



Red Hat Mobile Application Platform ホ スト型 3

クラウド API

Red Hat Mobile Application Platform ホスト型 3 向け

Red Hat Mobile Application Platform ホスト型3 クラウド API

Red Hat Mobile Application Platform ホスト型 3 向け

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 クラウド API の参考情報を提供します。

目次

第1章 \$FH.CACHE	8
1.1. キャッシュオブションの管理	8
1.2. 例	8
第2章 \$FH.DB	10
2.1. 例	10
第3章 \$FH.FORMS	19
3.1. \$FH.FORMS.GETFORMS	19
3.1.1. 例	19
3.2. \$FH.FORMS.GETFORM	19
3.2.1. 詳細	19
3.2.2. 例	20
3.3. \$FH.FORMS.GETPOPULATEDFORMLIST	20
3.3.1. 詳細	20
3.3.2. 例	20
3.4. \$FH.FORMS.GETSUBMISSIONS	20
3.4.1. 詳細	20
3.4.2. submissionsObject	20
3.4.3. 例	21
3.5. \$FH.FORMS.GETSUBMISSION	21
3.5.1. 詳細	21
3.5.2. 例	21
3.6. \$FH.FORMS.GETSUBMISSIONFILE	21
3.6.1. 詳細	21
3.6.2. fileStreamObject	22
3.6.3. 例	22
3.7. フォーム JSON の定義	22
3.7.1. ページ	23
3.7.2. フィールド	23
3.7.3. ページルール	23
3.7.4. フィールドルール	24
3.8. \$FH.FORMS.GETTHEME	24
3.8.1. 詳細	24
3.8.2. 例	24
3.8.3. \$fh.forms.getAppClientConfig	25
3.8.4. 詳細	25
3.8.5. 例	25
3.8.6. クライアント設定の JSON オブジェクト	25
3.9. \$FH.FORMS.SUBMITFORMDATA	25
3.9.1. 詳細	26
3.9.1.1. 例	26
3.9.2. 提出の JSON オブジェクト	26
3.9.3. フィールドエントリーの JSON オブジェクト	26
3.9.4. フィールド値のエントリー	26
3.10. \$FH.FORMS.GETSUBMISSIONSTATUS	27
3.10.1. 詳細	28
3.10.2. 例	28
3.10.3. 提出ステータスの JSON オブジェクト	28
3.11. \$FH.FORMS.SUBMITFORMFILE	28
3.11.1. 詳細	28
3.11.2. 例	29

3.11.3. submitFileResult JSON オブジェクト	29
3.12. \$FH.FORMS.COMPLETESUBMISSION	29
3.12.1. 詳細	29
3.12.2. 例	30
3.13. \$FH.FORMS.CREATESUBMISSIONMODEL	30
3.13.1. 詳細	30
3.13.2. 例	30
3.14. \$FH.FORMS.REGISTERLISTENER	31
3.14.1. 詳細	31
3.14.2. イベント: submissionStarted	32
3.14.3. イベント: submissionComplete	32
3.14.4. イベント: submissionError	33
3.14.4.1. 提出エラーのタイプ	33
3.14.4.1.1. 検証エラー	33
3.14.4.1.2. 提出の JSON 定義保存での他のエラー	34
3.14.4.1.3. ファイルアップロードのエラー	34
3.15. \$FH.FORMS.DEREGISTERLISTENER	34
3.15.1. 詳細	34
3.16. \$FH.FORMS.EXPORTCSV	35
3.16.1. 詳細	35
3.17. \$FH.FORMS.EXPORTSINGLEPDF	36
3.17.1. 詳細	36
第4章 \$FH.HASH	37
4.1. 例	37
第5章 \$FH.HOST	38
5.1. 例	38
第6章 \$FH.PUSH	39
6.1. 例	39
6.2. パラメーター	39
6.2.1. 通知	39
6.2.2. iOS 固有	40
6.2.3. Windows 固有	40
6.2.4. 他の設定	41
6.2.5. プロジェクトでのクライアントアプリの選択	41
6.2.6. 受信者のフィルタリング	41
6.2.7. 応答処理	41
第7章 \$FH.SEC	42
7.1. 例	42
第8章 \$FH.SERVICE	44
8.1. 例	44
第9章 \$FH.STATS	45
9.1. 例	45
第10章 \$FH.SYNC	46
10.1. \$FH.SYNC.SETCONFIG	47
10.1.1. 使用法	47
10.1.2. パラメーター	48
10.1.2.1. options	48
10.1.3. 例	52

10.2. \$FH.SYNC.CONNECT	52
10.2.1. 使用法	52
10.2.2. パラメーター	52
10.2.2.1. mongodbConnectionString	52
10.2.2.2. mongodbConf	53
10.2.2.3. redisConnectionString	53
10.2.2.4. コールバック	53
10.2.3. 例	53
10.3. \$FH.SYNC.INIT	53
10.3.1. 使用法	54
10.3.2. パラメーター	54
10.3.2.1. dataset_id	54
10.3.2.2. options	54
10.3.2.3. コールバック	55
10.3.3. 例	55
10.3.4. \$fh.sync.init (バージョン 7.0.0 以前の fh-mbaas-api 向け)	55
10.3.4.1. 例	55
10.4. \$FH.SYNC.INVOKE	56
10.4.1. 使用法	56
10.4.2. パラメーター	56
10.4.2.1. dataset_id	56
10.4.2.2. params	56
10.4.3. 例	56
10.4.4. \$fh.sync.invoke (バージョン 7.0.0 以前の fh-mbaas-api 向け)	57
10.4.4.1. 例	57
10.5. \$FH.SYNC.STOP	57
10.5.1. 使用法	57
10.5.2. パラメーター	57
10.5.2.1. dataset_id	57
10.5.2.2. コールバック	58
10.5.3. 例	58
10.5.4. \$fh.sync.stop (バージョン 7.0.0 以前の fh-mbaas-api 向け)	58
10.5.4.1. 例	58
10.6. \$FH.SYNC.STOPALL	58
10.6.1. 使用法	58
10.6.2. パラメーター	59
10.6.2.1. コールバック	59
10.6.3. 例	59
10.6.4. \$fh.sync.stopAll (バージョン 7.0.0 以前の fh-mbaas-api 向け)	59
10.6.4.1. 例	59
10.7. \$FH.SYNC.HANDLELIST	59
10.7.1. 使用法	60
10.7.2. パラメーター	60
10.7.2.1. dataset_id	60
10.7.2.2. listHandler	60
10.7.3. 例	61
10.7.4. \$fh.sync.handleList (バージョン 7.0.0 以前の fh-mbaas-api 向け)	62
10.7.4.1. 例	62
10.8. \$FH.SYNC.GLOBALHANDLELIST	63
10.8.1. 使用法	63
10.8.2. パラメーター	64
10.8.2.1. listHandler	64
10.8.3. 例	64

10.8.4. \$fh.sync.globalHandleList (バージョン 7.0.0 以前の fh-mbaas-api 向け)	65
10.8.4.1. 例	65
10.9. \$FH.SYNC.HANDLECREATE	66
10.9.1. 使用法	66
10.9.2. パラメーター	66
10.9.2.1. dataset_id	66
10.9.2.2. createHandler	66
10.9.3. 例	67
10.9.4. \$fh.sync.handleCreate (バージョン 7.0.0 以前の fh-mbaas-api 向け)	67
10.9.4.1. 例	67
10.10. \$FH.SYNC.GLOBALHANDLECREATE	68
10.10.1. 使用法	68
10.10.2. パラメーター	68
10.10.2.1. createHandler	68
10.10.3. 例	68
10.10.4. \$fh.sync.globalHandleCreate (バージョン 7.0.0 以前の fh-mbaas-api 向け)	69
10.10.4.1. 例	69
10.11. \$FH.SYNC.HANDLEREAD	69
10.11.1. 使用法	69
10.11.2. パラメーター	69
10.11.2.1. dataset_id	69
10.11.2.2. readHandler	70
10.11.3. 例	70
10.11.4. \$fh.sync.handleRead (バージョン 7.0.0 以前の fh-mbaas-api 向け)	71
10.11.4.1. 例	71
10.12. \$FH.SYNC.GLOBALHANDLEREAD	71
10.12.1. 使用法	71
10.12.2. パラメーター	71
10.12.2.1. readHandler	72
10.12.3. 例	72
10.12.4. \$fh.sync.globalHandleRead (バージョン 7.0.0 以前の fh-mbaas-api 向け)	72
10.12.4.1. 例	73
10.13. \$FH.SYNC.HANDLEUPDATE	73
10.13.1. 使用法	73
10.13.2. パラメーター	73
10.13.2.1. dataset_id	73
10.13.2.2. updateHandler	73
10.13.3. 例	74
10.13.4. \$fh.sync.handleUpdate (バージョン 7.0.0 以前の fh-mbaas-api 向け)	74
10.13.4.1. 例	75
10.14. \$FH.SYNC.GLOBALHANDLEUPDATE	75
10.14.1. 使用法	75
10.14.2. パラメーター	75
10.14.2.1. updateHandler	75
10.14.3. 例	76
10.14.4. \$fh.sync.globalHandleUpdate (バージョン 7.0.0 以前の fh-mbaas-api 向け)	77
10.14.4.1. 例	77
10.15. \$FH.SYNC.HANDLEDELETE	77
10.15.1. 使用法	77
10.15.2. パラメーター	77
10.15.2.1. dataset_id	77
10.15.2.2. deleteHandler	77
10.15.3. 例	78

10.15.4. \$fh.sync.handleDelete (バージョン 7.0.0 以前の fh-mbaas-api 向け)	78
10.15.4.1. 例	78
10.16. \$FH.SYNC.GLOBALHANDLEDELETE	79
10.16.1. 使用法	79
10.16.2. パラメーター	79
10.16.2.1. deleteHandler	79
10.16.3. 例	80
10.16.4. \$fh.sync.globalHandleDelete (バージョン 7.0.0 以前の fh-mbaas-api 向け)	80
10.16.4.1. 例	81
10.17. \$FH.SYNC.HANDLECOLLISION	81
10.17.1. 使用法	81
10.17.2. パラメーター	81
10.17.2.1. dataset_id	81
10.17.2.2. collisionHandler	81
10.17.3. 例	82
10.17.4. \$fh.sync.handleCollision (バージョン 7.0.0 以前の fh-mbaas-api 向け)	82
10.17.4.1. 例	83
10.18. \$FH.SYNC.GLOBALHANDLECOLLISION	83
10.18.1. 使用法	83
10.18.2. パラメーター	83
10.18.2.1. collisionHandler	84
10.18.3. 例	84
10.18.4. \$fh.sync.globalHandleCollision (バージョン 7.0.0 以前の fh-mbaas-api 向け)	85
10.18.4.1. 例	85
10.19. \$FH.SYNC.LISTCOLLISIONS	85
10.19.1. 使用法	85
10.19.2. パラメーター	85
10.19.2.1. dataset_id	86
10.19.2.2. listCollisionsHandler	86
10.19.3. 例	86
10.19.4. \$fh.sync.listCollisions (バージョン 7.0.0 以前の fh-mbaas-api 向け)	87
10.19.4.1. 例	87
10.20. \$FH.SYNC.GLOBALLISTCOLLISIONS	88
10.20.1. 使用法	88
10.20.2. パラメーター	88
10.20.2.1. listCollisionsHandler	88
10.20.3. 例	88
10.20.4. \$fh.sync.globalListCollisions (バージョン 7.0.0 以前の fh-mbaas-api 向け)	89
10.20.4.1. 例	89
10.21. \$FH.SYNC.REMOVECOLLISION	89
10.21.1. 使用法	90
10.21.2. パラメーター	90
10.21.2.1. dataset_id	90
10.21.2.2. removeCollisionHandler	90
10.21.3. 例	90
10.21.4. \$fh.sync.removeCollision (バージョン 7.0.0 以前の fh-mbaas-api 向け)	91
10.21.4.1. 例	91
10.22. \$FH.SYNC.GLOBALREMOVECOLLISION	91
10.22.1. 使用法	91
10.22.2. パラメーター	91
10.22.2.1. removeCollisionHandler	92
10.22.3. 例	92
10.22.4. \$fh.sync.globalRemoveCollision (バージョン 7.0.0 以前の fh-mbaas-api 向け)	92

