studio でフィールドに Field Code が割り当てられていない場合は、**null** が返されます。

5.14.7. \$fh.forms.models.Field.getHelpText()

```
field.getHelpText();
```

5.14.7.1. 詳細

フィールド指示のテキストが返されます。

5.14.7.2. 例

```
field.getHelpText();
```

5.14.8. \$fh.forms.models.Field.validate(inputValue)

```
field.validate(inputValue, callback(err,res))
```

5.14.8.1. 詳細

検証が成功すると inputValue オブジェクトが、失敗するとエラーメッセージが返されます。

5.14.8.2. 例

```
var field = form.getFieldModelById(fieldId);
var inputValue = elem.value; //Value of an element such as text field, numberfield etc

field.validate(inputValue, function (err, res) {
  if (err) {
    console.log('Validation Error', err);
  } else {
    console.log('Validation Successful', res);
  }
});
```

5.14.9. \$fh.forms.models.Field.getRules()

```
field.getRules();
```

5.14.9.1. 詳細

フィールドの関連付けられている rule オブジェクトの配列が返されます。

5.14.9.2. 例

```
field.getRules();
```

5.14.10. \$fh.forms.models.Field.getCheckboxOptions()

```
field.getCheckboxOptions();
```

5.14.10.1. 詳細

チェックボックス選択肢の 배열が返されます。



注記

チェックボックスフィールドでのみ有効です。

5.14.10.2. 例

```
if (field.getType() == 'checkboxes') {  
  console.log('Checkbox options: ', field.getCheckBoxOptions());  
}
```

5.14.11. \$fh.forms.models.Field.getRadioOption()

```
field.getRadioOption();
```

5.14.11.1. 詳細

ラジオボックスのオプションを返します。



注記

ラジオボタンフィールドでのみ有効です。

5.14.11.2. 例

```
if (field.getType() == 'radio') {  
  console.log('Radio options: ', field.getRadioOption());  
}
```

5.15. \$FH.FORMS.MODELS.SUBMISSIONS

5.15.1. \$fh.forms.models.submissions.getSubmissions()

```
$fh.forms.models.submissions.getSubmissions();
```

5.15.1.1. 詳細

提出メタデータ一覧を返します。

5.15.1.2. 例

```
submissions.getSubmissions();
```

5.15.2. \$fh.forms.models.submissions.getSubmissionMetaList()

```
submissions.getSubmissionMetaList();
```

5.15.2.1. 詳細

提出メタデータ一覧を返します。

5.15.2.2. 例

```
submissions.getSubmissionMetaList();
```

5.15.3. \$fh.forms.models.submissions.findById(formId)

```
submissions.findById(formId);
```

5.15.3.1. 詳細

指定されたフォームのメタデータを取得します。

5.15.3.2. 例

```
var formId = '53146bf95a133733451cd35b';

$fh.forms.getSubmissions({}, function (err, submissions) {
  if (err) {
    console.log('Error loading submissions', err);
  } else {
    console.log('Array of completed submissions', submissions);
    var foundSubmissions = submissions.findById(formId);
    console.log('Array of submissions for specified form: ',
foundSubmissions);
  }
});
```

5.15.4. \$fh.forms.models.submissions.clear(cb)

```
submissions.clear(function(err))
```

5.15.4.1. 詳細

このモデルおよびローカルストレージから提出メタデータ一覧をクリアします。

5.15.4.2. 例

```
$fh.forms.getSubmissions({}, function(err, submissions) {  
  if (err) {  
    console.log('Error loading submissions', err);  
  } else {  
    console.log('Array of completed submissions', submissions);  
    submissions.clear(function(err) {  
      if (err) console.err(err);  
    });  
  }  
});
```

5.15.5. \$fh.forms.models.submissions.getDrafts()

```
submissions.getDrafts()
```

5.15.5.1. 詳細

提出の drafts() を返します。

5.15.5.2. 例

```
submissions.getDrafts();
```

5.15.6. \$fh.forms.models.submissions.getPending()

```
submissions.getPending()
```

5.15.6.1. 詳細

保留中の提出を返します。

5.15.6.2. 例

```
submissions.getPending();
```

5.15.7. \$fh.forms.models.submissions.getSubmitted()

```
submissions.getSubmitted()
```

5.15.7.1. 詳細

提出済みの提出を返します。

5.15.7.2. 例

```
submissions.getSubmitted();
```

5.15.8. \$fh.forms.models.submissions.getError()

```
submissions.getError()
```

5.15.8.1. 詳細

エラーのあった提出を返します。

5.15.8.2. 例

```
submissions.getError();
```

5.15.9. \$fh.forms.models.submissions.getInProgress()

```
submissions.getInProgress()
```

5.15.9.1. 詳細

現在進行中の提出を返します。

5.15.9.2. 例

```
submissions.getInProgress();
```

5.15.10. \$fh.forms.models.submissions.getSubmissionByMeta(meta,cb)

```
submissions.getSubmissionByMeta(meta, function(err, res))
```

5.15.10.1. 詳細

提出一覧モデルから提出メタデータごとの提出モデルを取得します。

5.15.10.2. 例

```
var params = submissions.getSubmitted()[0];

submissions.getSubmissionByMeta(params, function (err, submission){
  if(err) console.error(err);
```



```
    console.log('Returned Submission',submission);
  });
```

5.15.11. \$fh.forms.models.submissions.getSubmissionByLocalId(localId,cb)

```
submissions.getSubmissionByLocalId(localId, function(err,
submissionModel){})
```

5.15.11.1. 詳細

提出の **Local ID** ごとの提出モデルを取得します。

5.15.11.2. 例

```
var localId = "sublocalid1";

submissions.getSubmissionByLocalId(localId, function (err, submission){
  if(err) console.error(err);

  console.log('Returned Submission',submission);
});
```

5.15.12.

\$fh.forms.models.submissions.getSubmissionByRemoteId(remoteId,cb)

```
submissions.getSubmissionByRemoteId(remoteId, function(err,
submissionModel){})
```

5.15.12.1. 詳細

提出の **Remote ID** ごとの提出モデルを取得します。

5.15.12.2. 例

```
var remoteId = "subremoteid1";

submissions.getSubmissionByRemoteId(remoteId, function (err, submission){
  if(err) console.error(err);

  console.log('Returned Submission',submission);
}
```

5.16. \$FH.FORMS.MODELS.SUBMISSION

5.16.1. 詳細

提出モデルにはユーザー入力と関連するメタ情報が含まれています。提出一覧で `.getSubmissions()` 関数を呼び出すと、提出モデル一覧が返されます。

```
submissions.getSubmissions(); //double check
```

以下は提出モデルで呼び出し可能な関数です。

5.16.2. \$fh.forms.models.Submission.saveDraft(cb)

```
submission.saveDraft(cb);
```

5.16.2.1. 詳細

現行の提出をドラフト / ローカルストレージに保存します。

5.16.2.2. 例

```
currentSubmission.saveDraft(function (err) {  
  if (err) {  
    console.log(err);  
  } else {  
    console.log('Draft saved to local storage');  
  }  
});
```

5.16.3. \$fh.forms.models.Submission.submit()

```
submission.submit(cb)
```

5.16.3.1. 詳細

現行の提出を提出します。アップロードのタスクを作成します。

5.16.3.2. 例

```
currentSubmission.submit(function (err) {  
  console.log(!err);  
  if (err) {  
    console.log(err);  
  }  
});  
  
currentSubmission.on("submit", function () {  
  /* Upload the submission. */  
  currentSubmission.upload(function (err) {  
    if (err) {  
      console.log(err);  
    }  
  });  
});
```

5.16.4. \$fh.forms.models.Submission.getStatus()

```
submission.getStatus();
```

5.16.4.1. 詳細

提出の現在のステータスを返します。

5.16.4.2. 例

```
submission.getStatus();
```

5.16.5. \$fh.forms.models.Submission.addComment()

```
submission.addComment(message, [username]);
```

5.16.5.1. 詳細

ユーザーが現行提出にコメントを追加できるようにします。

5.16.5.2. 例

```
submission.addComment('test message', 'test user');
```

5.16.6. \$fh.forms.models.Submission.getComments()

```
submission.getComments()
```

5.16.6.1. 詳細

現行提出についてのコメントの配列を返します。

5.16.6.2. 例

```
submission.addComment('test message', 'test user');  
  
console.log(submission.getComments()); //will return an array containing  
the above comment
```

5.16.7. \$fh.forms.models.Submission.removeComment(timestamp)

```
submission.removeComment(timestamp);
```

5.16.7.1. 詳細

タイムスタンプで現行提出からコメントを削除します。

5.16.7.2. 例

```
submission.removeComment(timestamp);
```

5.16.8. \$fh.forms.models.Submission.addInputValue(params,cb)

```
$fh.forms.models.Submission(params, callback(err, res))
```

5.16.8.1. 詳細

フィールドの提出に値を追加します。これは入力の値を検証し、これに失敗すると文字列としてエラーメッセージが返されます。transaction モードの場合は、提出にユーザー入力の値を即座に追加せず、temp 変数が追加されます。transaction モードでない場合は、提出に即座に入力値が追加されます。

5.16.8.2. 例

```
var params = {
  "fieldId": '53146c1f04e694ec1ad715b6',
  "value": 'New example text'
};
currentSubmission.addInputValue(params, function(err, res) {
  if (err) console.error(err);

  console.log('Newly added input: ', res);
});
```

5.16.9. \$fh.forms.models.Submission.startInputTransaction()

```
submission.startInputTransaction()
```

5.16.9.1. 詳細

ユーザー入力の処理を開始します。既に開始している場合は、一時入力を中断します。

5.16.9.2. 例

```
submission.startInputTransaction();
```

5.16.10. \$fh.forms.models.Submission.endInputTransaction(isSucceed)

```
$fh.forms.models.Submission.endInputTransaction(isSucceed) // 'isSucceed'  
is a boolean value.
```

5.16.10.1. 詳細

処理を終了します。処理が成功すると、一時入力をアップロードする提出にコピーします。失敗すると、一時入力を破棄します。'isSucceed' はブール値です。'true' が関数に渡されると、処理が完

了してユーザー入力が一時的ストレージから提出に移動されます。'false' が渡される場合は、一時的な値が破棄され、処理が完了されません。

5.16.10.2. 例

```
submission.startInputTransaction();

var params = {
  "fieldId": '53146c1f04e694ec1ad715b6',
  "value": 'Example text'
};

submission.addInputValue(params, function(err, res) {
  if (err) {
    console.log('Error adding input', err);
    submission.endInputTransaction(false); //Transaction failed. New values are not added to submission.
  } else {
    console.log('Updated value: ', res);
    submission.endInputTransaction(true); //End input transaction. New value is added to submission.
  }
});
```

5.16.11. \$fh.forms.models.Submission.reset()

```
submission.reset();
```

5.16.11.1. 詳細

この提出からすべてのユーザー入力を削除します。

5.16.11.2. 例

```
submission.reset();
```

5.16.12. \$fh.forms.models.Submission.getForm(cb)

```
submission.getForm(function (err, form))
```

5.16.12.1. 詳細

提出に関連するフォームオブジェクトを返します。

5.16.12.2. 例

```
submission.getForm(function(err, form) {
  if (err) console.error(err);
```

```
    console.log('Form associated with submission: ', form);
  });
```

5.16.13. \$fh.forms.Submission.removeFieldValue(fieldId, [index])

```
submission.removeFieldValue(fieldId, [index]) //Index is only specified
when referencing repeated fields
```

5.16.13.1. 詳細

ID に基づいて特定フィールドから値を削除します。

5.16.13.2. 例

```
var exampleFieldId = '1234';

submission.removeFieldValue(exampleFieldId);
```

5.16.14. \$fh.forms.Submission.getInputValueByFieldId(fieldId,cb)

```
submission.getInputValueByFieldId(field, function(err, res))
```

5.16.14.1. 詳細

フィールドに関連する入力値を取得します。

```
var fieldId = '1234';

currentSubmission.getInputValueByFieldId(fieldId, function(err, res) {
  if (err) console.error(err);

  console.log('Field value: ', res);
});
```

5.16.15. 提出のイベント

提出 モデルは、提出プロセスを移動する際にイベントを生成します。イベント起動時に関数が実行されると、この オブジェクトは常に、イベントを生成した 提出 オブジェクトを参照します。

5.16.15.1. inprogress: The Submission Is In The Process Of Uploading.

```
submission.on('inprogress', function(uploadTask) {
  var self = this;
  uploadTask.submissionModel(function(err, submissionModel) {
    //The 'this' parameter in the event refers the submission model
that emitted the event.
    assert.strictEqual(self, submissionModel);
  });
});
```

5.16.15.2. error: There Was An Error Uploading The Submission.

```
submission.on('error', function(errorMessage) {
  console.error(errorMessage);
  assert.equal(errorMessage, this.getErrorMessage());
});
```

5.16.15.3. savedraft: The Submission Was Saved As A Draft

```
submission.on('savedraft', function() {
  //This is the local ID of the submission that emitted the 'savedraft'
  event.
  assert.isString(this.getLocalId());
});
```

5.16.15.4. validationerror: There Was A Validation Error When Making A Submission.

このエラーはローカルでのみ生成されます。これはサーバー側の検証エラーではありません。

```
submission.on('validationerror', function(validationObject) {
  //This is the local ID of the submission that emitted the 'savedraft'
  event.
  assert.isString(false, validationObject.valid);
});
```

検証オブジェクト

```
{
  valid: < true / false > ,
  < fieldid1 > : {
    valid: < true / false > ,
    errorMessages: [
      "Validation Error Message 1",
      "Validation Error Message 2"
    ]
  },
  ...,
  < fieldidN > : {
    valid: < true / false > ,
    errorMessages: [
      "Validation Error Message 1",
      "Validation Error Message 2"
    ]
  }
}
```

5.16.15.5. submit: The Submission Is Valid And Can Now Be Uploaded.

提出はローカル検証をパスしました。クラウドにアップロードする準備ができました。

```
submission.on('submit', function() {
  //Valid Submission, it can be uploaded now or at any other time
  this.upload(function(err, uploadTask) {
```

```

    ...
  });
});

```

5.16.15.6. submitted: The Submission Is Valid And Has Completed Uploading All Data.

```

submission.on('submitted', function(remoteSubmissionId) {
  assert.equal(remoteSubmissionId, this.getRemoteSubmissionId());
});

```

5.16.15.7. queued: The Submission JSON Definition Has Been Uploaded. Proceeding To Upload Any Files.

この時点で リモート の提出 ID がクラウド側に割り当てられています。これでクライアントとクラウドの両方の側で提出が有効と認識されます。

```

submission.on('queued', function(remoteSubmissionId) {
  assert.equal(remoteSubmissionId, this.getRemoteSubmissionId());
});

```

5.16.15.8. progress: The Progress For A Submission Has Been Incremented.

```

submission.on('progress', function(progressJSON) {
  //The Current Progress Of The Submission
});

```

5.16.15.8.1. progressJSON

```

{
  'formJSON': false, //Boolean specifying if the submission JSON has
  been uploaded.
  'currentFileIndex': 0,
  'totalFiles': 3,
  'totalSize': 54321, //Size in Bytes of the entire submission
  'uploaded': 12345, //Size, in Bytes already uploaded
  'retryAttempts': 1, //Number of times the submission has been tried.
  'submissionTransferType': < upload / download > , //Is the submission
  being uploaded or downloaded.
  'submissionRemoteId': "remoteSubmissionID1234", //The remote
  submission ID if it is available.
  'submissionLocalId': "localSubmissionID1234" //The local submission
  ID
}

```

5.16.15.9. downloaded: The Submission Has Completed Downloading (Only Used When Downloading Submissions.)

提出がモバイルクライアントデバイスにダウンロードされる際に、全ファイルが含まれます。

```

submission.on('downloaded', function(remoteSubmissionId) {

```



```
    assert.equal(remoteSubmissionId, this.getRemoteSubmissionId());  
  });
```

第6章 \$FH.HASH

```
$fh.hash(options, success, failure);
```

文字列のハッシュ値を生成します。

サポートされるプラットフォーム

✳ JavaScript SDK

- Cordova
- Web Apps

詳細なバージョン情報については、[Supported Configurations](#) (英語) を参照してください。

6.1. 例

```
var options = {
  "algorithm": "SHA256", // Can be MD5 | SHA1 | SHA256 | SHA512
  "text": "Need more widgets. Add some columns." // Text to hash
};

$fh.hash(options, function (res) {
  // The generated hash value
  var hashvalue = res.hashvalue;
}, function(msg) {
  // Error message for why the hash failed
  console.error(msg);
});
```

第7章 \$FH.INIT

クライアントアプリが対応するクラウドアプリと通信するためには、クライアント SDK はクラウドを呼び出す前に初期化される必要があります。

初期化プロセスでは、サーバーを呼び出してプラットフォーム上のアプリが有効かどうかを検証し、アプリのプロパティが取得されます。このプロセスが完了するまでは、他の API メソッドを実行することはできません。初期化プロセスは非同期で実行されるので、メインのアプリスレッドがブロックされることはありません。

サポートされるプラットフォーム

✧ JavaScript SDK

- Cordova
- Web Apps

✧ Android SDK

✧ iOS Objective-C SDK

✧ iOS Swift SDK

✧ .NET SDK

- Windows
- Xamarin

詳細なバージョン情報については、[Supported Configurations](#) (英語) を参照してください。

7.1. 例

JavaScript

アプリが読み込まれると Javascript SDK が初期化されるので、init API を呼び出す必要がありません。このため、たとえばアプリが \$fh.cloud を読み込むと、すぐに使用することができます。アプリがいつ初期化を完了したかを確認するには、コールバックを "fhinit" に登録します。

```
$fh.on("fhinit", function(err, host){
    if(err){
        //Init has failed due to some error. Normally this is due to no
        network connection.
    } else {
        //The js sdk has initialized. The host value will be the URL of the
        cloud app the client app will communicate with.
    }
});
```

Android (Java)

```
FH.init(this, new FHActCallback() {
    public void success(FHResponse pRes) {
        // Init okay, free to use FHActRequest
```

```

        // Note: pRes will be null. To get the cloud host url, use
        FH.getCloudHost.
        String cloudAppUrl = FH.getCloudHost();
    }

    public void fail(FHResponse pRes) {
        // Init failed
        Log.e("FHInit", pRes.getErrorMessage(), pRes.getError());
    }
});

```

iOS (Objective-C)

```

#import <FH/FH.h>
#import <FH/FHResponse.h>

- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    // Override point for customization after application launch.

    // Call a cloud side function when init finishes
    void (^success)(FHResponse *)=^(FHResponse * res) {
        // Initialisation is now complete, you can now make FHCloud requests
        // Note: res will be nil. To get the cloud host url, use [FH
        getCloudHost].
        NSString * cloudAppUrl = [FH getCloudHost];
        NSLog(@"SDK initialised OK");
    };

    void (^failure)(id)=^(FHResponse * res){
        NSLog(@"Initialisation failed. Response = %@", res.rawValue);
    };

    //View loaded, init the library
    [FH initWithSuccess:success AndFailure:failure];

    return YES;
}

```

iOS (Swift)

```

import FeedHenry

func application(application: UIApplication,
didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) ->
Bool {
    // Override point for customization after application launch.
    FH.init { (resp:Response, error: NSError?) -> Void in
        if let error = error {
            print("Initialisation failed. Response = \(error)")
            return
        }
        // Initialisation is now complete, you can now make FH.cloud calls.
    }
}

```

```
        print("SDK initialised OK: \"(resp.parsedResponse)\")
    }
    return true
}
```

.NET (C#)

```
try
{
    bool initd = await FHClient.Init();
    if(initd) {
        //Initialisation is successful
        //To get the cloud host url, use FH.GetCloudHost().
        string cloudAppUrl = FH.GetCloudHost();
    }
}
catch(FHException e)
{
    //Initialisation failed, handle exception
}
```

第8章 \$FH.MBAAS

```
$fh.mbaas(options, success, failure);
```

MBaaS サービスのエンドポイントを呼び出します。

サポートされるプラットフォーム

✳ JavaScript SDK

■ Web Apps

詳細なバージョン情報については、[Supported Configurations](#) (英語) を参照してください。

8.1. 例

```
$fh.mbaas({
  "service": "db", //the MBaaS service name.
  "params": {}, //json object to send to the MBaaS service
  "timeout": 25000 // timeout value specified in milliseconds. Default:
60000 (60s)
}, function(res) {
  // Cloud call was successful. Alert the response
  alert('Got response from cloud:' + JSON.stringify(res));
}, function(msg,err) {
  // An error occured during the cloud call. Alert some debugging
information
  alert('Cloud call failed with error message:' + msg + '. Error
properties:' + JSON.stringify(err));
});
```

第9章 \$FH.PUSH

```
$fh.push(onNotification(e), regSuccessHandler, regErrorHandler(err),
pushConfig)
```

サーバーに登録し、プッシュ通知の受信を開始します。

サポートされるプラットフォーム

- ✧ JavaScript SDK
 - Cordova
- ✧ Android SDK
- ✧ iOS Objective-C SDK
- ✧ iOS Swift SDK
- ✧ .NET SDK
 - Windows
 - Xamarin