10.22.4.1. 例	93
10.23. \$FH.SYNC.INTERCEPTREQUEST	93
10.23.1. 使用法	93
10.23.2. パラメーター	93
10.23.2.1. dataset_id	93
10.23.2.2. requestInterceptor	93
10.23.3. 例	94
10.23.4. \$fh.sync.interceptRequest (バージョン 7.0.0 以前の fh-mbaas-api 向け)	94
10.23.4.1. 例	94
10.24. \$FH.SYNC.SETRECORDHASHFN	95
10.24.1. 使用法	95
10.24.2. パラメーター	95
10.24.2.1. dataset_id	95
10.24.2.2. generateHash	95
10.24.3. Returns	96
10.24.4. 例	96
10.25. \$FH.SYNC.SETGLOBALHASHFN	96
10.25.1. 使用法	96
10.25.2. パラメーター	96
10.25.2.1. dataset_id	96
10.25.2.2. generateHash	96
10.25.3. Returns	97
10.25.4. 例	97
10.26. \$FH.SYNC.GLOBALINTERCEPTREQUEST	97
10.26.1. 使用法	97
10.26.2. パラメーター	97
10.26.2.1. requestInterceptor	97
10.26.3. 例	98
10.26.4. \$fh.sync.globalInterceptRequest (バージョン 7.0.0 以前の fh-mbaas-api 向け)	98
10.26.4.1. 例	98
10.27. \$FH.SYNC.GETEVENTEMITTER	99
10.27.1. 使用法	99
10.27.2. パラメーター	99
10.27.3. 例	99

第1章 \$FH.CACHE

```
$fh.cache(options, callback);
```

クラウドアプリのキャッシュ層にある値を保存、読み込み、削除します。

デフォルトでは、キャッシュストアのメモリー上限は 1GB です。この上限に達すると、LRU (最も長く使われていない) アルゴリズムを使用してデータの削除を開始します。

キャッシュに値を保存する際に "expire" オプションが指定されていない場合は、その値はメモリー上限に達するまでキャッシュにとどまります。その後は、どの値を削除するかは LRU アルゴリズムで判断されます。

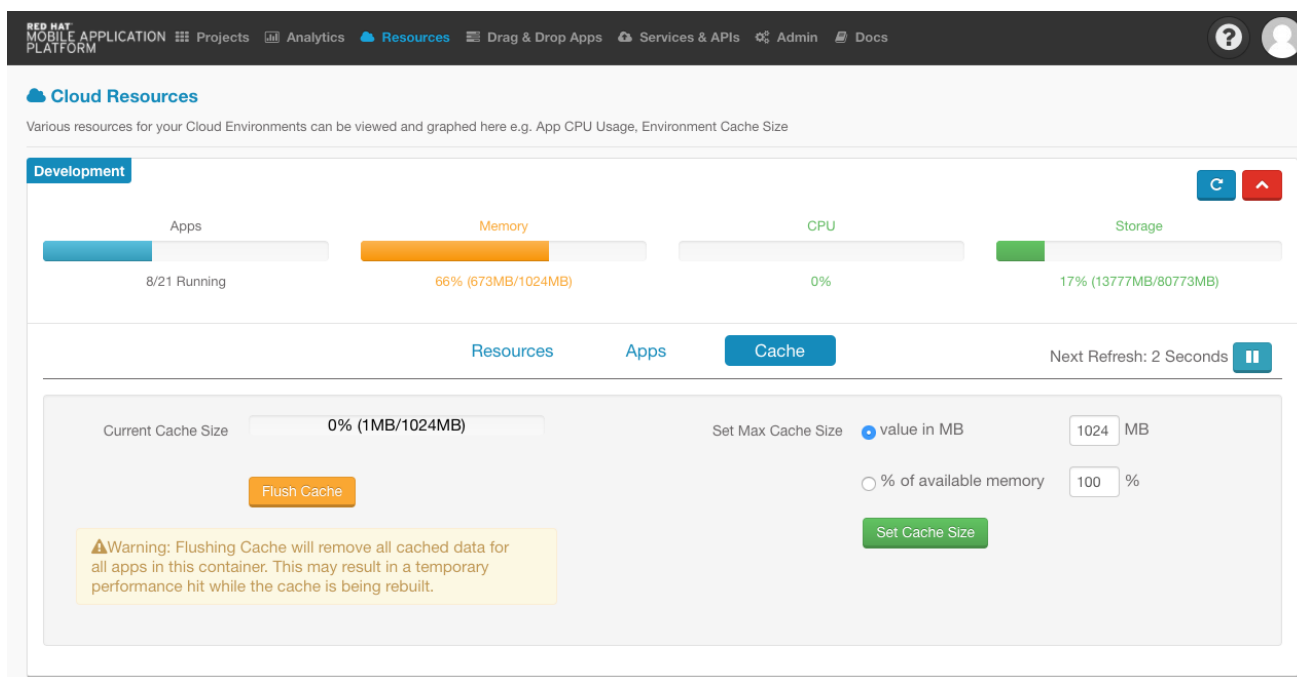
1.1. キャッシュオプションの管理

キャッシュのメモリー上限は Studio で管理が可能で、キャッシュストアのフラッシュもできます。これら 2 つのオプションは、リソース > [環境] > キャッシュ で設定できます。



注記

キャッシュの最大サイズの設定は、環境内の全クラウドアプリに影響します。



1.2. 例

値をキャッシュに保存

```
var options = {
  "act": "save",
  "key": "foo", // The key associated with the object
  "value": "bar", // The value to be cached, must be serializable
  "expire": 60 // Expiry time in seconds. Optional
};
$fh.cache(options, function (err, res) {
```

```
    if (err) return console.error(err.toString());

    // res is the original cached object
    console.log(res.toString());
  });
```

キャッシュから値を読み込み

```
var options = {
  "act": "load",
  "key": "foo" // key to look for in cache
};
$fh.cache(options, function (err, res) {
  if (err) return console.error(err.toString());

  // res is the original cached object
  console.log(res.toString());
});
```

キャッシュから値を削除

```
var options = {
  "act": "remove",
  "key": "foo" // key to look for in cache
};
$fh.cache(options, function (err, res) {
  if (err) return console.error(err.toString());

  // res is the removed cached object
  console.log(res.toString());
});
```

第2章 \$FH.DB

```
$fh.db(options, callback);
```

ホストされているデータストレージにアクセスします。CRUDL (作成、読み込み、更新、削除、一覧表示) およびインデックス操作をサポートします。また、指定されたエンティティ内の全レコードを削除する `deleteall` もサポートします。

2.1. 例

単一エントリー (row) を作成

```
var options = {
  "act": "create",
  "type": "myFirstEntity", // Entity/Collection name
  "fields": { // The structure of the entry/row data. A data is analogous
    to "Row" in MySql or "Documents" in MongoDB
    "firstName": "Joe",
    "lastName": "Bloggs",
    "address1": "22 Blogger Lane",
    "address2": "Bloggsville",
    "country": "Bloggland",
    "phone": "555-123456"
  }
};
$fh.db(options, function (err, data) {
  if (err) {
    console.error("Error " + err);
  } else {
    console.log(JSON.stringify(data));
    /*
      The output will be something similar to this
      {
        "fields": {
          "address1": "22 Blogger Lane",
          "address2": "Bloggsville",
          "country": "Bloggland",
          "fistName": "Joe",
          "lastName": "Bloggs",
          "phone": "555-123456"
        },
        "guid": "4e563ea44fe8e7fc190000002", // unique id for this data
entry    "type": "myFirstEntity"
      }
    */
  }
});
```

単一コールで複数レコードを作成

```
var options = {
  "act": "create",
  "type": "myCollectionType", // Entity/Collection name
```

```

    "fields": [{ // Notice 'fields' is an array of data entries
      "id": 1,
      "name": "Joe"
    }, {
      "id": 2,
      "name": "John"
    }]
  };
  $fh.db(options, function (err, data) {
    if (err) {
      console.error("Error " + err);
    } else {
      console.log(JSON.stringify(data));
      /*
        The output will be something similar to this
        {
          "status": "ok",
          "count": 2
        }
      */
    }
  });

```

単一のエントリーを読み取り

```

var options = {
  "act": "read",
  "type": "myFirstEntity", // Entity/Collection name
  "guid": "4e563ea44fe8e7fc190000002" // Row/Entry ID
};
$fh.db(options, function (err, data) {
  if (err) {
    console.error("Error " + err);
  } else {
    console.log(JSON.stringify(data));
    /* Sample output
      {
        "fields": {
          "address1": "22 Blogger Lane",
          "address2": "Bloggsville",
          "country": "Bloggland",
          "firstName": "Joe",
          "lastName": "Bloggs",
          "phone": "555-123456"
        },
        "guid": "4e563ea44fe8e7fc190000002",
        "type": "myFirstEntity"
      }
    */
  }
});

```

エントリー全体を更新

```

// The update call updates the entire entity.

```

```
// It will replace all the existing fields with the new fields passed in.
var options = {
  "act": "update",
  "type": "myFirstEntity", // Entity/Collection name
  "guid": "4e563ea44fe8e7fc190000002", // Row/Entry ID
  "fields": {
    "firstName": "Jane"
  }
};
$fh.db(options, function (err, data) {
  if (err) {
    console.error("Error " + err);
  } else {
    console.log(JSON.stringify(data));
    /* Output:
      {
        "fields": {
          "firstName": "Jane"    //only one field now
        },
        "guid": "4e563ea44fe8e7fc190000002",
        "type": "myFirstEntity"
      }
    */
  }
});
```

単一フィールドを更新

```
var options = {
  "act": "read",
  "type": "myFirstEntity", // Entity/Collection name
  "guid": "4e563ea44fe8e7fc190000002" // Row/Entry ID
};
$fh.db(options, function (err, entity) {
  var entFields = entity.fields;
  entFields.firstName = 'Jane';

  options = {
    "act": "update",
    "type": "myFirstEntity",
    "guid": "4e563ea44fe8e7fc190000002",
    "fields": entFields
  };
  $fh.db(options, function (err, data) {
    if (err) {
      console.error("Error " + err);
    } else {
      console.log(JSON.stringify(data));
      /*output
      {
        "fields": {
          "address1": "22 Blogger Lane",
          "address2": "Bloggsville",
          "country": "Bloggland",
          "firstName": "Jane",
          "lastName": "Bloggs",

```



```

        "phone": "555-123456"
      },
      "guid": "4e563ea44fe8e7fc19000002",
      "type": "myFirstEntity"
    }
  }
  */
}
});
});

```

エントリー (row) を削除

```

var options = {
  "act": "delete",
  "type": "myFirstEntity", // Entity/Collection name
  "guid": "4e563ea44fe8e7fc19000002" // Row/Entry ID to delete
};
$fh.db(, function (err, data) {
  if (err) {
    console.error("Error " + err);
  } else {
    console.log(JSON.stringify(data));
    /* output
    {
      "fields": {
        "address1": "22 Blogger Lane",
        "address2": "Bloggsville",
        "country": "Bloggland",
        "fistName": "Jane",
        "lastName": "Bloggs",
        "phone": "555-123456"
      },
      "guid": "4e563ea44fe8e7fc19000002",
      "type": "myFirstEntity"
    }
    */
  }
});

```

エンティティー (集合) の全エントリーを削除

```

var options = {
  "act": "deleteall",
  "type": "myFirstEntity" // Entity/Collection name
};
$fh.db(options, function (err, data) {
  if (err) {
    console.error("Error " + err);
  } else {
    console.log(JSON.stringify(data));
    /* output
    {
      status: "ok",
      count: 5
    }
    */
  }
});

```

```

    */
  }
});

```

一覧表示

```

var options = {
  "act": "list",
  "type": "myFirstEntity", // Entity/Collection name
};
$fh.db(options, function (err, data) {
  if (err) {
    console.error("Error " + err);
  } else {
    console.log(JSON.stringify(data));
  }
  /* output
  {
    "count": 1,
    "list": [{
      "fields": {
        "address1": "22 Blogger Lane",
        "address2": "Bloggsville",
        "country": "Bloggland",
        "fistName": "Joe",
        "lastName": "Bloggs",
        "phone": "555-123456"
      },
      "guid": "4e563ea44fe8e7fc19000002",
      "type": "myFirstEntity"
    }]
  }
  */
});

```

ソート表示

```

var sort_ascending = {
  "act": "list",
  "type": "myFirstEntity", // Entity/Collection name
  "sort": {
    "username": 1 // Sort by the 'username' field ascending a-z
  }
};

var sort_descending = {
  "act": "list",
  "type": "myFirstEntity", // Entity/Collection name
  "sort": {
    "username": -1 // Sort by the 'username' field descending z-a
  }
};

```

ページネーションによる一覧表示

■

```

var options = {
  "act": "list",
  "type": "myFirstEntity", // Entity/Collection name
  "skip": 20, ❶
  "limit": 10 ❷
};
$fh.db(options, function (err, data) {
  if (err) {
    console.error("Error " + err);
  } else {
    console.log(JSON.stringify(data));
    /* output
    {
      "count": 10,
      "list": [{
        "fields": {
          "address1": "22 Blogger Lane",
          "address2": "Bloggsville",
          "country": "Bloggland",
          "firstName": "Joe",
          "lastName": "Bloggs",
          "phone": "555-123456"
        },
        "guid": "4e563ea44fe8e7fc19000002",
        "type": "myFirstEntity"
      }, ...
    ]
    */
  }
});