詳細なバージョン情報については、[Supported Configurations](#) (英語) を参照してください。

9.1. 例

Javascript

```
// register with the server to start receiving push notifications
$fh.push(function(e) {
  // on android we could play a sound, if we add the Media plugin
  if (e.sound && (typeof Media !== 'undefined')) {
    var media = new Media("/android_asset/www/" + e.sound);
    media.play();
  }

  if (e.coldstart) {
    // notification started the app
  }

  // show text content of the message
  alert(e.alert);

  // only on iOS
  if (e.badge) {
    push.setApplicationIconBadgeNumber(successHandler, e.badge);
  }
}, function() {
  // successfully registered
}, function(err) {
  // handle errors
}, {
```

```
// optional filtering criteria
alias: "user@example.com",
categories: ["Curling", "Hurling"]
});
```

パラメーター

- ※ **onNotification(e)** 関数 - 着信通知のハンドラーで、以下のプロパティを含みます。
 - **alert**、**sound**、**badge** - [クラウド API](#) の **message** オブジェクト内の対応オプションと同等のセマンティクス。
 - **coldstart** - 受信した通知がアプリケーション起動の原因である場合、true。
 - **payload** - クラウド API の **message.userData** オプションに対応。
- ※ **regSuccessHandler** 関数 - 登録が成功すると起動するコールバック。
- ※ **regErrorHandler(err)** 関数 - エラーのために登録が失敗した場合に起動するコールバック。その後、文字列引数として渡されます。
- ※ **pushConfig** オブジェクト - オプションの設定で、受信した通知を条件のセットを用いてフィルタリング可能とします。条件のセマンティクスと使用方法についての情報は、[通知の送信](#) を参照してください。利用可能なプロパティは以下の通りです。
 - **alias** 文字列 - 固有ユーザーの識別子
 - **categories** アレイ - カテゴリー一覧

Android (Java)

サーバーに登録し、プッシュ通知の受信を開始します。

```
FH.pushRegister(new FHActCallback() {
    @Override
    public void success(FHResponse fhResponse) {
        startActivity(...);
    }

    @Override
    public void fail(FHResponse fhResponse) {
        Toast.makeText(getApplicationContext(),
            fhResponse.getErrorMessage(),
            Toast.LENGTH_SHORT).show();
        finish();
    }
});
```

メッセージを処理するために **MessageHandler** の実装を作成します。これを有効にするために、**MessageHandler** を **RegistrarManager** に登録します。詳細情報は、Aerogear ドキュメンテーションの [Receiving notifications](#) (英語) を参照してください。

```
import org.jboss.aerogear.android.unifiedpush.MessageHandler;
import org.jboss.aerogear.android.unifiedpush.gcm.UnifiedPushMessage;
```



```
public class MyMessageHandler implements MessageHandler {
    @Override
    public void onMessage(Context context, Bundle bundle) {
        String message = bundle.getString(UnifiedPushMessage.ALERT_KEY);
    }
}
```

パラメーター

メッセージハンドラーの **onMessage** メソッドに渡される **Bundle** 内のキーです。

- ※ **UnifiedPushMessage.ALERT_KEY** - 通知テキストで、[クラウド API](#) の **message.alert** プロパティに対応。
- ※ クラウド API の **message.userData** とともに送信されるすべてのキー。JSON オブジェクトを含む文字列として表示。

iOS (Objective-C)

サーバーに登録し、プッシュ通知の受信を開始します。

```
- (void)application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken {
    [FH pushRegister:deviceToken andSuccess:^(FHResponse success) {
        NSLog(@"Push registration successful");
    } andFailure:^(FHResponse failed) {
        NSLog(@"Push registration Error: %@", failed.error);
    }];
}
```

処理メッセージ:

```
- (void)application:(UIApplication *)application
didReceiveRemoteNotification:(NSDictionary *)userInfo {
    NSLog(@"message received: %@", userInfo[@"aps"][@"alert"][@"body"]);
}
```

パラメーター

- ※ **userInfo** NSDictionary
 - **aps** NSDictionary
 - **alert** NSString/NSDictionary - 通知テキストで、[クラウド API](#) の **message.alert** プロパティに対応。iOS ドキュメンテーションの [Local and Remote Notification Programming Guide](#) にあるように、タイプは **NSString** または **NSDictionary** のいずれかにすることが可能
 - クラウド API の **message.userData** とともに送信されるすべてのキー。

.NET (C#)

サーバーに登録し、プッシュ通知の受信を開始します。

```
protected override async void OnNavigatedTo(NavigationEventArgs e)
{
    try
    {
        await FHClient.Init();
        // register with the server to start receiving push notifications
        FH.RegisterPush(HandleNotification);
    }
    catch (Exception ex)
    {
        new MessageDialog("Error", ex.Message).ShowAsync();
    }
    ...
}
```

処理メッセージ:

```
private void HandleNotification(object sender, PushReceivedEvent e)
{
    Console.WriteLine(e.Args.Message);
}
```

パラメーター

✎ **PushReceivedEvent.Args**

- **Message** 文字列 - 通知テキストで、[クラウド API](#) の **message.alert** プロパティーに対応。
- **Data** IDictionary<string, string> - クラウド API 内の **message.userData** に渡される値のディクショナリー。

第10章 \$FH.SEC

```
$fh.sec(options, success, failure);
```

キーペアの生成およびデータの暗号化と暗号解除。

サポートされるプラットフォーム

✳ JavaScript SDK

- Cordova
- Web Apps

詳細なバージョン情報については、[Supported Configurations](#) (英語) を参照してください。

10.1. 例

```
// Generate a new Key
var options = {
  "act": "keygen",
  "params": {
    "algorithm": "AES", // Only AES supported
    "keysize": "128" // 128 or 256
  }
};

$fh.sec(options, function(res) {
  // The algorithm used for the generation
  var algorithm = res.algorithm;

  // The generated key (hex format)
  var secretkey = res.secretkey;

  // The generated initial vector (hex format)
  var iv = res.iv;
}, function(code) {
  // Error code. One of:
  // bad_act    : invalid action type
  // no_params  : params missing
  // no_params_algorithm : missing algorithm in params
  console.error(code);
});

// Encrypt data
var options = {
  "act": "encrypt",
  "params": {
    // The data to be encrypted
    "plaintext": "Need a new page to start on",
    // The secret key used to do the encryption. (Hex format)
    "key": secretkey,
    // The algorithm used for encryption. Should be either "RSA" or "AES"
```

```
        "algorithm": "AES",
        // IV only required if algorithm is "AES"
        "iv": iv
    }
};
$fh.sec(options, function (res) {
    // The encrypted data (hex format)
    var ciphertext = res.ciphertext;
}, function (code) {
    // Error code. One of:
    // bad_act    : invalid action type
    // no_params  : params missing
    // no_params_algorithm : missing algorithm in params
    console.error(code);
});

// Decrypt data
var options = {
    "act": "encrypt",
    "params": {
        // The data to be decrypted
        "ciphertext":
"dc87f02ae3fce8149d1e2b97a747581f8bc7c0c01b435a87ba56661b1ae",
        // The secret key used to do the decryption. (Hex format)
        "key": secretkey,
        // The algorithm used for decryption. Should be either "RSA" or "AES"
        "algorithm": "AES",
        // IV only required if algorithm is "AES"
        "iv": iv
    }
};
$fh.sec(options, function (res) {
    // The decrypted data (hex format)
    var plaintext = res.plaintext;
}, function (code) {
    // Error code. One of:
    // bad_act    : invalid action type
    // no_params  : params missing
    // no_params_algorithm : missing algorithm in params
    console.error(code);
});
```

第11章 \$FH.SYNC

Sync API は、クライアントアプリとバックエンドのデータストア間におけるデータ同期に回復性の高いメカニズムを提供します。Sync API 使用時には、クライアントアプリは `$fh.cloud` コールではなく、Sync API ですべてのデータ操作を実行します。

データセットを制限するためにバックエンドデータストアに渡されるクエリパラメーターとともに、データセットは **manage** を呼び出し、データセットの一意の ID を指定することで、Sync サービスの制御下に置かれます。

Sync クライアントは、新規レコードを受信した場合や更新がバックエンドにコミットされた場合など、データの状態が変更されるとイベントを使用してアプリに通知します。Sync Service についての詳細な説明は、[データ同期フレームワーク](#) を参照してください。

サポートされるプラットフォーム

- ✳ JavaScript SDK
 - Cordova
 - Web Apps
- ✳ Android SDK
- ✳ iOS Objective-C SDK
- ✳ iOS Swift SDK
- ✳ .NET SDK
 - Windows
 - Xamarin

詳細なバージョン情報については、[Supported Configurations](#) (英語) を参照してください。

11.1. \$FH.SYNC.INIT

```
$fh.sync.init(options);
```

11.1.1. 詳細

クライアントデータ sync servicew を初期化します。

11.1.2. 例

JavaScript

```
$fh.sync.init({
  // How often to synchronize data with the cloud, in seconds.
  // Optional. Default: 10
  "sync_frequency": 10,

  // Should local changes be synchronized to the cloud immediately, or
```

```
should they wait for the next synchronization interval.  
// Optional. Default: true  
"auto_sync_local_updates": true,  
  
// Should a notification event be triggered when loading or saving to  
client storage fails.  
// Optional. Default: true  
"notify_client_storage_failed": true,  
  
// Should a notification event be triggered when a synchronization  
cycle with the server has been started.  
// Optional. Default: true  
"notify_sync_started": true,  
  
// Should a notification event be triggered when a synchronization  
cycle with the server has been completed.  
// Optional. Default: true  
"notify_sync_complete": true,  
  
// Should a notification event be triggered when an attempt was made to  
update a record while offline.  
// Optional. Default: true  
"notify_offline_update": true,  
  
// Should a notification event be triggered when an update failed due  
to data collision.  
// Optional. Default: true  
"notify_collision_detected": true,  
  
// Should a notification event be triggered when an update was applied  
to the local data store.  
// Optional. Default: true  
"notify_local_update_applied": true,  
  
// Should a notification event be triggered when an update failed for a  
reason other than data collision.  
// Optional. Default: true  
"notify_remote_update_failed": true,  
  
// Should a notification event be triggered when an update was applied  
to the remote data store.  
// Optional. Default: true  
"notify_remote_update_applied": true,  
  
// Should a notification event be triggered when a delta was received  
from the remote data store.  
// Optional. Default: true  
"notify_delta_received": true,  
  
// Should a notification event be triggered when a delta was received  
from the remote data store for a record.  
// Optional. Default: true  
"notify_record_delta_received": true,  
  
// Should a notification event be triggered when the synchronization  
loop failed to complete.
```

```

    // Optional. Default: true
    "notify_sync_failed": true,

    // Should log statements be written to console.log. Will be useful for
    debugging.
    // Optional. Default: false
    "do_console_log": false,

    // How many synchronization cycles to check for updates on crashed in-
    flight updates.
    // Optional. Default: 10
    "crashed_count_wait" : 10,

    // If crashed_count_wait limit is reached, should the client retry
    sending the crashed in flight pending records.
    // Optional. Default: true
    "resend_crashed_updates" : true,

    // Is the background synchronization with the cloud currently active.
    If this is set to false, the synchronization loop will not start
    automatically. You need to call startSync to start the synchronization
    loop.
    // Optional. Default: true
    "sync_active" : true,

    // Storage strategy to use for the underlying client storage framework
    Lawnchair. Valid values include 'dom', 'html5-filestorage', 'webkit-
    sqlite', 'indexed-db'.
    // Multiple values can be specified as an array and the first valid
    storage option will be used.
    // Optional. Default: 'html5-filestorage'
    "storage_strategy" : "html5-filestorage",

    // Amount of space to request from the HTML5 filestorage API when
    running in browser
    // Optional. Default: 50 * 1024 * 1024
    "file_system_quota" : 50 * 1024 * 1024,

    // If the app has legacy custom cloud sync function (the app implemented
    the data CRUDL operations in main.js file in FH V2 apps), it should be
    set to true. If set to false, the default MBaaS sync implementation will
    be used. When set to null or undefined, a check will be performed to
    determine which implementation to use.
    // Optional. Default: null
    "has_custom_sync" : null,

    // ios only. If set to true, the file will be backed by icloud.
    // Optional. Default: false
    "icloud_backup" : false
  });