```

❶ 結果の 3 ページ目を返します

❷ 返されるページのサイズは 10

地理的検索による一覧表示

```

var options = {
  act: "list",
  type: "collectionName", // Entity/Collection name
  "geo": {
    "employeeLocation": { //employeeLocation has been indexed as "2D"
      center: [-83.028965, 42.542144],
      radius: 5 //km
    }
  }
};
$fh.db(options, function (err, data) {
  if (err) {
    console.error("Error " + err);
  } else {
    console.log(JSON.stringify(data));
    /* output

```

```

    {
      "count": 1,
      "list": [{
        "fields": {
          "address1": "22 Blogger Lane",
          "address2": "Bloggsville",
          "country": "Bloggland",
          "firstName": "Joe",
          "lastName": "Bloggs",
          "phone": "555-123456"
        },
        "guid": "4e563ea44fe8e7fc190000002",
        "type": "myFirstEntity"
      }]
    }
  */
});

```

複数の制限による一覧表示

```

/* Possible query restriction types:
  "eq" - is equal to
  "ne" - not equal to
  "lt" - less than
  "le" - less than or equal
  "gt" - greater than
  "ge" - greater than or equal
  "like" Match some part of the field. Based on [Mongo regex matching
logic]
(http://www.mongodb.org/display/DOCS/Advanced+Queries#AdvancedQueries-
RegularExpressions)
  "in" - The same as $in operator in MongoDB, to select documents where
the field (specified by the _key_) equals any value in an array (specified
by the _value_)
*/
var options = {
  "act": "list",
  "type": "myFirstEntity", // Entity/Collection name
  "eq": {
    "lastName": "Bloggs"
  },
  "ne": {
    "firstName": "Jane"
  },
  "in": {
    "country": ["Bloggland", "Janeland"]
  }
};
$fh.db(options, function (err, data) {
  if (err) {
    console.error("Error " + err);
  } else {
    console.log(JSON.stringify(data));
  }
  /* output
  {

```

```

        "count": 1,
        "list": [{
            "fields": {
                "address1": "22 Blogger Lane",
                "address2": "Bloggsville",
                "country": "Bloggland",
                "firstName": "Joe",
                "lastName": "Bloggs",
                "phone": "555-123456"
            },
            "guid": "4e563ea44fe8e7fc190000002",
            "type": "myFirstEntity"
        }]
    }
    */
}
});

```

返された制限付きフィールドによる一覧表示

```

var options = {
    "act": "list",
    "type": "myFirstEntity",
    "eq": {
        "lastName": "Bloggs",
        "country": "Bloggland"
    },
    "ne": {
        "firstName": "Jane"
    },
    "fields": ["address1", "address2"]
};
$fh.db(options, function (err, data) {
    if (err) {
        console.error("Error " + err);
    } else {
        console.log(JSON.stringify(data));
    }
    /* output
    {
        "count": 1,
        "list": [{
            "fields": {
                "address1": "22 Blogger Lane",
                "address2": "Bloggsville"
            },
            "guid": "4e563ea44fe8e7fc190000002",
            "type": "myFirstEntity"
        }]
    }
    */
}
});

```

インデックスを追加

```
var options = {
  "act": "index",
  "type": "employee",
  "index": {
    "employeeName": "ASC" // Index type: ASC - ascending, DESC -
descending, 2D - geo indexation
    "location": "2D"
    // For 2D indexing, the field must satisfy the following:
    // It is a [Longitude, Latitude] array
    // Longitude should be between [-180, 180]
    // Latitude should be between [-90, 90]
    // Latitude or longitude should **NOT** be null
  }
};
$fh.db(options, function (err, data) {
  if (err) {
    console.error("Error " + err);
  } else {
    console.log(JSON.stringify(data));
    /* output
      {
        "status": "ok",
        "indexName": "employeeName_1_location_2d"
      }
    */
  }
});
```

第3章 \$FH.FORMS

3.1. \$FH.FORMS.GETFORMS

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

フォームサマリー情報のある JSON オブジェクトの配列を返します。



注記

これらのフォームサマリー JSON オブジェクトには、完全フォーム定義が含まれていません。完全フォーム定義を取得するには、\$fh.forms.getForm 関数を使用してください。

3.1.1. 例

```
//Get a list of forms associated with the project.
var options = {

};

$fh.forms.getForms(options,

/*
 * Function executed with forms.
 */
function (err, response) {
  if (err) return handleError(err);

  //An Array Of Forms Associated With The Project
  var formsArray = response.forms;

  /*
   exampleForm: {
     _id: <<Form Id>>,
     name: <<Form Name>>,
     description: <<Form Description>>
     lastUpdatedTimestamp: <<Timestamp of when the form was last
updated>>
   }
  */
  var exampleForm = formsArray[0];

  return callback(undefined, formsArray);
});
```

3.2. \$FH.FORMS.GETFORM

```
$fh.forms.getForm(options, callback)
```

3.2.1. 詳細

フォーム ID に基づいて特定のフォームモデルを取得します。フォーム ID は、\$fh.forms.getForms 関数を使用して取得できます。

3.2.2. 例

```
$fh.forms.getForm({
  "_id": formId
}, function (err, form) {
  if (err) return handleError(err);

  /*
   A JSON object describing a full form object.
  */
  return callback(undefined, form);
});
```

3.3. \$FH.FORMS.GETPOPULATEDFORMLIST

```
$fh.forms.getPopulatedFormList(options, callback)
```

3.3.1. 詳細

フォーム ID 一覧に基づいてフォームモデルを取得します。

3.3.2. 例

```
$fh.forms.getPopulatedFormList({
  "formids": [formId1, formId2 ... ]
}, function (err, arrayOfForms) {
  if (err) return handleError(err);

  /*
   A JSON object describing a full form object.
  */
  return callback(undefined, arrayOfForms);
});
```

3.4. \$FH.FORMS.GETSUBMISSIONS

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

3.4.1. 詳細

フィルター条件に基づいて全提出を一覧表示します。

3.4.2. submissionsObject


```
{
  submissions: [<SubmissionJSON>, <SubmissionJSON>]
}
```

3.4.3. 例

```
$fh.forms.getSubmissions({
  "formId": [formId1, formId2 ... ],
  "subId": [subId1, subId2 ...]
}, function (err, submissionsObject) {
  if (err) return handleError(err);

  /*
   An Object Containing An Array of JSON objects describing a full
  Submission object.
  */
  return callback(undefined, submissionsObject);
});
```

3.5. \$FH.FORMS.GETSUBMISSION

```
$fh.forms.getSubmission(options, callback)
```

3.5.1. 詳細

単一のフォーム提出を取得します。

3.5.2. 例

```
$fh.forms.getSubmissions({
  "submissionId": subId1
}, function (err, submission) {
  if (err) return handleError(err);

  /*
   A JSON object describing a full Submission object.
  */
  return callback(undefined, submission);
});
```

3.6. \$FH.FORMS.GETSUBMISSIONFILE

```
$fh.forms.getSubmissionFile(options, callback)
```

3.6.1. 詳細

提出に含まれる単一ファイルをストリームします。ファイルには、提出ファイルフィールド内の fileGroupId フィールドを使用してアクセスします。

3.6.2. fileStreamObject

```
{
  stream: <Readable File Stream>
}
```

3.6.3. 例

```
$fh.forms.getSubmissionFile({
  "_id": fileGroupId
}, function (err, fileStreamObject) {
  if (err) return handleError(err);

  /**
   * Pipe the file stream to a writable stream (for example, a FileWriter)
   */
  fileStreamObject.stream.pipe(writable_stream);
  fileStreamObject.stream.resume();
});
```

3.7. フォーム JSON の定義

フォーム JSON オブジェクトには、フォーム処理に必要なすべての情報が含まれています。

```
{
  "_id": "<<24 Character Form ID>>",
  "description": "This is an example form definition",
  "name": "Example Form",
  "updatedBy": "<<User ID of the person that last updated the form>>",
  "lastUpdatedTimestamp": 1410876316105, //Time the form was last updated.
  "subscribers": [
    //Email addresses to be notified when a submission has been made
    //against this form.
    "notifiedUser1@example.com",
    "notifiedUser2@example.com"
  ],
  "pageRules": [
    <<Page Rule JSON Object>>
  ],
  "fieldRules": [
    <<Field Rule JSON Object>>
  ],
  "pages": [
    <<Page JSON Object>>,
  ],
  //Convenient reference for the index of a page with <<Page Id>> in the
  "pages" array.
  "pageRef": {
    "<<Page Id>>": 0,
    "<<Page Id>>": 1
  },
  //Convenient reference for the index of a field. Both the page index
  and field index are given.
```

```

    "fieldRef":{
      "<<Field Id>>":{
        "page":0,
        "field":0
      },
      "<<Field Id>>":{
        "page":0,
        "field":1
      }
    }
  }
}

```

3.7.1. ページ

```

{
  "_id": "<<Page ID>>",
  "name": "Page 1",
  "fields": [
    <<Field JSON Object>>
  ]
}

```

3.7.2. フィールド

```

{
  "_id": "<<Field ID>>",
  "required": true,
  "type": "text", //Field Type
  "name": "A Sample Text Field",
  "repeating": false/true //Boolean that describes if a field is repeating
or not.
  "fieldOptions": {
    "validation": { // Optional validation parameters for the form.
      "validateImmediately": true //Flag for whether field inputs should
be immediately validated (for example, On-Blur on a Client App.)
    },
    "definition": { // Optional definition options.
      "minRepeat": 2, //Minimum number of entries for this field.
      "maxRepeat": 5 //Maximum number of entries for this field.
    }
  }
}
}

```

3.7.3. ページルール

この JSON オブジェクトは、Studio 内で作成されるページルールを記述します。

```

{
  "type": "skip", //A "skip" or "show" page rule
  "_id": "<<ID of the Page Rule>>",
  "targetPage": [
    "<<ID of the Page targeted by the Page Rule>>"
  ],
}

```

```

    "ruleConditionalStatements":[
      {
        "sourceField":"<<ID of the Field this condition is sourcing data
from>>",
        "restriction":"is",// Comparator operator for the conditional
statement.
        "sourceValue":"skippage" //Value To Compare.
      }
    ],
    //Combinator for "ruleConditionalStatements". Can be "and" or "or".
    "ruleConditionalOperator":"and",
  }

```

3.7.4. フィールドルール

この JSON オブジェクトは、Studio 内で作成されるフィールドルールを記述します。

```

{
  "type":"hide/show", //A "hide" or "show" field rule
  "_id":"<<ID of the Field Rule>>",
  "targetField":[
    "<<ID of the Field targeted by the Field Rule>>"
  ],
  "ruleConditionalStatements":[
    {
      "sourceField":"<<ID of the Field this condition is sourcing data
from>",
      "restriction":"is",// Comparator operator for the conditional
statement.
      "sourceValue":"hideMe" //Value To Compare.
    }
  ],
  //Combinator for "ruleConditionalStatements". Can be "and" or "or".
  "ruleConditionalOperator":"and"
}

```

3.8. \$FH.FORMS.GETTHEME

```
$fh.forms.getTheme(options, callback)
```

3.8.1. 詳細

プロジェクトに割り当てられているテーマを表す JSON オブジェクトを読み込みます。

3.8.2. 例

```

//Currently no parameters for loading a theme.
var options = {

};

$fh.forms.getTheme({}, function (err, theme) {
  if (err) return handleError(err);

```

```
    return callback(undefined, theme);
  });
```

3.8.3. \$fh.forms.getAppClientConfig

```
$fh.forms.getAppClientConfig(options, callback)
```

3.8.4. 詳細

プロジェクトに関連付けられているクライアント設定に含まれる JSON オブジェクトを返します。これらは Studio 内で設定されます。

3.8.5. 例

```
//Currently no options for loading app config.
var options = {

};

$fh.forms.getAppClientConfig(options, function (err, clientConfig) {
  if (err) return handleError(err);

  return callback(undefined, clientConfig);
});
```

3.8.6. クライアント設定の JSON オブジェクト

```
{
  "sent_items_to_keep_list": [5, 10, 20, 30, 40, 50, 100], //Array
  representing options available to
  "targetWidth": 480, //Camera Photo Width
  "targetHeight": 640, //Camera Photo Height
  "quality": 75, //Camera Photo Quality
  "debug_mode": false, //Set the Client To Debug Mode
  "logger" : false, //Client Logging
  "max_retries" : 0, //Maximum number of failed uplod attempts before
  returning an error to the user
  "timeout" : 30, // Number of seconds before a form upload times out.
  "log_line_limit": 300, //Maximum number of log entries before rotating
  logs
  "log_email": "test@example.com" //The email address that logs are sent
  to.
}
```

3.9. \$FH.FORMS.SUBMITFORMDATA

```
$fh.forms.submitFormData(options, callback)
```

3.9.1. 詳細

フォーム提出を表す JSON オブジェクトを提出します。

3.9.1.1. 例

```
var options = {
  "submission": <<Submission JSON Object>>,
  "appId": <<ID Of The Client Making The Submission.>>
};

$fh.forms.submitFormData(options, function(err,data){
  if(err) return callback(err);

  return callback(null,data);
});
```

3.9.2. 提出の JSON オブジェクト

```
{
  "formId": "<<ID Of Form Submitting Againsts>>",
  "deviceId": "<<ID of the device submitting the form>>",
  "deviceIpAddress": "<<IP Address of the device submitting the form>>",
  "formFields": [<<Field Entry JSON Object>>],
  "deviceFormTimestamp": "<<lastUpdatedTimestamp of the Form that the submission was submitted against.>>",
  "comments": [{ //Optional comments related to the submission
    "madeBy": "user",
    "madeOn": "12/11/10",
    "value": "This is a comment"
  }]
}
```

3.9.3. フィールドエントリーの JSON オブジェクト

```
{
  fieldId: <<ID Of The Field "fieldValues" Are Submitted Against>>,
  fieldValues: [<<Field Value Entry>>]
}
```

3.9.4. フィールド値のエントリー

以下では、各タイプのフィールド提出に必須のデータ形式を示しています。

- Text: 文字列
- TextArea: 文字列
- Number: 数値
- Radio: 文字列 (フォームで定義されている Radio フィールドのいずれかのオプションであること)

- Dropdown: 文字列 (フォームで表示されている Dropdown のいずれかのオプションであること)
- Webstite: 文字列
- Email: 文字列 (有効な E メール形式であること)
- DateStamp - 日付のみ: 文字列 (形式: **DD/MM/YYYY**)
- DateStamp - 時間のみ: 文字列 (形式: **HH:SS**)
- DateStamp - 日付 & 時間: 文字列 (形式: **YYYY-MM-DD HH:SS**)

チェックボックス

```
{
  selections: ["Check box Option 1", ...., "Check box Option 4"]
}
```

位置 (およびマップフィールド) - 緯度 & 経度

```
{
  lat: <<Valid Latitude Value>,
  long: <<Valid Longitude Value>>
}
```

位置 - 偏北距離 & 偏東距離

```
{
  zone: "11U",
  eastings: 594934,
  northings: 5636174
}
```

ファイルベースのフィールド - ファイル、写真、署名

```
{
  fileName: <<Name of the file to be uploaded>>,
  fileType: <<Valid mime type of the file>>,
  fileSize: <<Size of the file in Bytes>>,
  fileUpdateTime: <<Timestamp of the time the file was saved to device>>,
  hashName: "filePlaceholder12345" //A unique identifier for the file.
NOTE: Must begin with "filePlaceholder"
}
```



注記

hashName パラメーターはすべて **filePlaceholder** で始まる必要があり、それ以外の場合は提出が拒否されます。

3.10. \$FH.FORMS.GETSUBMISSIONSTATUS

```
$fh.forms.getSubmissionStatus(options, callback)
```

3.10.1. 詳細

提出の現在のステータスを返します。

3.10.2. 例

```
var options = {
  submission: {
    //This is the submission ID returned when the $fh.forms.submitFormData
    function returns.
    submissionId: "<<Remote Submission ID>>"
  }
};

$fh.forms.getSubmissionStatus(options, function(err, submissionStatus){
  if(err) return handleError(err);

  return callback(undefined, submissionStatus);
});
```

3.10.3. 提出ステータスの JSON オブジェクト



注記

提出は、\$fh.forms.completeSubmission 関数が呼び出された後でのみ、完了とマークされます。このため、**pendingFiles** アレイが空白となり、**status** が保留として設定される場合があります。

```
{
  "status": "pending/complete", //Status can either be pending or complete
  "pendingFiles": [
    "<<hashName of file not uploaded yet. (for example,
    filePlaceholder1234)>>"
  ]
}
```

3.11. \$FH.FORMS.SUBMITFORMFILE

```
$fh.forms.submitFormFile(options, callback)
```

3.11.1. 詳細

提出にファイルをアップロードします。



注記

ファイルは \$fh.forms.submitFormData 関数を使用して提出 JSON オブジェクトに追加され、提出済みである必要があります。



注記

提出にアップロードするファイルは、ローカルファイルシステムに既に存在する必要があります。



警告

keepFile パラメーターが **true** に設定されていない場合は、アップロードが完了すると、提出にアップロードされるファイルはファイルシステムから削除されます。

3.11.2. 例

```
var options = {
  "submission": {
    "fileId": "<<The File hashName>>",
    "submissionId": "<<The Submission ID Containing The File Input>>",
    "fieldId": "<<Field Id The File Is Being Submitted Against>>",
    "fileStream": "<</path/to/the/file/stored/locally>>" //path to the
file
    "keepFile": true/false //Keep the file storated at "fileStream" when
it has been uploaded.
  }
}

$fh.forms.submitFormFile(options, function(err, submitFileResult){
  if(err) return handleError(err);

  //File upload has completed successfully
  return callback(undefined, submitFileResult);
});
```

3.11.3. submitFileResult JSON オブジェクト

```
{
  status: 200 //Indicating that the file has uploaded successfully
  savedFileGroupId: <<Server ID of the file held in the submission>>
}
```

3.12. \$FH.FORMS.COMPLETESUBMISSION

```
$fh.forms.completeSubmission(options, callback)
```

3.12.1. 詳細

提出を完了としてマークします。提出 JSON の一部として提出された全ファイルがアップロードされていることをチェックします。

提出が正常に完了していれば、**completeResult** JSON オブジェクトに以下が含まれます。

```
{
  "status": "complete"
}
```

提出されたファイルがアップロードされていない場合は、**completeResult** JSON オブジェクトに以下が含まれます。

```
{
  "status": "pending",
  "pendingFiles": [
    "<<hashName of file not uploaded yet. (for example,
    filePlaceHolder1234)>>"
  ]
}
```

3.12.2. 例

```
var options = {
  "submission": {
    "submissionId": "<<The ID of the Submission to Complete>>"
  }
}

$fh.forms.completeSubmission(options, function (err, completeResult) {
  if (err) return handleError(err);

  return callback(undefined, completeResult);
});
```

3.13. \$FH.FORMS.CREATESUBMISSIONMODEL

3.13.1. 詳細

`$fh.forms.createSubmissionModel` 関数は、フォームを作成および提出する別のメソッドになります。

提出モデルは、提出作成プロセスを容易にする便利な関数を提供します。

3.13.2. 例

```
var options = {
  "form": <<A Form JSON Object Obtained using $fh.forms.getForm>>
};

$fh.forms.createSubmissionModel(options, function(err, submissionModel){
  if (err) return handleError(err);

  //Now use the Submisison Model Functions To Add data to the Submission
  var fieldInputOptions = {
    "fieldId": "<<The ID of the field To Add Data To>>",
    "fieldCode": "<<The fieldCode of the field To Add Data To>>"
  }
```

```

    "index": "<<The index to add the value to>>" //(This is used for
repeating fields with mutiple values)
    "value": "<<A valid input value to add to the submission>>"
  };

  //Note: the addFieldInput function is not asynchronous
  var error = submissionModel.addFieldInput(fieldInputOptions);

  if(error){
    return handleError(error);
  }

  /*
  Submitting the data as part of a submission.
  This function will upload all files passed to the submission using the
  addFieldInput function
  */
  submissionModel.submit(function(err, submissionId){
    if(err) return handleError(err);

    return callback(undefined, submissionId);
  });
});

```

3.14. \$FH.FORMS.REGISTERLISTENER



注記

お使いの **package.json** の **fh-mbaas-api** のバージョンは、**4.8.0** 以降である必要があります。

3.14.1. 詳細

\$fh.forms.registerListener 関数を使用すると、**EventEmitter** オブジェクトを登録して、発生する提出イベントをリッスンすることができます。



注記

\$fh.forms.registerListener と **\$fh.forms.deregisterListener** 関数は **EventEmitter** オブジェクトをパラメーターとしてのみ受け付けます。

```

//NodeJS Events Module. Note, this is required to register event emitter
objects to forms.
var events = require('events');
var submissionEventListener = new events.EventEmitter();

$fh.forms.registerListener(submissionEventListener, function(err){
  if (err) return handleError(err);

  //submissionEventListener has now been registered with the $fh.forms
  Cloud API. Any valid Forms Events will now emit.
});

```

3.14.2. イベント: **submissionStarted**

このイベントは、提出が提出され、検証、データベースに保存されると毎回生成されます。これは、提出の一部としてファイルがアップロードされる **前** に行われます。

submissionStarted イベントに渡されるオブジェクトには、以下のパラメーターが含まれています。

```
{
  "submissionId": "<<24-character submission ID>>",
  "submissionStartedTimestamp": "<<2015-02-04T19:18:58.746Z>>"
}
```

```
//NodeJS Events Module. Note, this is required to register event emitter
objects to forms.
var events = require('events');
var submissionEventListener = new events.EventEmitter();

submissionEventListener.on('submissionStarted', function(params){
  var submissionId = params.submissionId;
  var submissionStartedTimestamp = params.submissionStartedTimestamp;
  console.log("Submission with ID " + submissionId + " has started at " +
submissionStartedTimestamp);
});

$fh.forms.registerListener(submissionEventListener, function(err){
  if (err) return handleError(err);

  //submissionEventListener has now been registered with the $fh.forms
Cloud API. Any valid Forms Events will now emit.
});
```

3.14.3. イベント: **submissionComplete**

このイベントは、提出が提出、検証、データベースに保存され、全ファイルがデータベースに保存されて提出が検証されると毎回生成されます。これで提出は、\$fh.forms.getSubmission クラウド API を使って処理する準備が整ったことになります。

```
//NodeJS Events Module. Note, this is required to register event emitter
objects to forms.
var events = require('events');
var submissionEventListener = new events.EventEmitter();

submissionEventListener.on('submissionComplete', function(params){
  var submissionId = params.submissionId;
  var submissionCompletedTimestamp = params.submissionCompletedTimestamp;
  console.log("Submission with ID " + submissionId + " has completed at "
+ submissionCompletedTimestamp);
});

$fh.forms.registerListener(submissionEventListener, function(err){
  if (err) return handleError(err);

  //submissionEventListener has now been registered with the $fh.forms
Cloud API. Any valid Forms Events will now emit.
});
```

```
});
```

イベントに渡される **params** オブジェクトには、以下が含まれています。

```
{
  "submissionId": "<<24-character submission ID>>",
  "submissionCompletedTimestamp": "<<2015-02-04T19:18:58.746Z>>",
  "submission": "<<JSON definition of the Completed Submission.>>"
}
```

3.14.4. イベント: **submissionError**

このイベントは、提出の作成中にエラーが発生すると生成されます。

```
//NodeJS Events Module. Note, this is required to register event emitter
objects to forms.
var events = require('events');
var submissionEventListener = new events.EventEmitter();

submissionEventListener.on('submissionError', function(error){
  console.log("Error Submitting Form");
  console.log("Error Type: ", error.type);
});

$fh.forms.registerListener(submissionEventListener, function(err){
  if (err) return handleError(err);

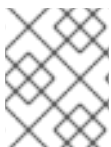
  //submissionEventListener has now been registered with the $fh.forms
  Cloud API. Any valid Forms Events will now emit.
});
```

3.14.4.1. 提出エラーのタイプ

提出エラーは、エラーの理由によって以下の異なるタイプに分けられます。

3.14.4.1.1. 検証エラー

提出されたデータが有効なものではありません。応答は以下の形式になります。



注記

繰り返しフィールドの場合、エラーメッセージはフィールドに入力された値と同じ順序になります。

```
{
  type: 'validationError',
  error: {
    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"
    ]
  }
}
}

```

3.14.4.1.2. 提出の JSON 定義保存での他のエラー

提出の JSON 定義の保存で予期しないエラーがありました。このイベントは、提出の保存時に発生する可能性のある検証 (例えば、提出のデータベースへの保存時に発生するエラー) 以外の全エラーをカバーします。

```

{
  type: 'jsonError',
  error: < Error message >
}

```

3.14.4.1.3. ファイルアップロードのエラー

提出のファイルアップロードでエラーがありました。

```

{
  type: 'fileUploadError',
  submissionId: < ID of the submission related to the file upload >,
  fileName: < The name of the file uploaded >,
  error: < Error message >
}