```

Android (Java)

```
FHSyncConfig syncConfig = new FHSyncConfig();
```

```
// Should local changes be synchronized to the cloud immediately, or
// should
// they wait for the next synchronization interval.
// Optional. Default: false
syncConfig.setAutoSyncLocalUpdates(false);

// How many synchronization cycles to check for updates on crashed in-
// flight
// updates.
// Optional. Default: 10
syncConfig.setCrashCountWait(10);

// Should a notification event be triggered when loading or saving to
// client
//storage fails.
// Optional. Default: false
syncConfig.setNotifyClientStorageFailed(false);

// Should a notification event be triggered when a delta was received
// from the
//remote data store.
// Optional. Default: false
syncConfig.setNotifyDeltaReceived(false);

// Should a notification event be triggered when an update was applied to
// the local
//data store.
// Optional. Default: false
syncConfig.setNotifyLocalUpdateApplied(false);

// Should a notification event be triggered when an attempt was made to
// update a
//record while offline.
// Optional. Default: false
syncConfig.setNotifyOfflineUpdate(false);

// Should a notification event be triggered when an update was applied to
// the remote
//data store.
// Optional. Default: false
syncConfig.setNotifyRemoteUpdateApplied(false);

// Should a notification event be triggered when a synchronization cycle
// with the
//server has been started.
// Optional. Default: false
syncConfig.setNotifySyncStarted(false);

// Should a notification event be triggered when the synchronization loop
// failed to complete.
// Optional. Default: false
syncConfig.setNotifySyncFailed(false);

// Should a notification event be triggered when a synchronization cycle
// with the
// server has been completed.
```



```

// Optional. Default: false
syncConfig.setNotifySyncComplete(false);

// Should a notification event be triggered when an update failed due to
// data collision.
// Optional. Default: false
syncConfig.setNotifySyncCollisions(false);

// Should a notification event be triggered when an update failed for a
// reason other
//than data collision.
// Optional. Default: false
syncConfig.setNotifyUpdateFailed(false);

// If the limit set in setCrashCountWait is reached, should the client
// retry sending the crashed in-flight pending records.
// Optional. Default: true
syncConfig.setResendCrashedUpdates(true);

// How often to synchronize data with the cloud, in seconds.
// Optional. Default: 10
syncConfig.setSyncFrequency(10);

// If the app has legacy custom cloud sync function (the app implemented
// the data
//CRUDL operations in main.js file in FH V2 apps), it should be set to
//true. If set
//to false, the default MBaaS sync implementation will be used.
// Optional. Default: false
syncConfig.setUseCustomSync(false);

syncClient = FHSyncClient.getInstance();
syncClient.init(appContext, syncConfig, new FHSyncListener() {
    /**The
    * is discussed
    */
});

```

iOS (Swift)

```

let conf = FHSyncConfig()

// How often to synchronize data with the cloud, in seconds.
// Optional. Default: 10
conf.syncFrequency = 10

// Should local changes be synchronized to the cloud immediately, or
// should they wait for the next synchronization interval.
// Optional. Default: true
conf.autoSyncLocalUpdates = true

// Should a notification event be triggered when loading or saving to
// client storage fails.

```

```
// Optional. Default: false
conf.notifyClientStorageFailed = true

// Should a notification event be triggered when a synchronization cycle
// with the server has been started.
// Optional. Default: false
conf.notifySyncStarted = true

// Should a notification event be triggered when a synchronization cycle
// with the server has been completed.
// Optional. Default: false
conf.notifySyncCompleted = true

// Should a notification event be triggered when an attempt was made to
// update a record while offline.
// Optional. Default: false
conf.notifyOfflineUpdate = true

// Should a notification event be triggered when an update failed due to
// data collision.
// Optional. Default: false
conf.notifySyncCollision = true

// Should a notification event be triggered when an update was applied to
// the local data store.
// Optional. Default: false
conf.notifyLocalUpdateApplied = true

// Should a notification event be triggered when an update failed for a
// reason other than data collision.
// Optional. Default: false
conf.notifyRemoteUpdateFailed = true

// Should a notification event be triggered when an update was applied to
// the remote data store.
// Optional. Default: false
conf.notifyRemoteUpdateApplied = true

// Should a notification event be triggered when a delta was received
// from the remote data store.
// Optional. Default: false
conf.notifyDeltaReceived = true

// Should a notification event be triggered when the synchronization loop
// failed to complete.
// Optional. Default: false
conf.notifySyncFailed = true

// Should log statements be written to console.log. Will be useful for
// debugging.
// Optional. Default: false
conf.debug = true

// How many synchronization cycles to check for updates on crashed in-
// flight updates.
// Optional. Default: 10
```

```

conf.crashCountWait = 10

// If crashCountWait limit is reached, should the client retry sending
the crashed in flight pending records.
// Optional. Default: true
conf.resendCrashedUpdates = true

// If the app has legacy custom cloud sync function (the app implemented
the data CRUDL operations in main.js file in FH V2 apps), it should be
set to true. If set to false, the default MBaaS sync implementation will
be used. When set to null or undefined, a check will be performed to
determine which implementation to use.
// Optional. Default: false
conf.hasCustomSync = false

// iOS only. If set to YES, the file will be backed by icloud.
// Optional. Default: false
conf.icloud_backup = false

syncClient = FHSyncClient(config: conf)

```

iOS (Objective-C)

```

FHSyncConfig* conf = [[FHSyncConfig alloc] init];

// How often to synchronize data with the cloud, in seconds.
// Optional. Default: 10
conf.syncFrequency = 10;

// Should local changes be synchronized to the cloud immediately, or
should they wait for the next synchronization interval.
// Optional. Default: YES
conf.autoSyncLocalUpdates = YES;

// Should a notification event be triggered when loading or saving to
client storage fails.
// Optional. Default: NO
conf.notifyClientStorageFailed = YES;

// Should a notification event be triggered when a synchronization cycle
with the server has been started.
// Optional. Default: NO
conf.notifySyncStarted = YES;

// Should a notification event be triggered when a synchronization cycle
with the server has been completed.
// Optional. Default: NO
conf.notifySyncCompleted = YES;

// Should a notification event be triggered when an attempt was made to
update a record while offline.
// Optional. Default: NO
conf.notifyOfflineUpdate = YES;

// Should a notification event be triggered when an update failed due to

```

```
data collision.
// Optional. Default: NO
conf.notifySyncCollision = YES;

// Should a notification event be triggered when an update was applied to
the local data store.
// Optional. Default: NO
conf.notifyLocalUpdateApplied = YES;

// Should a notification event be triggered when an update failed for a
reason other than data collision.
// Optional. Default: NO
conf.notifyRemoteUpdateFailed = YES;

// Should a notification event be triggered when an update was applied to
the remote data store.
// Optional. Default: NO
conf.notifyRemoteUpdateApplied = YES;

// Should a notification event be triggered when a delta was received
from the remote data store.
// Optional. Default: NO
conf.notifyDeltaReceived = YES;

// Should a notification event be triggered when the synchronization loop
failed to complete.
// Optional. Default: NO
conf.notifySyncFailed = YES;

// Should log statements be written to console.log. Will be useful for
debugging.
// Optional. Default: NO
conf.debug = YES;

// How many synchronization cycles to check for updates on crashed in-
flight updates.
// Optional. Default: 10
conf.crashCountWait = 10;

// If crashCountWait limit is reached, should the client retry sending
the crashed in flight pending records.
// Optional. Default: YES
conf.resendCrashedUpdates = YES;

// If the app has legacy custom cloud sync function (the app implemented
the data CRUDL operations in main.js file in FH V2 apps), it should be
set to true. If set to false, the default MBaaS sync implementation will
be used. When set to null or undefined, a check will be performed to
determine which implementation to use.
// Optional. Default: NO
conf.hasCustomSync = NO;

// iOS only. If set to YES, the file will be backed by icloud.
```

```
// Optional.Default: NO
conf.icloud_backup = NO;

FHSyncClient* syncClient = [[FHSyncClient alloc] initWithConfig:conf];
```

.NET (C#)

```
var client = FHSyncClient.GetInstance();
var config = new FHSyncConfig();

/// How often to synchronize data with the cloud, in seconds.
/// Default Value : 10
config.SyncFrequency = 10;

/// Should local changes be synchronized to the cloud immediately, or
/// should they wait for the next synchronization interval.
/// Default value : true
config.AutoSyncLocalUpdates = true;

/// How many synchronization cycles to check for updates on crashed in-
/// flight updates.
/// Default value : 10
config.CrashedCountWait = 10;

/// If CrashedCountWait limit is reached, should the client retry sending
/// the crashed in flight pending records.
/// Default value : true
config.ResendCrashedUpdated = true;

/// Is the background sync with the cloud currently active. If this is
/// set to false, the sync loop will not start automatically. You need to
/// call Start to start the synchronization loop.
/// Default value : true
config.SyncActive = true;

/// Set whether to use a legacy FH V2 sync Cloud App, the MBaaS sync
/// service,
/// or automatically select.
/// Values are SyncCloudType.Auto, SyncCloudType.Legacy,
/// SyncCloudType.Mbbas
/// Default value : Auto
config.SyncCloud = SyncCloudType.Auto;

client.Initialise(config);
```

11.2. \$FH.SYNC.NOTIFY

```
$fh.sync.notify(callback(data));
```

11.2.1. 詳細

sync サービスにクライアントへの通知がある場合、callback 関数が起動するように登録する。

11.2.2. 例

JavaScript

```
$fh.sync.notify(function(event) {
    // The dataset that the notification is associated with
    var dataset_id = event.dataset_id;

    // The unique identifier that the notification is associated with.
    // This will be the unique identifier for a record if the notification
    is related to an individual record,
    // or the current hash of the dataset if the notification is associated
    with a full dataset
    // (for example, sync_complete)
    var uid = event.uid;

    // Optional free text message with additional information
    var message = event.message;

    // The notification message code
    var code = event.code;
    /* Codes:
    * client_storage_failed: Loading or saving to client storage failed.
    This is a critical error and the Sync Client will not work properly
    without client storage.
    * sync_started: A synchronization cycle with the server has been
    started.
    * sync_complete: A synchronization cycle with the server has been
    completed.
    * offline_update: An attempt was made to update or delete a record
    while offline.
    * collision_detected: Update failed due to data collision.
    * remote_update_failed: Update failed for a reason other than data
    collision.
    * remote_update_applied: An update was applied to the remote data
    store.
    * local_update_applied: An update was applied to the local data
    store.
    * delta_received: A change was received from the remote data store
    for the dataset. It is best to listen to this notification and update the
    UI accordingly.
    * record_delta_received: A delta was received from the remote data
    store for the record. It is best to listen to this notification and
    update UI accordingly.
    * sync_failed: Synchronization loop failed to complete.
    */
});
```

Android (Java)

同期イベントは、**syncClient.init** を使用して登録した **FHSyncListener** インスタンスに送信されます。リスナーの各メソッドには、null でない **NotificationMessage** パラメーターが提供されます。

```

public class SampleSyncListener implements FHSyncListener {

    public void onSyncStarted(NotificationMessage notificationMessage) {
        /*Data sync is available. Update your UI, enable editing fields,
        display messages to the user, etc.*/
    }

    public void onSyncCompleted(NotificationMessage notificationMessage) {
        /*Sync has completed. Data has been successfully sent to the server
or
successfully received from the server. In either case you should
refresh
the data presented to the user.

You may retrieve your latest data for this message with
FHSyncClient.getInstance().list(notificationMessage.getDataId())*/
    }

    public void onUpdateOffline(NotificationMessage notificationMessage) {
        /*A create, delete, or update operation was called, but the device
is
not connected to the network. The UI should be updated, fields
disabled,
user notified, etc.*/
    }

    public void onCollisionDetected(NotificationMessage
notificationMessage) {
        /* The update could not be applied to the server. There are many
reasons
why this could happen and it is up to the application developer to
resolve the collision.

After the data has been updated to synchronize cleanly, the methods
FHSyncClient.listCollisions and FHSyncClient.removeCollision can be
used
to view and resolve the collision entries.

Use
FHSyncClient.getInstance().read(notificationMessage.getDataId(),
notificationMessage.getUID())
to view the data record.

*/
    }

    public void onRemoteUpdateFailed(NotificationMessage
notificationMessage) {
        /* The remote updated failed. You may use
notificationMessage.getExtraMessage()
to get additional details.

Use
FHSyncClient.getInstance().read(notificationMessage.getDataId(),

```

```

notificationMessage.getUID())

    to view the data record.*/
}

public void onRemoteUpdateApplied(NotificationMessage
notificationMessage) {
    /* An update was successfully processed by the remote server.

    Use
    FHSyncClient.getInstance().read(notificationMessage.getDataId(),
                                    notificationMessage.getUID())

    to view the data record.
    */
}

public void onLocalUpdateApplied(NotificationMessage
notificationMessage) {
    /* An update is applied locally and waiting to be sent to the
    remote
    server.

    Use
    FHSyncClient.getInstance().read(notificationMessage.getDataId(),
                                    notificationMessage.getUID())

    to view the data record.
    */
}

public void onDeltaReceived(NotificationMessage notificationMessage) {
    /*An incoming update has been applied. The UI should be updated if
    appropriate.

    Use
    FHSyncClient.getInstance().read(notificationMessage.getDataId(),
                                    notificationMessage.getUID())

    to view the data record.

    Use
    FHSyncClient.getInstance().list(notificationMessage.getDataId())
    to load all data records.

    notificationMessage.getExtraMessage() will return the type of
    operation
    (update, delete, create) which was performed.

    */
}

public void onSyncFailed(NotificationMessage notificationMessage) {
    /*
    For some reason the sync loop was unable to complete. This could
    be for
    many different reasons such as network connectivity, authentication
    issues, programming errors, etc.

    Use notificationMessage.getExtraMessage() to get extra information.

```



```

        */
    }

    public void onClientStorageFailed(NotificationMessage
notificationMessage) {
        /*
         Sync was not able to store data locally. This indicates a device
error
         such as out of space, invalid permissions, etc

         Use notificationMessage.getExtraMessage() to get extra information.
        */
    }
}

```

iOS (Objective-C)

同期通知は、標準の **NSNotificationCenter** 機能で配布されます。**kFHSyncStateChangedNotification** 通知の受信を開始するには、**NSNotificationCenter** の **addObserver:selector:name:object:** または **addObserverForName:object:queue:usingBlock:** メソッドを使用して登録します。

```

[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onSyncMessage:) name:kFHSyncStateChangedNotification
object:nil];

* (void) onSyncMessage:(NSNotification*) note
{
    FHSyncNotificationMessage* msg = (FHSyncNotificationMessage*) [note
object];
    NSString* code = msg.code;
    if([code isEqualToString:REMOTE_UPDATE_APPLIED_MESSAGE]) {
    }

    /* Codes:
    *
    * NSString *const SYNC_STARTED_MESSAGE = @"SYNC_STARTED";
    * A synchronization cycle with the server has been started.
    *
    * NSString *const SYNC_COMPLETE_MESSAGE = @"SYNC_COMPLETE";
    * A synchronization cycle with the server has been completed.
    *
    * NSString *const SYNC_FAILED_MESSAGE = @"SYNC_FAILED";
    * Synchronization loop failed to complete.
    *
    * NSString *const OFFLINE_UPDATE_MESSAGE = @"OFFLINE_UPDATE";
    * An attempt was made to update or delete a record while offline.
    *
    * NSString *const COLLISION_DETECTED_MESSAGE = @"COLLISION_DETECTED";
    * Update failed due to data collision.
    *
    * NSString *const REMOTE_UPDATE_FAILED_MESSAGE =
@"REMOTE_UPDATE_FAILED";
    * Update failed for a reason other than data collision.
    *

```

```

    * NSString *const REMOTE_UPDATE_APPLIED_MESSAGE =
    @"REMOTE_UPDATE_APPLIED";
    * An update was applied to the remote data store.
    *
    * NSString *const LOCAL_UPDATE_APPLIED_MESSAGE =
    @"LOCAL_UPDATE_APPLIED";
    * An update was applied to the local data store.
    *
    * NSString *const DELTA_RECEIVED_MESSAGE = @"DELTA_RECEIVED";
    * An change was received from the remote data store for the dataset.
    * It's best to listen to this notification and update UI accordingly.
    *
    * NSString *const CLIENT_STORAGE_FAILED_MESSAGE =
    @"CLIENT_STORAGE_FAILED";
    * Loading or saving to client storage failed. This is a critical error
    and the Sync Client will not work properly without client storage.
    */
}

```

iOS (Swift)

同期通知は、標準の **NSNotificationCenter** 機能で配布されます。**kFHSyncStateChangedNotification** 通知の受信を開始するには、**NSNotificationCenter** の **addObserver(_:selector:name:object:)** または **addObserverForName(_:object:queue:usingBlock:)** メソッドを使用して登録します。

```

NSNotificationCenter.defaultCenter().addObserver(self,
selector:Selector("onSyncMessage:"),
name:"kFHSyncStateChangedNotification", object:nil)

public func onSyncMessage(note: NSNotification) {
    if let msg = note.object as? FHSyncNotificationMessage, let code =
msg.code {
        if code == REMOTE_UPDATE_APPLIED_MESSAGE {
        }
        /* Codes:
        *
        * let SYNC_STARTED_MESSAGE = "SYNC_STARTED"
        * A synchronization cycle with the server has been started.
        *
        * let SYNC_COMPLETE_MESSAGE = "SYNC_COMPLETE"
        * A synchronization cycle with the server has been completed.
        *
        * let SYNC_FAILED_MESSAGE = "SYNC_FAILED"
        * Synchronization loop failed to complete.
        *
        * let OFFLINE_UPDATE_MESSAGE = "OFFLINE_UPDATE"
        * An attempt was made to update or delete a record while offline.
        *
        * let COLLISION_DETECTED_MESSAGE = "COLLISION_DETECTED"
        * Update failed due to data collision.
        *
        * let REMOTE_UPDATE_FAILED_MESSAGE = "REMOTE_UPDATE_FAILED"
        * Update failed for a reason other than data collision.
        *

```

```

* let REMOTE_UPDATE_APPLIED_MESSAGE = "REMOTE_UPDATE_APPLIED"
* An update was applied to the remote data store.
*
* let LOCAL_UPDATE_APPLIED_MESSAGE = "LOCAL_UPDATE_APPLIED"
* An update was applied to the local data store.
*
* let DELTA_RECEIVED_MESSAGE = "DELTA_RECEIVED"
* An change was received from the remote data store for the dataset.
* It's best to listen to this notification and update UI accordingly.
*
* let CLIENT_STORAGE_FAILED_MESSAGE = "CLIENT_STORAGE_FAILED"
* Loading or saving to client storage failed. This is a critical
error and the Sync Client will not work properly without client storage.
*/
}

```

.NET (C#)

以下のセクションでは、**client** は設定済みかつ初期化済みの **FHSyncClient** インスタンスです。タイプが **EventHandler<FHSyncNotificationEventArgs>** のイベントハンドラーをクライアントがサポートする別のイベントタイプに設定することができます。

```

/// The event arguments that will be sent to the sync event listeners
public class FHSyncNotificationEventArgs : EventArgs
{
    /// The id of the dataset
    public string DatasetId { set; get; }

    /// The unique universal id of the record
    public string Uid { private get; set; }

    /// Type fo the notification.
    public SyncNotification Code { get; set; }

    /// An message associated with the event argument. Could be
empty.
    public string Message { get; set; }
}

/// Loading or saving to client storage failed. This is a critical error
and the Sync Client will not work properly without client storage.
client.ClientStorageFailed += async (sender, args) => { };

/// A synchronization cycle with the server has been started.
client.SyncStarted += async (sender, args) => { };

/// A synchronization cycle with the server has been completed.
client.SyncCompleted += async (sender, args) => { };

/// An attempt was made to update or delete a record while offline.
client.OfflineUpdate += async (sender, args) => { };

/// Update failed due to data collision.
client.CollisionDetected += async (sender, args) => { };

```

```

/// Update failed for a reason other than data collision.
client.RemoteUpdateFailed += async (sender, args) => { };

/// An update was applied to the local data store.
client.LocalUpdateApplied += async (sender, args) => { };

/// An update was applied to the remote data store.
client.RemoteUpdateApplied += async (sender, args) => { };

/// A change was received from the remote data store for the dataset. It's
best to listen to this notification and update UI accordingly.
client.DeltaReceived += async (sender, args) => { };

/// A delta was received from the remote data store for the record. It's
best to listen to this notification and update UI accordingly.
client.RecordDeltaReceived += async (sender, args) => { };

/// Synchronization loop failed to complete.
client.SyncFailed += async (sender, args) => { };