```

3.15. \$fh.FORMS.DEREGISTERLISTENER



注記

お使いの `package.json` の `fh-mbaas-api` のバージョンは、**4.8.0** 以降である必要があります。

3.15.1. 詳細

\$fh.forms クラウド API からリスナーを削除します。

```

//NodeJS Events Module. Note, this is required to register event emitter
objects to forms.
var events = require('events');
var submissionEventListener = new events.EventEmitter();

$fh.forms.registerListener(submissionEventListener, function(err){

```

```

if (err) return handleError(err);

//submissionEventListener has now been registered with the $fh.forms
Cloud API. Any valid Forms Events will now emit.
submissionEventListener.on('submissionStarted', function(params){
  var submissionId = params.submissionId;
  console.log("Submission with ID " + submissionId + " has started");
});

//Removing the listener from the $fh.forms Cloud API.
$fh.forms.deregisterListener(submissionEventListener);
});

```

3.16. \$FH.FORMS.EXPORTCSV



注記

お使いの `package.json` の `fh-mbaas-api` のバージョンは、**5.10.0** 以降である必要があります。

3.16.1. 詳細

CSV ファイルを `$fh.forms` クラウド API からエクスポートします。エクスポートは、いくつかの CSV ファイルの zip ファイルを返します。フィルターをかけて入力値を使用する際には、以下のフィルターを使用することができます。

- **projectId**: フィルターとなるプロジェクトの GUID。
- **submissionId**: フィルターとなる単一提出 ID またはそれらのアレイ。
- **formId**: フィルターとなる単一フォーム ID またはそれらのアレイ。
- **fieldHeader**: エクスポートされる CSV ファイルで使用するヘッダーテキスト。オプションは、フィールド名に使用する **name** または Form Builder で定義されるフィールドコードを使用する **fieldCode**。

```

// This is the input parameter to filter the list of CSV files.
var queryParams = {
  projectId: "projectId",
  submissionId: "submissionId",
  formId: ["formId1", "formId2"],
  fieldHeader: "name"
};

$fh.forms.exportCSV(queryParams, function(err, fileStreamObject) {
  // fileStreamObject is a zip file containing CSV files associated
  // to the form it was submitted against.
  if (err) return handleError(err);
  /**
   * Pipe the file stream to a writable stream (for example, a FileWriter)
   */
  fileStreamObject.pipe(writable_stream);
  fileStreamObject.resume();
});

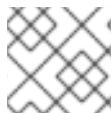
```

3.17. \$fh.FORMS.EXPORTSINGLEPDF



注記

お使いの `package.json` の `fh-mbaas-api` のバージョンは、**5.12.0** 以降である必要があります。



注記

この API は、MBaaS バージョン `>=4.1.0` と互換性があります。

3.17.1. 詳細

`$fh.forms` クラウド API から特定の提出の PDF ファイルをエクスポートします。エクスポートは、ストリームファイルを返します。

```
var params = {
  submissionId: "submissionId"
};

$fh.forms.exportSinglePDF(params, function(err, fileStreamObject){
  if (err) return handleError(err);
  /**
   * Pipe the file stream to a writable stream (for example, a FileWriter)
   */
  fileStreamObject.pipe(writable_stream);
  fileStreamObject.resume();
});
```


第4章 \$FH.HASH

```
$fh.hash(options, callback)
```

特定入力のハッシュ値を生成します。

4.1. 例

```
var options = {
  "algorithm": 'SHA256', // Possible values: MD5 | SHA1 | SHA256 | SHA512
  "text": 'Need more widgets'
};
$fh.hash(options, function (err, result) {
  if (err) {
    return console.error("Failed to generate hash value: " + err);
  } else {
    return console.log("Hash value is :" + result.hashvalue);
  }
});
```

第5章 \$FH.HOST

```
$fh.host(callback);
```

MBaaS のパブリックホスト名をフェッチします。各種の認証ライブラリーでのコールバック URL の設定に便利です。

5.1. 例

```
// Fetch our own host
$fh.host(function (err, host) {
  if (err) return console.error(err);

  console.log('Host: ', host);
});
```

第6章 \$FH.PUSH

```
$fh.push(message, options, callback(err, res))
```

クラウドから登録済みクライアントにプッシュメッセージを送信します。

6.1. 例

関連付けられているプロジェクトの全クライアントアプリにある全デバイスにメッセージをプッシュ

```
var message = {
  alert: "hello from FH"
}, options = {
  broadcast: true
};

$fh.push(message, options,
function (err, res) {
  if (err) {
    console.log(err.toString());
  } else {
    console.log("status : " + res.status);
  }
});
```

特定のクライアントアプリにある特定 **deviceType** のメッセージをプッシュ

```
var message = {
  alert: "hello from FH"
},
options = {
  apps: ["3uzl1ebi6utciy56majgqlj8"], // list of App IDs
  criteria: {
    deviceType: "android"
  }
};

$fh.push(message, options,
function (err, res) {
  if (err) {
    console.log(err.toString());
  } else {
    console.log("status : " + res.status);
  }
});
```

6.2. パラメーター

6.2.1. 通知

- **message** オブジェクト

- **alert** 文字列 — メインのメッセージ
- **sound** 文字列 — (iOS のみ) アプリバンドルのサウンドファイルの名前、もしくは **default**
- **badge** 文字列 — アプリアイコンのバッジとして表示される数
- **userData** オブジェクト — 渡される追加のユーザーデータ

6.2.2. iOS 固有

- **message.apns** オブジェクト — [Apple Push Notification Service](#) に固有のオプション
 - **title** 文字列 — 通知目的を説明した短い文字列
 - **action** 文字列 — アクションボタンのラベル
 - **urlArgs** アレイ — (Safari のみ) `website.json` ファイルの `urlFormatString` 値内にあるプレースホルダーとペアになる値
 - **titleLocKey** 文字列 — (iOS の) 現行ローカライゼーションの `Localizable.strings` ファイル内にあるタイトル文字列へのキー
 - **titleLocArgs** アレイ — (iOS のみ) `title-loc-key` の形式指定子で表示される各種の文字列の値
 - **actionCategory** 文字列 — 対話式通知用のアクションカテゴリーの識別子
 - **contentAvailable** 数値 — (iOS のみ) サイレント通知を送信することで、アプリケーションに新規コンテンツが利用可能になったことを通知。利用可能な値は **1** のみ

6.2.3. Windows 固有

- **message.windows** オブジェクト — Windows プラットフォームに固有のオプション (**Windows Notification Service** および **Microsoft Push Notification Service**)
 - **type** 文字列 — 送信するメッセージのタイプ。利用可能な値は、**toast**、**raw**、**badge** または **tile**。
 - **duration** 文字列 — Toast メッセージの表示期間。利用可能な値は、**long** または **short**。
 - **badge** 文字列 — 数字の代わりに通知バッジとして使用するグリフ。利用可能な値は、**none**、**activity**、**alert**、**available**、**away**、**busy**、**newMessage**、**paused**、**playing**、**unavailable**、**error** または **attention**。バッジタイプの詳細については、[official documentation](#) を参照してください。数値には、**message.badge** パラメーターを使用してください。
 - **tileType** 文字列 — **TileSquareText02** や **TileWideBlockAndText02** などのタイルテンプレート。使用可能な値については、[tile template catalog](#) を参照してください。
 - **images** アレイ — タイルに表示される画像一覧。ローカルファイルのパス (例、**Assets/image.png**) または URL (例、<http://host/image.png>) になります。要素数は、選択した **tileType** で必要とされる画像数と一致する必要があります。

- **textFieldIds** アレイ — タイルに表示されるテキスト一覧。要素数は、選択した **tileType** で必要とされるテキストフィールドの数と一致する必要があります。

6.2.4. 他の設定

- **options** オブジェクト
 - **options.config** オブジェクト
 - **ttl** 番号 — (APNS および GCM のみ) 秒単位での time to live。

6.2.5. プロジェクトでのクライアントアプリの選択



警告

broadcast または **apps** のオプションは手動で設定する必要があり、デフォルト値はありません。

- **options.broadcast** ブール値 — **true** に設定すると、送信しているクラウドアプリを含んでいるプロジェクトにある全クライアントアプリに通知が送信されます。
- **options.apps** アレイ — 通知の送信先となるクライアントアプリの ID 一覧

6.2.6. 受信者のフィルタリング

- **options.criteria** オブジェクト — 通知受信者の選択条件。
詳細は、[通知の送信](#) を参照してください。
 - **alias** アレイ — ユーザー固有の識別子一覧
 - **categories** アレイ — カテゴリー一覧
 - **deviceType** アレイ — デバイスタイプ一覧
 - **variants** アレイ — バリエーション ID の一覧

6.2.7. 応答処理

- **callback(err, res)** 関数 — メッセージが統合プッシュサーバーに送信された後に起動されるコールバック。**err** に設定されると、エラー応答が含まれます。パラメーター **res** には通常のサーバー応答が含まれます。

第7章 \$FH.SEC

```
$fh.sec(options, callback)
```

キーの生成、データの暗号化および暗号解除。

7.1. 例

RSA キー生成

```
$fh.sec({
  "act": 'keygen',
  "params": {
    "algorithm": "RSA", // RSA or AES
    "keysize": 1024 // 1024 or 2048 for RSA
  }
}, function (err, res) {
  if (err) {
    return console.log("RSA key generation failed. Error: " + err);
  }
  return console.log("Public key is " + res.public + " Private key is " +
    res.private + ' Modulu is ' + res.modulu);
});
```

RSA 暗号化

```
$fh.sec({
  "act": 'encrypt',
  "params": {
    "algorithm": "RSA", // padding: PKCS#1
    "plaintext": "Need more starting pages",
    "public": pubkey
  }
}, function (err, result) {
  if (err) {
    return console.log("Encryption failed: " + err);
  }
  return console.log("Encrypted data is " + result.ciphertext);
});
```

RSA 暗号解除

```
$fh.sec({
  "act": 'decrypt',
  "params": {
    "algorithm": "RSA",
    "ciphertext": "23941A28432482E374102FF48723BCB9847324",
    "private": privatekey
  }
}, function (err, result) {
  if (err) {
    return console.log("Decryption failed: " + err);
  }
});
```

```

    }
    return console.log("Decryption data is " + result.plaintext);
  });

```

AES キー生成

```

$fh.sec({
  "act": 'keygen',
  "params": {
    "algorithm": "AES", // AES or RSA
    "keysize": 128 // 128 or 256 for AES
  }
}, function (err, res) {
  if (err) {
    return console.log("AES key generation failed. Error: " + err);
  }
  return console.log("AES secret key is " + res.secretkey + "
Initialisation Vector is " + res.iv);
});

```

AES 暗号化

```

$fh.sec({
  "act": 'encrypt',
  "params": {
    "algorithm": "AES", // mode : CBC, padding: PKCS#5
    "plaintext": "Need more starting pages",
    "key": secretkey,
    "iv": iv
  }
}, function (err, result) {
  if (err) {
    return console.log("Encryption failed: " + err);
  }
  return console.log("Encrypted data is " + result.ciphertext);
});

```

AES 暗号解除

```

$fh.sec({
  "act": 'decrypt',
  "params": {
    "algorithm": "AES",
    "ciphertext": "23941A28432482E374102FF48723BCB9847324",
    "key": secretkey,
    "iv": iv
  }
}, function (err, result) {
  if (err) {
    return console.log("Decryption failed: " + err);
  }
  return console.log("Decryption data is " + result.plaintext);
});

```

第8章 \$FH.SERVICE

```
$fh.service(options, callback);
```

MBaaS サービス のエンドポイントを呼び出します。

最低要件

- `fh-mbaas-api` : v4.9.1

8.1. 例

```
var $fh = require('fh-mbaas-api');

$fh.service({
  "guid" : "0123456789abcdef01234567", // The 24 character unique id of
the service
  "path": "/hello", //the path part of the url excluding the hostname -
this will be added automatically
  "method": "POST", //all other HTTP methods are supported as well. for
example, HEAD, DELETE, OPTIONS
  "params": {
    "hello": "world"
  }, //data to send to the server - same format for GET or POST
  "timeout": 25000, // timeout value specified in milliseconds. Default:
60000 (60s)
  "headers" : {
    // Custom headers to add to the request. These will be appended to the
default headers.
  }
}, function(err, body, res) {
  console.log('statusCode: ', res && res.statusCode);
  if ( err ) {
    // An error occurred during the call to the service. log some
debugging information
    console.log('service call failed - err : ', err);
  } else {
    console.log('Got response from service - status body : ',
res.statusCode, body);
  }
});
```


第9章 \$FH.STATS

一時統計カウンターとタイマーを利用します。これらは Studio でグラフとして表示できます。

9.1. 例

```
// Increment a counter.  
// The name for the counter you want to increment.  
// If this doesn't exist, it is created, and starts at 0.  
var counter = 'my_counter';  
$fh.stats.inc(counter);  
  
// Decrement a counter  
$fh.stats.dec(counter);  
  
// Record a timer value  
// The name for the timer you want to record a value for.  
// If it doesn't exist, it is created.  
var timer_name = 'my_timer';  
// Timing in milliseconds of the interval you wish to record  
// (for example, time difference between a timer start and end)  
var time_in_ms = 500;  
$fh.stats.timing(timer_name, time_in_ms);
```

第10章 \$FH.SYNC

クラウド同期フレームワークでは、バックエンドデータへのアクセスを提供し、データ競合を管理するハンドラー関数を定義する必要があります。これらは、`handleXXX()` 関数で指定されます。管理される一意のデータセットはそれぞれ、`dataset_id` で識別されます。これは、同期フレームワークのどの関数を呼び出す際にも、最初のパラメーターとして指定される必要があります。



注記

ハンドラー関数のデフォルト実装は、MBaaS サービスの一部として既に提供されています。これらはホストされているデータベースサービスを指定して、(`$fh.db` で) データを保存します。使用するアプリにおいてこれで十分な場合は、さらにハンドラー関数を実装する必要はなく、`init` 関数を呼び出して `sync` オプションを提供してください。

ただし、デフォルト実装がアプリ要件を満たさない場合 (データを他の場所に保存する必要がある場合など) は、以下に挙げるハンドラー関数の実装を提供する必要があります。CRUDL (作成、読み込み、更新、削除、一覧表示) 操作すべてに実装を提供する、または対応するハンドラー関数の上書きを提供するだけで特定操作のデフォルト動作を変更することもできます。例えば、上書きの `$fh.sync.handleRead` 関数を提供して独自のデータ論理読み込みを実装し、その他の操作ではデフォルトの MBaaS 実装を使用するようにすることが可能です。

以下の表では、本リリースにおける `fh-mbaas-api` バージョン 7 の導入に関連した API の変更を掲載しています。7.0.0 以前のバージョンも引き続き使用可能で、そのドキュメントも利用可能になっています。

API	7.0.0 以降	7.0.0 より前
<code>\$fh.sync.setConfig</code>	新規	該当なし
<code>\$fh.sync.connect</code>	新規	該当なし
<code>\$fh.sync.init</code>	変更なし	以前のドキュメント
<code>\$fh.sync.invoke</code>	変更なし	以前のドキュメント
<code>\$fh.sync.stop</code>	変更なし	以前のドキュメント
<code>\$fh.sync.stopAll</code>	変更なし	以前のドキュメント
<code>\$fh.sync.handleList</code>	変更あり	以前のドキュメント
<code>\$fh.sync.globalHandleList</code>	変更あり	以前のドキュメント
<code>\$fh.sync.handleCreate</code>	変更あり	以前のドキュメント
<code>\$fh.sync.globalHandleCreate</code>	変更あり	以前のドキュメント
<code>\$fh.sync.handleRead</code>	変更あり	以前のドキュメント

API	7.0.0 以降	7.0.0 より前
\$fh.sync.globalHandleRead	変更あり	以前のドキュメント
\$fh.sync.handleUpdate	変更あり	以前のドキュメント
\$fh.sync.globalHandleUpdate	変更あり	以前のドキュメント
\$fh.sync.handleDelete	変更あり	以前のドキュメント
\$fh.sync.globalHandleDelete	変更あり	以前のドキュメント
\$fh.sync.handleCollision	変更なし	以前のドキュメント
\$fh.sync.globalHandleCollision	変更なし	以前のドキュメント
\$fh.sync.listCollisions	変更あり	以前のドキュメント
\$fh.sync.globalListCollisions	変更あり	以前のドキュメント
\$fh.sync.removeCollision	変更あり	以前のドキュメント
\$fh.sync.globalRemoveCollision	変更あり	以前のドキュメント
\$fh.sync.setGlobalHashFn	新規	該当なし
\$fh.sync.setRecordHashFn	新規	該当なし
\$fh.sync.interceptRequest	変更あり	以前のドキュメント
\$fh.sync.globalInterceptRequest	変更あり	以前のドキュメント
\$fh.sync.getEventEmitter	新規	該当なし

10.1. \$FH.SYNC.SETCONFIG

同期サーバーを設定します。

同期サーバーの準備ができた後、データセットの初期化前にこの API を呼び出します。**sync.init** への呼び出しが明示的になければ、データセットの初期化は通常、最初の同期クライアントの接続時に行われます。いずれの場合でも、**setConfig** を呼び出す妥当なタイミングは、**sync:ready** イベントがトリガーする時になります。

10.1.1. 使用法

```
$fh.sync.setConfig(options)
```

10.1.2. パラメーター

10.1.2.1. options

- 説明: 同期サーバーの設定オプション。
- タイプ: Object
- サポートされるキー
 - **pendingWorkerInterval**
 - 説明: 保留中ワーカーの間隔をミリ秒単位で設定します。ワーカーの間隔に関する詳細は、[データ同期設定ガイド](#) を参照してください。
 - タイプ: 数字
 - デフォルト値: 1
 - **pendingWorkerBackoff**
 - 説明: 保留中ワーカーのバックオフ戦略を指定します。ワーカーのバックオフに関する詳細は、[データ同期設定ガイド](#) を参照してください。
 - タイプ: Object
 - デフォルト値: **{strategy: 'exp', max: 60000}**
 - strategy
バックオフ戦略を定義します。有効な値は **exp** (exponential: 指数) と **fib** (fibonacci) です。これ以外の値を設定すると、ワーカーバックオフは無効になります。
 - max
バックオフ戦略の最大遅延時間 (ミリ秒単位)。
 - **pendingWorkerConcurrency**
 - 説明: 同時実行の保留中ワーカーの数を設定します。0 に設定すると保留中ワーカーは無効になります。
 - タイプ: 数字
 - デフォルト値: 1
 - **ackWorkerInterval**
 - 説明: ack ワーカーの間隔をミリ秒単位で設定します。ワーカーの間隔に関する詳細は、[データ同期設定ガイド](#) を参照してください。
 - タイプ: 数字
 - デフォルト値: 1
 - **ackWorkerBackoff**
 - 説明: ack ワーカーのバックオフ戦略を指定します。ワーカーのバックオフに関する詳細は、[データ同期設定ガイド](#) を参照してください。

- タイプ: Object
- デフォルト値: **{strategy: 'exp', max: 60000}**
 - strategy
バックオフ戦略を定義します。有効な値は **exp** (exponential: 指数) と **fib** (fibonacci) です。これ以外の値を設定すると、ack ワーカーのバックオフ戦略は無効になります。
 - max
バックオフ戦略の最大遅延時間 (ミリ秒単位)。
- **ackWorkerConcurrency**
 - 説明: 同時実行の ack ワーカーの数を設定します。0 に設定すると ack ワーカーは無効になります。
 - タイプ: 数字
 - デフォルト値: 1
- **syncWorkerInterval**
 - 説明: 同期ワーカーの間隔をミリ秒単位で設定します。ワーカーの間隔に関する詳細は、[データ同期設定ガイド](#) を参照してください。
 - タイプ: 数字
 - デフォルト値: 100
- **syncWorkerBackoff**
 - 説明: 同期ワーカーのバックオフ戦略を指定します。ワーカーのバックオフに関する詳細は、[データ同期設定ガイド](#) を参照してください。
 - タイプ: Object
 - デフォルト値: **{strategy: 'none'}**
 - strategy
バックオフ戦略を定義します。有効な値は **exp** (exponential: 指数) と **fib** (fibonacci) です。これ以外の値を設定すると、バックオフ戦略は無効になります。
 - max
バックオフ戦略の最大遅延時間 (ミリ秒単位)。
- **syncWorkerConcurrency**
 - 説明: 同時実行の同期ワーカーの数を設定します。0 に設定すると同期ワーカーは無効になります。
 - タイプ: 数字
 - デフォルト値: 1
- **collectStats**
 - 説明: 同期サーバーがパフォーマンス数値データを収集するかどうかを決定します。

デフォルトでは、同期サーバーは db 操作のタイミングや API のタイミング、キューのサイズなど、いくつかのパフォーマンスデータを収集します。

数値データは Redis に保存され、パフォーマンスオーバーヘッドは最小限に抑えられます。

- タイプ: ブール値
- デフォルト値: true
- **statsRecordsToKeep**
 - 説明: Redis 内の各メトリックシリーズで保存される数値データのポイント数を決定します。
 - タイプ: 数字
 - デフォルト値: 1000
- **collectStatsInterval**
 - 説明: 数値を収集する頻度をミリ秒単位で決定します。
 - タイプ: 数字
 - デフォルト値: 5000
- **metricsInfluxdbHost**
 - 説明: 同期サーバーがパフォーマンスデータを送信する InfluxDB ホストを指定します。
 - タイプ: 文字列
 - デフォルト値: null
- **metricsInfluxdbPort**
 - 説明: InfluxDB ポートを指定します。UDP ポートにする必要があります。
 - タイプ: 数字
 - デフォルト値: null
- **queueMessagesTTL**
 - 説明: 削除のマークが付けられたキューメッセージの有効時間 (TTL) を設定します。TTL が過ぎるとメッセージがデータベースから削除されます。(秒単位)
 - タイプ: 数字
 - デフォルト値: 86400 (1 日)
- **datasetClientCleanerRetentionPeriod**
 - 説明: 非アクティブな datasetClient が削除されるまでデータベースに保存される期間を指定します。
datasetClients の不要な削除をさけるために、アプリに適切な値でこの設定を上書きすることを検討してください。例えば、週末にこのアプリを使用するユーザーがいない場合は、この値を '96h' に設定できます。

- タイプ: 文字列
- デフォルト値: '24h'
 - サポートされる単位は以下の通りです: **s** (秒)、**m** (分)、**h** (時間)、**d** (日)、**w** (週)、**y** (年)
- **datasetClientCleanerCheckFrequency**
 - 説明: datasetClient のクリーナージョブの実行頻度を指定します。
 - タイプ: 文字列
 - デフォルト値: '1h'
- **useCache**
 - 説明: データセットクライアントのレコードをキャッシュする際に Redis を使用するかどうかを指定します。
これを有効にするとデータベース上のリクエスト数が低減され、**syncRecords** API 呼び出しの時間が短縮されます。

これは実験的な機能で、変更が全クライアントで表示されるまでに時間がかかる場合があります。
 - タイプ: ブール値
 - デフォルト値: false
- **schedulerInterval**
 - 説明: 同期スケジューラーの間隔をミリ秒単位で設定します。
通常はデフォルト値を変更する必要はありません。
 - タイプ: 数字
 - デフォルト値: 500
- **schedulerLockName**
 - 説明: 常に 1 つの同期スケジューラーのみが実行可能であることを担保するためにロックが使用されます。このフィールドではロックの名前を決定します。
通常はデフォルト値を変更する必要はありません。
 - タイプ: 文字列
 - デフォルト値: **locks:sync:SyncScheduler**
- **schedulerLockMaxTime**
 - 説明: 常に 1 つの同期スケジューラーのみが実行可能であることを担保するためにロックが使用されます。このフィールドでは、同期スケジューラーがロックを保持できる最大時間を決定します。これは、同期スケジューラーがロックを永遠に保持しないようにするためのものです (例: プロセスがクラッシュした場合など)。
通常はデフォルト値を変更する必要はありません。
 - タイプ: 数字

- デフォルト値: 20000

○ datasetClientUpdateConcurrency

- 説明: 同期サーバーに対して多くの同時実行の同期リクエストがある場合、データセットクライアントに多くの更新操作が生成されます。データベースの過剰負荷をさけるために、これらの操作はキューに入れられ、このオプションの同時実行で遂行されます。通常はデフォルト値を変更する必要はありません。
- タイプ: 数字
- デフォルト値: 10

10.1.3. 例

```
$fh.events.on('sync:ready', function(){
  var pendingWorkerInterval = process.env.PENDING_WORKER_INTERVAL || 500;
  var syncWorkerInterval = process.env.SYNC_WORKER_INTERVAL || 500;
  var ackWorkerInterval = process.env.ACK_WORKER_INTERVAL || 500;
  var useCache = process.env.USE_CACHE === 'true';
  var syncConfig = {
    pendingWorkerInterval: parseInt(pendingWorkerInterval),
    ackWorkerInterval: parseInt(ackWorkerInterval),
    syncWorkerInterval: parseInt(syncWorkerInterval),
    collectStatsInterval: 4000,
    metricsInfluxdbHost: process.env.METRICS_HOST,
    metricsInfluxdbPort: parseInt(process.env.METRICS_PORT),
    useCache: useCache
  };
  $fh.sync.setConfig(syncConfig);
});
```

10.2. \$FH.SYNC.CONNECT

同期サーバーが MongoDB および Redis といった必須リソースに接続されるようにします。



注記

fh-mbaas-api から同期サーバーを使用する際は、**sync.connect** が自動的に呼び出され、クラウドアプリのデータベースを使用します。このシナリオでは、**sync.connect** を呼び出す必要はありません。