```

11.2.3. 同期通知

このセクションでは、JavaScript SDK 向けの通知構造について説明します。Objective-C、Swift、Android および .NET のすべての SDK は専用のオブジェクトで通知構造を定義しますが、JavaScript 情報もこれらすべての SDK に便利なものです。

JavaScript SDK 内の通知は以下の構造になります。

```

{
  "dataset_id": String,
  "uid": [String],
  "code": String,
  "message": Object|String
}

```

ここでは、

- ※ **dataset_id** は通知に関連するデータセットの名前になります。
- ※ **uid** は関連レコードの UID です。これは永続的でないレコード (レコードハッシュ) の UID、または永続的レコードの永続的 UID のどちらでも構いません。**sync_started**、**sync_failed** および **local_update_applied** など特定のレコードを通知が参照しない場合は、**uid** の値を **null** とすることができます。また、**sync_complete** などの通知の場合は、**uid** エントリはありません。詳細については、個別の通知構造を参照してください。
- ※ **code** は **sync_started** など、通知のタイプの識別子になります。
- ※ **message** は通知とともに送信される追加データで、各通知ごとに異なります。詳細については、個別の通知を参照してください。

11.2.3.1. Sync Started

同期ループが開始されます。この通知はクライアントがオンラインかオフラインかに関係なく、同期ループごとに送信されます。

11.2.3.1.1. 通知コード

- ※ Javascript - **sync_started**
- ※ Objective-C/Swift - **SYNC_STARTED**

11.2.3.1.2. 通知の構造

```
{
  "dataset_id": "myDataset",
  "uid": null, 1
  "code": "sync_started",
  "message": null 2
}
```

1

uid は常に **null** です。

2

message は常に **null** です。

11.2.3.2. Sync Complete

同期ループが完了します。同期ループには、バックエンドへの初期同期要求やその他の要求、応答の結果として実行されるアクションが含まれます。この通知は **sync_failed** イベントと同じ同期ループでは発生しません。つまり、同期ループは完了するか失敗するかのいずれかになります。

11.2.3.2.1. 通知コード

- ※ JavaScript - **sync_complete**
- ※ Objective-C/Swift - **SYNC_COMPLETE**

11.2.3.2.2. 通知の構造

```
{
  "dataset_id": "myDataset",
  "code": "sync_complete",
  "message": "online" // message is always "online"
}
```



注記

uid がこの構造に表示されることはありません。

11.2.3.3. Sync Failed

同期ループが完全に失敗しました。これは以下の 2 つの場合に発生します。同期ループ中に sync クライアントがオフラインだった。または、sync レコード要求が 500 などエラーコードを応答で受け取った。Sync failed 通知が送信されるのは、**notify_sync_failed** オプションが **true** に設定されている場合のみです。

11.2.3.3.1. 通知コード

- ※ JavaScript - **sync_failed**
- ※ Objective-C/Swift - **SYNC_FAILED**

11.2.3.3.2. 通知の構造

```
{
  "dataset_id": "myDataset",
  "uid": "a3387ce5d175cf73ec5f5d7614c7c6b4f44393f2_1",
  "code": "sync_failed",
  "message": "offline" 1
}
```

1

クライアントがオフラインの場合は、**message** は "offline" になります。その他の場合は、メッセージには同期サーバーから失敗した回答メッセージが含まれます。

11.2.3.4. Record Delta Received



注記

この通知は JavaScript SDK にのみあります。Delta Received は、他のすべての SDK で同一タスクを実行します。

変更がレコードのリモートデータストアで受信されます。この通知は、同期応答で返される各変更 (**update**, **delete**, **create**) で発生します。

11.2.3.4.1. 通知コード

- ※ JavaScript - **record_delta_received**

11.2.3.4.2. 通知の構造

```
{
  "dataset_id": "myDataset",
  "uid": "58e3b2cff7fc7297eb635053",
  "code": "record_delta_received",
  "message": "update" 1
}
```

1

message は "create"、"update" または "delete" のいずれかになります。

11.2.3.5. Delta Received

変更がデータセットのリモートデータストアで受信されます。JavaScript SDK では、各同期レコードコールごとに発生します。複数の変更があってイベントオブジェクトがグローバルハッシュを提供する場合でも 1 回しか発生しないので、**record_delta_received** イベントとは異なります。更新をバックエンドから受信したかどうか確認するには、この通知を使用します。ただし、変更についての知識は必要ありません。

Objective-C、Swift および Android の SDK では、この通知は **record_delta_received** 通知として機能します。つまり、同期サーバーによって返される各更新ごとに 1 回発生します。

11.2.3.5.1. 通知コード

JavaScript - **delta_received**

Objective-C/Swift - **DELTA_RECEIVED**

11.2.3.5.2. 通知の構造

```
{
  "dataset_id": "myDataset",
  "uid": "a3387ce5d175cf73ec5f5d7614c7c6b4f44393f2_1",
  "code": "delta_received",
  "message": "partial dataset" 1
}
```

1

message は常に "partial dataset" です。

11.2.3.6. Local Update Applied

変更がローカルデータセットに適用されます。これは以下の状況で発生します。* データセットがローカルのデータストレージから読み込まれる。* データセットがローカルで新規の保留中レコードで更新される (例えば、ユーザーが新規レコードを作成する場合など)。別のクライアントが元になっているサーバーから変更を受け取った場合には、この通知は発生しません。このような変更のタイプを確認する場合は、**record_delta_received** イベントを使用します。

11.2.3.6.1. 通知コード

JavaScript - **local_update_applied**

Objective-C/Swift - **LOCAL_UPDATE_APPLIED**

11.2.3.6.2. 通知の構造

```
{
  "dataset_id": "myDataset",
  "uid": "8783f802c8d43053ee2bf02009dd79d89b186afb",
  "code": "local_update_applied",
  "message": "update" 1
}
```

1

message は "load"、"create"、"update" または "delete" のいずれかになります。

11.2.3.7. Remote Update Applied

更新がリモートデータストアに適用されます。この通知は、バックエンドで正常に適用された各レコードごとに発生します。

11.2.3.7.1. 通知コード

- ✎ JavaScript - **remote_update_applied**
- ✎ Objective-C/Swift - **REMOTE_UPDATE_APPLIED**

11.2.3.7.2. 通知の構造

```
{
  "dataset_id": "myDataset",
  "uid": "58e3b2e6f7fc7297eb635100",
  "code": "remote_update_applied",
  "message": { 1
    "_id": "58e3b2e60d6412349926e95b",
    "action": "create",
    "cuid": "5733885DDC134A6FA9754BA67E4E4BAC",
    "hash": "66c3961bc1dc7139427a95d3471a27ae30d4cb96",
    "msg": null,
    "oldUid": "66c3961bc1dc7139427a95d3471a27ae30d4cb96",
    "timestamp": 1491317478112,
    "type": "applied",
    "uid": "58e3b2e6f7fc7297eb635100"
  }
}
```

1

message は、同期サーバーから受信した **update** オブジェクトです。

11.2.3.8. Remote Update Failed

更新のリモートデータストアへの適用に失敗しました。この通知は、適用に失敗した各レコードごとに発生します。更新失敗の一般的な理由は、データがリモートデータストアに維持されないためです。

11.2.3.8.1. 通知コード

- ✧ JavaScript - **remote_update_failed**
- ✧ Objective-C/Swift - **REMOTE_UPDATE_FAILED**

11.2.3.8.2. 通知の構造

remote_update_applied と似ていますが、**code** が変更されています。

11.2.3.9. Collision Detected

同期サーバーで競合が見つかりました。この通知は、同期サーバーが検出した競合ごとに発生します。競合は、古くなったクライアントが状態の変わったレコードを更新しようとするると発生します。競合の処理方法は、サーバー側の競合ハンドラーが決定します。

11.2.3.9.1. 通知コード

- ✧ JavaScript - **collision_detected**
- ✧ Objective-C/Swift - **COLLISION_DETECTED**

11.2.3.9.2. 通知の構造

remote_update_applied と似ていますが、**code** が変更されています。

11.2.3.10. Offline Update

クライアントがオフラインの間にデータセットからレコードが追加、更新または削除されます。

11.2.3.10.1. 通知コード

- ✧ JavaScript - **offline_update**
- ✧ Objective-C/Swift - **OFFLINE_UPDATE**

11.2.3.10.2. 通知の構造

```
{
  "dataset_id": "myDataset",
  "uid": "58e3b2cff7fc7297eb635053",
  "code": "offline_update",
  "message": "update" 1
}
```

message は "create"、"update" または "delete" のいずれかになります。

11.2.3.11. Client Storage Failed

同期の基礎的なデータストレージ (Lawnchair) が問題に遭遇しました。これは、クエリが完了に失敗した場合に発生します。

11.2.3.11.1. 通知コード

- ✳ JavaScript - **client_storage_failed**
- ✳ Objective-C/Swift - **CLIENT_STORAGE_FAILED**

11.2.3.11.2. 通知の構造

```
{
  "dataset_id": "myDataset",
  "uid": null, 1
  "code": "client_storage_failed",
  "message": "load from local storage failed"
}
```

1

uid は常に **null** です。

11.3. \$FH.SYNC.MANAGE

```
$fh.sync.manage(dataset_id, options, query_params, meta_data, callback);
```

11.3.1. 詳細

データセットを sync service の管理下に置きます。同一データセットで **manage** を複数回呼び出すとオプションと **query_params** を更新しますが、データセットは複数回同期されません。

11.3.2. 例

JavaScript

```
var dataset_id = 'tasks';

// Configuration options object.
// These override the options passed to init.
var options = {
  "sync_frequency": 30 // Sync every 30 seconds for the 'tasks' dataset
};

// Parameters object to be passed to the cloud sync service.
// It will be passed to the dataHandler when listing dataset on the back
```

```

end.
// If the default MBaaS cloud implementation is used (which uses $fh.db
// for data handlers), all the valid list options can be used here.
// For example, to list the tasks that are assigned to a user called
// "Tom", the query params should be
var query_params = {
  "eq": {
    "assigned": "Tom"
  }
};

// Extra params that will be sent to the back-end data handlers.
var meta_data = {};
$fh.sync.manage(dataset_id, options, query_params, meta_data, function(){
  console.log('dataset ' + dataset_id + ' is now managed by sync');
});

```

Android (Java)

```

//queryParams are any query supported by $fh.db
JSONObject queryParams = new JSONObject();

//MetaData such as sessionTokens, userIds, etc
JSONObject metaData = new JSONObject();

//Any String identifier
String dataSet = "myDataSetId";

// If configOverride is null then the config provided in
FHSyncClient.init
// will be used instead.
FHSyncConfig configOverride = null;

FHSyncClient.getInstance().manage(dataSet, configOverride, queryParams,
metaData);

```

iOS (Objective-C)

```

// Unique Id for the dataset to manage.
#define DATA_ID @"tasks"

// Configuration options object.
// These override the options passed to init.
FHSyncConfig* conf = [[FHSyncConfig alloc] init];
conf.syncFrequency = 10;

// Parameters object to be passed to the cloud sync service.
// For example, to list the tasks that are assigned to a user called
// "Tom":
NSDictionary* query = @{@"assigned": @"Tom"};

// Extra params that will be sent to the back-end data handlers.
NSMutableDictionary* metaData = nil;

```

```
// Initialise Sync Client
FHSyncClient* syncClient = [[FHSyncClient alloc] initWithConfig:conf];

// Put a dataset under the management of the sync service.
[syncClient manageWithDataId:DATA_ID AndConfig:conf AndQuery:query
AndMetaData:metaData];
```

iOS (Swift)

```
public let DATA_ID = "tasks"

// Configuration options object.
// These override the options passed to init.
let conf = FHSyncConfig()
conf.syncFrequency = 10

// Parameters object to be passed to the cloud sync service.
// For example, to list the tasks that are assigned to a user called
// "Tom":
let query = ["assigned": "Tom"]

// Initialise Sync Client
let syncClient = FHSyncClient(config: conf)

// Put a dataset under the management of the sync service.
syncClient.manageWithDataId(DATA_ID, andConfig:conf, andQuery:query)
```

.NET (C#)

以下のセクションでは、**client** は設定済みかつ初期化済みの FHSyncClient インスタンスです。Windows プラットフォーム上で FHSyncClient が管理するデータセットは、インターフェース **IFHSyncModel** を実装する必要があります。

```
/// The datasetId needs to be unique for your app and will be used to
name the
/// collection in the cloud.
const string DatasetId = "tasks";

/// Query is a Dictionary of parameters to be sent to the server with
each sync
/// operation. If the default MBaaS cloud implementation is used (which
uses
/// $fh.db for data handlers), all the valid list options can be used
here.
/// For example, to list the tasks that are assigned to a user called
// "Tom",
/// the query params should be
Dictionary<string, string> query = new Dictionary<string, string>
{
    {"eq", "{\"assigned\", \"Tom\"}"}
};

/// When you manage a DataSet you may set new configuration parameters to
/// override the parameters for the sync client. If you do not wish to do
```

```

this,
/// you may pass null into the FHSyncClient.manage method.
var config = new FHSyncConfig();
config.SyncFrequency = 100;

/// Put a dataset under the management of the sync service. Note that
Task
/// is an implementation of the IFHSyncModel.
client.Manage<Task>(DatasetId, config, query);

```

11.4. \$FH.SYNC.DOLIST

```
$fh.sync.doList(dataset_id, success, failure);
```

11.4.1. 詳細

データセットのレコード一覧を取得します。

11.4.2. 例

JavaScript

```

// Unique Id for the dataset to manage.
// This must correspond to an "act" function which represents the cloud
portion of the sync contract.
var dataset_id = 'tasks';

$fh.sync.doList(dataset_id, function(res) {
// The data returned by the sync service.
// Always a full data set (even in the case of deltas).
console.log(res);

//res is a JSON object
for(var key in res){
    if(res.hasOwnProperty(key)){
        // Unique Id of the record, used for read, update & delete operations
        (string).
        var uid = key;
        // Record data, opaque to sync service.
        var data = res[key].data;
        // Unique hash value for this record
        var hash = res[key].hash;
    }
}

}, function(code, msg) {
// Error code. Currently only 'unknown_dataset' is possible
console.error(code);

```

```
// Optional free text message with additional information
console.error(msg);

});
```

Android (Java)

```
FHClient fhClient = FHSyncClient.getInstance();

// Unique Id for the dataset being manage.
String dataSetId = "photos";

// The data returned by the sync service.
// Always a full data set (even in the case of deltas).
JSONObject allData = fhClient.getSyncClient().list("photos");

Iterator<String> keysIterator = allData.keys();
List<Project> itemsToSync = new ArrayList<>();

while (keysIterator.hasNext()) {
    // Unique Id of the record, used for read,
    // update & delete operations (string).
    String uid = keysIterator.next();

    // Record data
    JSONObject record = allData.getJSONObject(uid);

    // The synced data object. In Android this can be a JSON serialized
    POJO
    JSONObject dataObj = record.getJSONObject("data");

    // Unique hash value for this record
    String hash = record.getString("hash");
}

projects.addAll(itemsToSync);
bus.post(new ProjectsAvailable(new ArrayList<Project>(projects)));
```

iOS (Objective-C)

```
// Unique Id for the dataset to manage.
#define DATA_ID @"tasks"

// The data returned by the sync service.
// Always a full data set (even in the case of deltas).
NSDictionary* items = [syncClient listWithDataId:DATA_ID];
[items enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    // Unique Id of the record, used for read,
    // update & delete operations (string).
    NSString* uid = key; +
    // Record data
```

```

NSMutableDictionary* object = obj;
NSMutableDictionary* dataObj = object[@"data"];
uid = object[@"uid"];
}];

```

iOS (Swift)

```

// Unique Id for the dataset to manage.
public let DATA_ID = "tasks"
// The data returned by the sync service.
// Always a full data set (even in the case of deltas).
let items = syncClient.listWithDataId(DATA_ID)
for (key, value) in items {
if let data = value["data"], let uid = value["uid"] {
// do something with item
}
}

```

.NET (C#)

```

/// The datasetId needs to be unique for your app and will be used to
name the
/// collection in the cloud.
const string DatasetId = "tasks";

foreach (var item in client.List<Task>(DatasetId))
{
/// Do Something with item
}

```

11.5. \$FH.SYNC.DOCREATE

```

$fh.sync.doCreate(dataset_id, data, success, failure);

```

11.5.1. 詳細

一意の ID に関連付けられているデータを更新します。

11.5.2. 例

JavaScript

```

var dataset_id = 'tasks';

// Record data to create, opaque to sync service.
var data = {
  "name": "Organise widgets",
  "time": Date.now() + 100000,
  "user": "joe@bloggs.com"
};

```

```

$fh.sync.doCreate(dataset_id, data, function(res) {
  // The update record which will be sent to the cloud
  console.log(res);
}, function(code, msg) {
  // Error code. One of 'unknown_dataset' or 'unknown_id'
  console.error(code);

  // Optional free text message with additional information
  console.error(msg);

});

```

Android (Java)

```

String dataSetId = "tasks";

// Record data to create
JSONObject data = new JSONObject();
data.put("name", "Organise widgets");
data.put("time", new Date().getTime() + 100000);
data.put("user", "joe@bloggs.com");

syncClient.create(dataSetId, data);

```

iOS (Objective-C)

```

// Unique Id for the dataset to manage.

#define DATA_ID @"tasks"

NSDate* now = [NSDate date];
NSMutableDictionary* data = [NSMutableDictionary dictionary];
[data setObject:shoppingItem.name forKey:@"name"];
[data setObject:[NSNumber numberWithInt:[now
timeIntervalSince1970]*1000] forKey:@"created"];
[syncClient createWithDataId:DATA_ID AndData:data];

```

iOS (Swift)

```

// Unique Id for the dataset to manage.
public let DATA_ID = "tasks"

let myItem: [String: AnyObject] = ["name": name, "created": created*1000]

syncClient.createWithDataId(DATA_ID, andData: myItem)

```

.NET (C#)

以下のセクションでは、**client** は設定済みかつ初期化済みの FHSyncClient インスタンスです。Task は **IFHSyncModel** を実装するクラスで、**string Name** プロパティを持っています。

■


```

/// The datasetId needs to be unique for your app and will be used to
name the
/// collection in the cloud.
const string DatasetId = "tasks";

Task task = new Task();
task.Name = "task name";

client.Create(MainPage.DatasetId, task);

```

11.6. \$FH.SYNC.DOREAD

```
$fh.sync.doRead(dataset_id, uid, success, failure);
```

11.6.1. 詳細

単一のデータレコードを読み取ります。

11.6.2. 例

JavaScript

```

var dataset_id = 'tasks';

// Unique Id of the record to read.
var uid = '42abcdefg';

$fh.sync.doRead(dataset_id, uid, function(data) {
// The record data
console.log(data.data); //the data files
console.log(data.hash); //the hash value of the data
}, function(code, msg) {
// Error code. One of 'unknown_dataset' or 'unknown_id'
console.error(code);

// Optional free text message with additional information
console.error(msg);
});

```

Android (Java)

```

//name of dataset to manage
String dataSetId = "tasks";

// Unique Id of the record to read.
String uid = "42abcdefg";

JSONObject record = FHSyncClient.getInstance().read(dataSetId, uid);

```

```
if (data != null) {
  JSONObject document = record.getJSONObject("data");
  String uid = record.getString("uid");
}
```

iOS (Objective-C)

```
// Unique Id for the dataset to manage.

#define DATA_ID @"tasks"

// The data returned by the sync service.
// Always a full data set (even in the case of deltas).
NSDictionary* item = [syncClient readWithDataId:DATA_ID
AndUID:@"42abcdefg"];
```

iOS (Swift)

```
// Unique Id for the dataset to manage.
public let DATA_ID = "tasks"

// The data returned by the sync service.
// Always a full data set (even in the case of deltas).
let item = syncClient.readWithDataId(DATA_ID, andUID: "42abcdefg")
```

.NET (C#)

```
string datasetId = "tasks";

/// Unique Id of the record to read.
string uid = "42abcdefg";

Task task = client.Read(datasetId, uid);
```

11.7. \$FH.SYNC.DOUPDATE

```
$fh.sync.doUpdate(dataset_id, uid, data, success, failure);
```

11.7.1. 詳細

一意の ID に関連付けられているデータを更新します。

11.7.2. 例

JavaScript

```
var dataset_id = 'tasks';
```

```

// Unique Id of the record to update.
var uid = '42abcdefg';

// Record data to update. Note that you need to provide the FULL data to
// update.
$fh.sync.doRead(dataset_id, uid, function(data){
var fields = data.data;
fields.name = "Organise layouts";
$fh.sync.doUpdate(dataset_id, uid, fields, function(data) {
// The updated record which will be send to the cloud
console.log(data);
}, function(code, msg) {
// Error code. One of 'unknown_dataset' or 'unknown_id'
console.error(code);

// Optional free text message with additional information
console.error(msg);
});
});

```

Android (Java)

```

// name of dataset to manage
String dataSetId = "tasks";

// Unique Id of the record to read and update.
String uid = "42abcdefg";

// Fetch a record
JSONObject record = FHSyncClient.getInstance().read(dataSetId, uid);

// Fetch the data of the record and change a field
JSONObject data = record.getJSONObject("data");
data.set("newField", "newValue");

// Update the data in the sync system
FHSyncClient.getInstance().update(dataSetId, uid, data);