10.2.1. 使用法

```
$fh.sync.connect(mongodbConnectionString, mongodbConf,
  redisConnectionString, callback)
```

10.2.2. パラメーター

10.2.2.1. mongodbConnectionString

- 説明: MongoDB の接続 url。この url はすぐに **mongodb** モジュールに渡されます。

- タイプ: 文字列

10.2.2.2. mongodbConf

- 説明: MongoDB の接続オプション。このオプションに関する詳細は、[MongoClient.connect API doc](#) を参照してください。

通常は、この API を呼び出す必要はありませんが、環境変数を使用することで MongoDB 接続オプションのいくつかを制御することができます。

- SYNC_MONGODB_POOLSIZE
 - 説明: MongoDB 接続のプールサイズを指定
 - タイプ: 数字
 - デフォルト値: 50
- タイプ: Object

10.2.2.3. redisConnectionString

- 説明: Redis の接続 URL。必要な場合は、これに認証情報を含めることができます。この URL は **redis** モジュールに直接渡されます。
- タイプ: 文字列

10.2.2.4. コールバック

- 説明: コールバック関数
- タイプ: 関数

10.2.3. 例

```
var mongodbUrl = "mongodb://mongouser:mongopass@127.0.0.1";
var redisUrl = "redis://redisuser:redisspass@127.0.0.1";

$fh.sync.connect(mongodbUrl, {}, redisUrl, function(err){
  if (err) {
    console.error('Connection error for sync', err);
  } else {
    console.log('sync connected');
  }
});
```

10.3. \$FH.SYNC.INIT

注記: RHMAP 3.19 には fh-mbaas-api バージョン 7 が含まれています。ご使用の package.json ファイルにある fh-mbaas-api のバージョンが 7.0.0 より前の場合は、[以前のバージョンのセクション](#) を参照してください。

データセットの同期オプションを設定し、同期プロセスを開始します。この API は、データセット上での同期プロセスの開始にも使用できます。

この API 呼び出しはオプションですが、特定のデータセットのオプションを上書きすることができます。

10.3.1. 使用法

```
$fh.sync.init(dataset_id, options, callback)
```

10.3.2. パラメーター

10.3.2.1. dataset_id

- 説明: 同期する必要があるデータセットの ID。
- タイプ: 文字列

10.3.2.2. options

- 説明: データセットの同期オプション。
- タイプ: Object
- サポートされるキー:
 - **syncFrequency**
 - 説明: 同期操作の間で同期サーバーが待機する時間を設定します (秒単位)。このオプションの値を決定するには、[Configuring Sync Frequency](#) および [Sync Server Performance and Scaling](#) を確認してください。
 - タイプ: 数字
 - デフォルト値: 30
 - **clientSyncTimeout**
 - 説明: アクティビティがないためにデータセットクライアントが "inactive" とマークされるまでの時間を指定します (秒単位)。データセットクライアントが "inactive" とマークされると、同期サーバーはこのデータセットクライアントとバックエンドとの同期を停止します。

ユーザーが後でこの "inactive" なデータセットクライアントにアクセスすると、次の同期操作が完了するまで、ユーザーは待機する必要がある可能性があります。このシナリオを避けるには、このタイムアウトを比較的長い時間に設定します。
 - タイプ: 数字
 - デフォルト値: 3600 (1 時間)
 - **backendListTimeout**
 - 説明: 同期サーバーがバックエンドデータベースとの同期操作の完了を待機するタイムアウト値を指定します (秒単位)。
 - タイプ: 数字
 - デフォルト値: 300

10.3.2.3. コールバック

- 説明: `fh.init` が完了した際に起動するコールバック関数。
- タイプ: 関数

10.3.3. 例

```
var datasetId = "todolist";
var syncOptions = {
  syncFrequency: 10,
  clientSyncTimeout: 24*60*60
};
$fh.sync.init(datasetId, syncOptions, function(err){
  if (err) {
    console.error('sync init failed due to error', err);
  } else {
    console.log('sync init finished successfully');
  }
});
```

10.3.4. \$fh.sync.init (バージョン 7.0.0 以前の fh-mbaas-api 向け)

```
$fh.sync.init(dataset_id, options, callback)
```

指定されたデータセットのクラウドデータ同期サービスを初期化します。

10.3.4.1. 例

```
// Unique Id for the dataset to initialise.
var dataset_id = 'tasks';
// Configuration options
var options = {
  "syncFrequency": 10, // How often to synchronise data with the back end
  data store in seconds. Default: 10s
  "logLevel": "info" // The level of logging. Can be useful for debugging.
  Valid options including: 'silly', 'verbose', 'info', 'warn', 'debug',
  'error'
};
$fh.sync.init(dataset_id, options, function(err) {
  // Check for any error thrown during initialisation
  if (err) {
    console.error(err);
  } else {
    //you can now provide data handler function overrides (again, not
    required at all). For example,
    $fh.sync.handleList(dataset_id, function(dataset_id, params, cb,
    meta_data){
      //implement the data listing logic
    });
  }
});
```

10.4. \$FH.SYNC.INVOKE

注記: RHMAP 3.19 には fh-mbaas-api バージョン 7 が含まれています。ご使用の package.json ファイルにある fh-mbaas-api のバージョンが 7.0.0 より前の場合は、[以前のバージョンのセクション](#) を参照してください。

同期サーバーの内部同期関数を開始します。通常の開発ではこれらの関数を開始する必要はありません。この API は 同期クライアントから呼び出されるので、**params** オブジェクトは同期クライアントからのリクエストのボディと同じであることに留意してください。

10.4.1. 使用法

```
$fh.sync.invoke(dataset_id, params, callback)
```

10.4.2. パラメーター

10.4.2.1. dataset_id

- 説明: データセットの ID。
- タイプ: 文字列

10.4.2.2. params

- 説明: ターゲットの関数名を指定します。ターゲット関数に渡されます。
- タイプ: Object
- サポートされるキー:
 - **fn**
 - 説明: 開始するターゲットの関数名を指定します。以下の関数がサポートされています。
 - **sync**
 - **syncRecords**
 - **listCollisions**
 - **removeCollision**
 - タイプ: 文字列
 - 他のフィールドはターゲット関数によって異なります。必須フィールドのソースコードを確認してください。

10.4.3. 例

```
var datasetId = "todo";
var syncRecordsParams = {
  fn: 'syncRecords',
  dataset_id: datasetId,
  query_params: {},
}
```

```

    clientRecs: {},
    __fh: {
      cuid: 'testdeviceid'
    }
  };

  $fh.sync.invoke(datasetId, syncRecordsParams, function(err, syncData) {
    if (err) {
      console.error('Failed to call syncRecords due to error', err);
    } else {
      console.log('Got sync data', syncData);
    }
  });

```

10.4.4. \$fh.sync.invoke (バージョン 7.0.0 以前の fh-mbaas-api 向け)

```
$fh.sync.invoke(dataset_id, params, callback)
```

同期サーバーを起動します。

10.4.4.1. 例

```

// This should be called from a cloud "act" function.
// The params passed to the "act" function:
var params = {
  "limit": 50
};
$fh.sync.invoke(dataset_id, params, function() {
  // "act" function callback
});

```

10.5. \$FH.SYNC.STOP

注記: RHMAP 3.19 には fh-mbaas-api バージョン 7 が含まれています。ご使用の package.json ファイルにある fh-mbaas-api のバージョンが 7.0.0 より前の場合は、[以前のバージョンのセクション](#) を参照してください。

特定のデータセットとデータセットバックエンドとの同期を停止します。

10.5.1. 使用法

```
$fh.sync.stop(dataset_id, callback)
```

10.5.2. パラメーター

10.5.2.1. dataset_id

- 説明: データセットの ID。
- タイプ: 文字列

10.5.2.2. コールバック

- 説明: コールバック関数
- タイプ: 関数

10.5.3. 例

```
var datasetId = "todolist";
$fh.sync.stop(datasetId, function(err){
  if (err) {
    console.error('sync stop failed due to error', err);
  } else {
    console.log('sync stop finished successfully');
  }
});
```

10.5.4. \$fh.sync.stop (バージョン 7.0.0 以前の fh-mbaas-api 向け)

```
$fh.sync.stop(dataset_id, callback)
```

指定された dataset_id のクラウドデータ同期を停止します。

10.5.4.1. 例

```
// This will remove any reference to the dataset from the sync service.
// Any subsequent cloud calls to sync.invoke will fail with an
unknown_dataset error.
// The dataset can be put back under control of the sync service by
calling the
// sync.init() function again.
// Calling stop on a non-existent dataset has no effect.
// Calling stop multiple times on the same dataset has no effect.
$fh.sync.stop(dataset_id, function() {
  // Callback to invoke once the dataset has been removed from the
management
  // of the service.
  // There are no parameters passed to this callback.
});
```

10.6. \$FH.SYNC.STOPALL

注記: RHMAP 3.19 には fh-mbaas-api バージョン 7 が含まれています。ご使用の package.json ファイルにある fh-mbaas-api のバージョンが 7.0.0 より前の場合は、[以前のバージョンのセクション](#) を参照してください。

すべてのデータセットとデータセットバックエンドとの同期を停止します。

10.6.1. 使用法

```
$fh.sync.stopAll(callback)
```

10.6.2. パラメーター

10.6.2.1. コールバック

- 説明: コールバック関数
- タイプ: 関数

10.6.3. 例

```
$fh.sync.stopAll(function(err){
  if (err) {
    console.error('sync stopAll failed due to error', err);
  } else {
    console.log('sync stopAll finished successfully');
  }
});
```

10.6.4. \$fh.sync.stopAll (バージョン 7.0.0 以前の fh-mbaas-api 向け)

```
$fh.sync.stopAll(callback)
```

全データセットのクラウドデータ同期サービスを停止します。

10.6.4.1. 例

```
// This will remove all reference to all datasets from the sync service.
// Any subsequent cloud calls to sync.invoke() will fail with an
unknown_dataset error.
// Any of the datasets can be put back under control of the sync service
by calling
// the sync.init() function again and passing the required dataset_id.
// Calling stop multiple times has no effect -
// except that the return data to the callback (an array of dataset_ids
which are no longer being synced) will be different.
$fh.sync.stopAll(function(err, res) {
  if (err) console.error(err); // Any error thrown during the removal of
the datasets

  // A JSON Array of Strings - representing the dataset_Ids which have
been
  // removed from the sync service.
  console.log(res);
});
```

10.7. \$FH.SYNC.HANDLELIST

注記: RHMAP 3.19 には fh-mbaas-api バージョン 7 が含まれています。ご使用の package.json ファイルにある fh-mbaas-api のバージョンが 7.0.0 より前の場合は、[以前のバージョンのセクション](#) を参照してください。

特定のデータセットのデータをデータセットバックエンドから一覧表示するハンドラー関数を定義します。



注記

返されたデータが有効な [JSON objects](#) であることを確認してください。例えば、データオブジェクトは JSON Object 内では有効ではありません。



注記

返されたデータ (またはネスト化されたオブジェクト) に **_id** フィールドがないことを確認してください。MongoDB は、**_id** フィールドが含まれているデータを保存しません。

10.7.1. 使用法

```
$fh.sync.handleList(dataset_id, function listHandler(dataset_id,  
query_params, meta_data, cb){});
```

10.7.2. パラメーター

10.7.2.1. dataset_id

- 説明: データセットの ID。
- タイプ: 文字列

10.7.2.2. listHandler

- 説明: レコードを一覧表示する際に呼び出される関数。
- タイプ: 関数
- 呼び出されるパラメーター:
 - **dataset_id**
 - 説明: データセットの ID。
 - タイプ: 文字列
 - **query_params**
 - 説明: リストの一部として使用するクエリパラメーター。null とすることも可能。
 - タイプ: Object
 - **meta_data**
 - 説明: データに関連付けるメタデータ。null とすることも可能。
 - タイプ: Object
 - **cb**

- 説明: コールバック関数
- タイプ: 関数

10.7.3. 例

```
$fh.sync.handleList("todo", function(dataset_id, params, meta_data, cb) {
  // The dataset identifier that this function was defined for
  console.log(dataset_id);

  // JSON object representing query parameters passed from the client.
  // These can be used to restrict the data set returned.
  console.log(params);

  // The callback into the sync service to store the dataset
  // cb(err, data)
  cb(null, { // A JSON Object - representing the data
    uid_1 : { /* data */ },
    uid_2 : { /* data */ },
    uid_3 : { /* data */ }
  });
});

// It is recommended that the handleList function converts data from the
// back end
// format into a full JSON Object.
// This is a sensible approach when reading data from relational and
// nonrelational
// databases, and works well for SOAP and XML data.
// However, it may not always be feasible - for example, when reading non
// structured data.
// In these cases, the recomened approach is to create a JSON object with
// a single
// key called "data" and set the value for this key to be the actual data.
// for example, xml data
/*
<dataset>
  <row>
    <userid>123456</userid>
    <firstname>Joe</firstname>
    <surname>Bloggs</surname>
    <dob>1970-01-01</dob>
    <gender>male</gender>
  </row>
</dataset>
*/
/* json data
{
  "123456" : {
    "userid" : "123456",
    "firstname" : "Joe",
    "surname" : "Bloggs",
    "dob" : "1970-01-01",
    "gender" : "male"
  }
}
*/
```

```

*/

// And for non structured data:
/*
123456|Joe|Bloggs|1970-01-01|male

{
  "123456" : {
    "data" : "123456|Joe|Bloggs|1970-01-01|male"
  }
}
*/