```

iOS (Objective-C)

```

// Unique Id for the dataset to manage.
#define DATA_ID @"tasks"

// The Updated data
NSDate* now = [NSDate date];
NSMutableDictionary* data = [NSMutableDictionary dictionary];
[data setObject:shoppingItem.name forKey:@"name"];
[data setObject:[NSNumber numberWithInt:[now
timeIntervalSince1970]*1000] forKey:@"created"];

NSDictionary* item = [syncClient updateWithDataId:DATA_ID
AndUID:@"42abcdefg" AndData:data];

```

iOS (Swift)

```
// Unique Id for the dataset to manage.
public let DATA_ID = "tasks"

// The Updated data
let myItem: [String: AnyObject] = ["name": name, "created": created*1000]
syncClient.updateWithDataId(DATA_ID, andUID: uid, andData: myItem)
```

.NET (C#)

```
string datasetId = "tasks";

/// Unique Id of the record to read.
string uid = "42abcdefg";

Task task = client.Read(datasetId, uid);

task.Name = "new name";

Task task = client.Update(datasetId, task);
```

11.8. \$FH.SYNC.DODELETE

```
$fh.sync.doDelete(dataset_id, uid, success, failure);
```

11.8.1. 詳細

一意の ID に関連付けられているデータを削除します。

11.8.2. 例

JavaScript

```
var dataset_id = 'tasks';

// Unique Id of the record to delete.
var uid = '42abcdefg';

$fh.sync.doDelete(dataset_id, uid, function(data) {
// The deleted record data sent to the cloud.
console.log(data);
}, function(code, msg) {
// Error code. One of 'unknown_dataset' or 'unknown_id'
console.error(code);

// Optional free text message with additional information
console.error(msg);
})
```

Android (Java)

```
// name of dataset to manage
String dataSetId = "tasks";

// Unique Id of the record to remove.
String uid = "42abcdefg";

FHSyncClient.getInstance().delete(dataSetId, uid);
```

iOS (Objective-C)

```
// Unique Id for the dataset to manage.

#define DATA_ID @"tasks"

NSDictionary* item = [syncClient deleteWithDataId:DATA_ID
AndUID:@"42abcdefg"];
```

```
<div class="tab-pane" id="example-doDelete-swift">
```

```
// Unique Id for the dataset to manage.
public let DATA_ID = "tasks"

syncClient.deleteWithDataId(DATA_ID, andUID: uid)
```

.NET (C#)

```
string datasetId = "tasks";

/// Unique Id of the record to delete.
string uid = "42abcdefg";

client.Delete(datasetId, uid);
```

11.9. \$FH.SYNC.STARTSYNC

```
$fh.sync.startSync(dataset_id, success, failure)
```

11.9.1. 詳細

'sync_active' オプションが false に設定されている場合、sync loop を開始します。

11.9.2. 例

JavaScript

```
var dataset_id = 'tasks';
```

```
$fh.sync.startSync(dataset_id, function(){
  console.log('sync loop started');
}, function(error){
  console.log('failed to start sync loop. Error : ' + error);
});
```

Android (Java)

FHSyncListener が Activity または Fragment を参照する場合は、[アクティビティのライフサイクル管理](#) を検討する必要があります。この状況では、**pauseSync** および **resumeSync** のメソッドが作成されます。また、同期を完全にシャットダウンする **destroy** メソッドもあります。

```
// Synchronization is automatically started by the FHSyncClient.init
// method.
// However, synchronization may be paused and resumed in the Activity
// lifecycle onPause and onResume methods.

@Override
public void onPause() {
  super.onPause();
  FHSyncClient.getInstance().pauseSync();
}

@Override
public void onResume() {
  super.onResume();
  FHSyncClient.getInstance().resumeSync(new FHSyncListener() { });
}

public void onDestroy() {
  super.onDestroy();
  FHSyncClient.getInstance().destroy();
}
```

iOS (Objective-C)

iOS Synchronization API には **startSync** メソッドがありません。同期は [init](#) メソッドで開始されます。

iOS (Swift)

iOS Synchronization API には **startSync** メソッドがありません。同期は [init](#) メソッドで開始されます。

.NET (C#)

```
string datasetId = "tasks";

client.Start(datasetId);
```

11.10. \$FH.SYNC.STOPSYNC

```
$fh.sync.stopSync(dataset_id, success, failure)
```

11.10.1. 詳細

データセットの sync loop を停止します。

11.10.2. 例

JavaScript

```
var dataset_id = 'tasks';

$fh.sync.stopSync(dataset_id, function(){
  console.log('sync loop stopped');
}, function(error){
  console.log('failed to stop sync loop. Error : ' + error);
});
```

Android (Java)

stop 関数はデータセットの同期を停止しますが、**FHSyncClient** インスタンスにアタッチされている **FHSyncListener** は削除されません。

```
String dataSetId = "tasks";

FHSyncClient.getInstance().stop(dataSetId);
```

iOS (Objective-C)

```
// Unique Id for the dataset to manage.

#define DATA_ID @"tasks"

[syncClient stopWithDataId:DATA_ID];
```

iOS (Swift)

```
// Unique Id for the dataset to manage.
public let DATA_ID = "tasks"

syncClient.stopWithDataId(DATA_ID)
```

.NET (C#)

```
string datasetId = "tasks";

client.Stop(datasetId);
```

11.11. \$FH.SYNC.DOSYNC

```
$fh.sync.doSync(dataset_id, success, failure)
```

11.11.1. 詳細

sync_active が true の場合、sync loop をほぼ即座に (500 ミリ秒以内) 実行します。

11.11.2. 例

JavaScript

```
var dataset_id = 'tasks';

$fh.sync.doSync(dataset_id, function(){
  console.log('sync loop will run');
}, function(error){
  console.log('failed to run sync loop. Error : ' + error);
});
```

Android (Java)

Android SDK には **doSync** メソッドはありません。代わりに [forceSync](#) を使用してください。

iOS (Objective-C)

iOS Synchronization API には **doSync** メソッドがありません。代わりに [forceSync](#) を使用してください。

iOS (Swift)

iOS Synchronization API には **doSync** メソッドがありません。代わりに [forceSync](#) を使用してください。

NET (C#)

Windows Synchronization API には **doSync** メソッドがありません。代わりに [forceSync](#) を使用してください。

11.12. \$FH.SYNC.FORCESYNC

```
$fh.sync.forceSync(dataset_id, success, failure)
```

11.12.1. 詳細

sync_active が false の場合でも、sync loop をほぼ即座に (500 ミリ秒以内) 実行します。

11.12.2. 例

JavaScript

```
var dataset_id = 'tasks';

$fh.sync.forceSync(dataset_id, function(){
  console.log('sync loop will run');
}, function(error){
  console.log('failed to run sync loop. Error : ' + error);
});
```

Android (Java)

FHSyncClient が **FHSyncClient.destroy()** で破棄されている場合は、**forceSync** を呼び出す前に **init** を再度呼び出す必要があります。同期が一時停止されると同期ループは実行されたままになりますが、リスナーはアタッチされず、イベントは起動されません。

```
String dataSetId = "tasks";

FHSyncClient.getInstance().forceSync(dataSetId);
```

iOS (Objective-C)

```
// Unique Id for the dataset to manage.

#define DATA_ID @"tasks"

[syncClient forceSync:DATA_ID];
```

iOS (Swift)

```
// Unique Id for the dataset to manage.
public let DATA_ID = "tasks"

syncClient.forceSync(DATA_ID)
```

.NET (C#)

```
string datasetId = "tasks";

client.ForceSync(datasetId);
```

第12章 \$FH.ACT

注意

この API は非推奨となっています。代わりに [\\$fh.cloud](#) を使用してください。

```
$fh.act(options, success, failure);
```

JSON オブジェクトを入力として受信し、JSON オブジェクトを出力として返すクラウド側の JavaScript 関数を呼び出します。

サポートされるプラットフォーム

JavaScript SDK

- Cordova
- Web Apps

Android SDK

iOS Objective-C SDK

iOS Swift SDK

.NET SDK

- Windows
- Xamarin

詳細なバージョン情報については、[Supported Configurations](#) (英語) を参照してください。

12.1. 例

JavaScript

```
$fh.act({
  "act": "getTweets", // Name of the Cloud function to call
  "req": {
    "qs": "feedhenry" // Set of key/value pairs that are passed as
    paramaters to the Cloud function
  },
  "secure": true, // Whether or not to use https for the Cloud call.
  Default: true
  "timeout": 25000 // timeout value specified in milliseconds. Default:
  60000 (60s)
}, function(res) {
  // Cloud call was successful. Alert the response
  alert('Got response from cloud:' + JSON.stringify(res));
}, function(msg,err) {
  // An error occured during the cloud call. Alert some debugging
```

```

information
    alert('Cloud call failed with error message:' + msg + '. Error
properties:' + JSON.stringify(err));
});

```

Android (Java)

```

//build the request object. The first parameter is the name of the cloud
side function to be called,
//the second parameter is the data parameter for the function
FHActRequest request = FH.buildActRequest("getTweets", new
JSONObject().put("qs", "feedhenry"));
//the request will be executed asynchronously
request.executeAsync(new FHActCallback() {
    @Override
    public void success(FHResponse res) {
        //the function to execute if the request is successful
        try {
            JSONArray resObj = res.getJson().getJSONArray("tweets");
            Log.d(TAG, resObj.toString(2));
            for(int i=0;i<resObj.length();i++){
                JSONObject event = resObj.getJSONObject(i);
                ...
            }
        } catch(Exception e){
            Log.e(TAG, e.getMessage(), e);
        }
    }

    @Override
    public void fail(FHResponse res) {
        //the function to execute if the request is failed
        Log.e(TAG, res.getErrorMessage(), res.getError());
    }
});

```

iOS (Objective-C)

```

FHActRequest * action = (FHActRequest *) [FH buildActRequest:@"getTweets"
WithArgs:[NSDictionary dictionaryWithObject:@"feedhenry" forKey:@"qs"]];
[action execAsyncWithSuccess:^(FHResponse * actRes){
    //the actRes will contain 10 tweets about "feedhenry"
    //the JSON response from the cloud will be parsed to NSDictionary
    automatically
    NSDictionary* resData = actRes.parsedResponse;
    NSArray * tweets = (NSArray *) [resData objectForKey:@"tweets"];
    //display tweets in the UI
    ...
} AndFailure:^(FHResponse * actFailRes){
    //if there is any error, you can check the rawResponse string
    NSLog(@"Failed to read tweets. Response = %@", actFailRes.rawResponse);
}
];

```

//You can also use the delegate pattern with the FHActRequet object. If you use that pattern, you need to implement the FHResponseDelegate protocol and assign an instance to the FHActRequest instance. When the request is executed, replace the blocks with __nil__.