```

10.7.4. \$fh.sync.handleList (バージョン 7.0.0 以前の fh-mbaas-api 向け)

```
$fh.sync.handleList(dataset_id, callback)
```

データセットのデータをバックエンドデータソースから一覧表示するハンドラー関数を定義します。データセットが初期化された後にこの API を呼び出してください。

10.7.4.1. 例

```

$fh.sync.handleList(dataset_id, function(dataset_id, params, cb,
meta_data) {
  // The dataset identifier that this function was defined for
  console.log(dataset_id);

  // JSON object representing query parameters passed from the client.
  // These can be used to restrict the data set returned.
  console.log(params);

  // The callback into the sync service to store the dataset
  // cb(err, data)
  cb(null, { // A JSON Object - representing the data
    uid_1 : { /* data */ },
    uid_2 : { /* data */ },
    uid_3 : { /* data */ }
  });
});
// It is recommended that the handleList function converts data from the
back end
// format into a full JSON Object.
// This is a sensible approach when reading data from relational and
nonrelational
// databases, and works well for SOAP and XML data.
// However, it may not always be feasible - for example, when reading non
structured data.
// In these cases, the recomened approach is to create a JSON object with
a single
// key called "data" and set the value for this key to be the actual data.
// for example, xml data
/*
<dataset>
  <row>
    <userid>123456</userid>

```

```

    <firstname>Joe</firstname>
    <surname>Bloggs</surname>
    <dob>1970-01-01</dob>
    <gender>male</gender>
  </row>
</dataset>
*/
/* json data
{
  "123456" : {
    "userid" : "123456",
    "firstname" : "Joe",
    "surname" : "Bloggs",
    "dob" : "1970-01-01",
    "gender" : "male"
  }
}
*/

// And for non structured data:
/*
123456|Joe|Bloggs|1970-01-01|male

{
  "123456" : {
    "data" : "123456|Joe|Bloggs|1970-01-01|male"
  }
}
*/

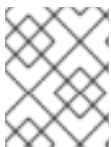
```

10.8. \$FH.SYNC.GLOBALHANDLELIST

注記: RHMAP 3.19 には fh-mbaas-api バージョン 7 が含まれています。ご使用の package.json ファイルにある fh-mbaas-api のバージョンが 7.0.0 より前の場合は、[以前のバージョンのセクション](#) を参照してください。

データセットバックエンドからのデータを一覧表示するグローバルのハンドラー関数を定義します。

これは複数のデータセットで使用可能ですが、通常は [handleList API](#) を使用するデータセットにハンドラーが割り当てられていない場合にのみ使用されます。



注記

返されたデータが有効な [JSON objects](#) であることを確認してください。例えば、データオブジェクトは JSON Object 内では有効ではありません。



注記

返されたデータ (またはネスト化されたオブジェクト) に `_id` フィールドがないことを確認してください。MongoDB は、`_id` フィールドが含まれているデータを保存します。

10.8.1. 使用法

```
$fh.sync.globalHandleList(function listHandler(dataset_id, query_params, meta_data, cb){});
```

10.8.2. パラメーター

10.8.2.1. listHandler

- 説明: レコードを一覧表示する際に呼び出される関数。
- タイプ: 関数
- 呼び出されるパラメーター:
 - **dataset_id**
 - 説明: データセットの ID。
 - タイプ: 文字列
 - **query_params**
 - 説明: リストの一部として使用するクエリパラメーター。null とすることも可能。
 - タイプ: Object
 - **meta_data**
 - 説明: データに関連付けるメタデータ。null とすることも可能。
 - タイプ: Object
 - **cb**
 - 説明: コールバック関数
 - タイプ: 関数

10.8.3. 例

```
$fh.sync.globalHandleList(function(dataset_id, params, meta_data, cb) {  
  // The dataset identifier that this function was defined for  
  console.log(dataset_id);  
  
  // JSON object representing query parameters passed from the client.  
  // These can be used to restrict the data set returned.  
  console.log(params);  
  
  // The callback into the sync service to store the dataset  
  // cb(err, data)  
  cb(null, { // A JSON Object - representing the data  
    uid_1 : { /* data */ },  
    uid_2 : { /* data */ },  
    uid_3 : { /* data */ }  
  });  
});
```

```

// It is recommended that the handleList function converts data from the
// back end
// format into a full JSON Object.
// This is a sensible approach when reading data from relational and
// nonrelational
// databases, and works well for SOAP and XML data.
// However, it may not always be feasible - for example, when reading non
// structured data.
// In these cases, the recomened approach is to create a JSON object with
// a single
// key called "data" and set the value for this key to be the actual data.
// for example, xml data
/*
<dataset>
  <row>
    <userid>123456</userid>
    <firstname>Joe</firstname>
    <surname>Bloggs</surname>
    <dob>1970-01-01</dob>
    <gender>male</gender>
  </row>
</dataset>
*/
/* json data
{
  "123456" : {
    "userid" : "123456",
    "firstname" : "Joe",
    "surname" : "Bloggs",
    "dob" : "1970-01-01",
    "gender" : "male"
  }
}
*/

// And for non structured data:
/*
123456|Joe|Bloggs|1970-01-01|male

{
  "123456" : {
    "data" : "123456|Joe|Bloggs|1970-01-01|male"
  }
}
*/

```

10.8.4. \$fh.sync.globalHandleList (バージョン 7.0.0 以前の fh-mbaas-api 向け)

```
$fh.sync.globalHandleList(callback)
```

\$fh.sync.handleList と似ていますが、ハンドラーをグローバルに設定することで複数のデータセットが同一のハンドラー関数を使用できるようになります。グローバルハンドラーは、\$fh.sync.handleList でデータセットにハンドラーが割り当てられていない場合にのみ使用されます。

10.8.4.1. 例

```
$fh.sync.globalHandleList(function(dataset_id, params, cb, meta_data){  
  //list data for the specified dataset_id  
});
```

10.9. \$FH.SYNC.HANDLECREATE

注記: RHMAP 3.19 には fh-mbaas-api バージョン 7 が含まれています。ご使用の package.json ファイルにある fh-mbaas-api のバージョンが 7.0.0 より前の場合は、[以前のバージョンのセクション](#) を参照してください。

特定のデータセットでデータセットバックエンドに単一のデータ行を作成するハンドラー関数を定義します。

10.9.1. 使用法

```
$fh.sync.handleCreate(dataset_id, function createHandler(dataset_id, data,  
  meta_data, cb){});
```

10.9.2. パラメーター

10.9.2.1. dataset_id

- 説明: データセットの ID。
- タイプ: 文字列

10.9.2.2. createHandler

- 説明: レコードを作成する際に呼び出される関数。
- タイプ: 関数
- 呼び出されるパラメーター:
 - **dataset_id**
 - 説明: データセットの ID。
 - タイプ: 文字列
 - **data**
 - 説明: 作成するデータ
 - タイプ: Object
 - **meta_data**
 - 説明: データに関連付けるメタデータ。null とすることも可能。
 - タイプ: Object
 - **cb**

- 説明: コールバック関数
- タイプ: 関数

10.9.3. 例

```
$fh.sync.handleCreate("todo", function(dataset_id, data, meta_data, cb){
  // The dataset identifier that this function was defined for
  console.log(dataset_id);

  // Row of data to create
  console.log(data);

  // Sample back-end storage call
  var savedData = saveData(data);
  var res = {
    "uid": savedData.uid, // Unique Identifier for row
    "data": savedData.data // The created data record - including any
    system or UID fields added during the create process
  };

  // Callback function for when the data has been created, or if theres an
  error
  return cb(null, res);
});
```

10.9.4. \$fh.sync.handleCreate (バージョン 7.0.0 以前の fh-mbaas-api 向け)

```
$fh.sync.handleCreate(dataset_id, callback)
```

バックエンドで単一行のデータを作成するハンドラー関数を定義します。データセットが初期化された後に呼び出してください。

10.9.4.1. 例

```
// data source for a dataset.
$fh.sync.handleCreate(dataset_id, function(dataset_id, data, cb,
meta_data) {
  // The dataset identifier that this function was defined for
  console.log(dataset_id);

  // Row of data to create
  console.log(data);

  // Sample back-end storage call
  var savedData = saveData(data);
  var res = {
    "uid": savedData.uid, // Unique Identifier for row
    "data": savedData.data // The created data record - including any
    system or UID fields added during the create process
  };

  // Callback function for when the data has been created, or if theres an
```

```
error
  return cb(null, res);
});
```

10.10. \$FH.SYNC.GLOBALHANDLECREATE

注記: RHMAP 3.19 には fh-mbaas-api バージョン 7 が含まれています。ご使用の package.json ファイルにある fh-mbaas-api のバージョンが 7.0.0 より前の場合は、[以前のバージョンのセクション](#) を参照してください。

バックエンドで単一行のデータを作成するグローバルのハンドラー関数を定義します。

これは複数のデータセットで使用可能ですが、[handleCreate API](#) を使用するデータセットにハンドラーが割り当てられていない場合にのみ使用されます。

10.10.1. 使用法

```
$fh.sync.globalHandleCreate(function createHandler(dataset_id, data,
meta_data, cb){});
```

10.10.2. パラメーター

10.10.2.1. createHandler

- 説明: レコードを作成する際に呼び出される関数。
- タイプ: 関数
- 呼び出されるパラメーター:
 - **dataset_id**
 - 説明: データセットの ID。
 - タイプ: 文字列
 - **data**
 - 説明: 作成するデータ
 - タイプ: Object
 - **meta_data**
 - 説明: データに関連付けるメタデータ。null とすることも可能。
 - タイプ: Object
 - **cb**
 - 説明: コールバック関数
 - タイプ: 関数

10.10.3. 例


```
$fh.sync.globalHandleCreate(function(dataset_id, data, meta_data, cb){
  // The dataset identifier that this function was defined for
  console.log(dataset_id);

  // Row of data to create
  console.log(data);

  // Sample back-end storage call
  var savedData = saveData(data);
  var res = {
    "uid": savedData.uid, // Unique Identifier for row
    "data": savedData.data // The created data record - including any
    system or UID fields added during the create process
  };

  // Callback function for when the data has been created, or if theres an
  error
  return cb(null, res);
});
```

10.10.4. \$fh.sync.globalHandleCreate (バージョン 7.0.0 以前の fh-mbaas-api 向け)

```
$fh.sync.globalHandleCreate(callback)
```

\$fh.sync.handleCreate と似ていますが、ハンドラーをグローバルに設定することで複数のデータセットが同一のハンドラー関数を使用できるようになります。グローバルハンドラーは、\$fh.sync.handleCreate でデータセットにハンドラーが割り当てられていない場合にのみ使用されます。

10.10.4.1. 例

```
$fh.sync.globalHandleCreate(function(dataset_id, data, cb, meta_data){
  //create data for the specified dataset_id
});
```

10.11. \$FH.SYNC.HANDLEREAD

注記: RHMAP 3.19 には fh-mbaas-api バージョン 7 が含まれています。ご使用の package.json ファイルにある fh-mbaas-api のバージョンが 7.0.0 より前の場合は、[以前のバージョンのセクション](#) を参照してください。

特定のデータセットでデータセットバックエンドから単一行のデータを読み取るハンドラー関数を定義します。

10.11.1. 使用法

```
$fh.sync.handleRead(dataset_id, function readHandler(dataset_id, uid,
meta_data, cb){});
```

10.11.2. パラメーター

10.11.2.1. dataset_id

- 説明: データセットの ID。
- タイプ: 文字列

10.11.2.2. readHandler

- 説明: レコードを読み取る際に呼び出される関数。
- タイプ: 関数
- 呼び出されるパラメーター:
 - **dataset_id**
 - 説明: データセットの ID。
 - タイプ: 文字列
 - **uid**
 - 説明: 読み取るレコードの一意的 ID
 - タイプ: 文字列
 - **meta_data**
 - 説明: データに関連付けるメタデータ。null とすることも可能。
 - タイプ: Object
 - **cb**
 - 説明: コールバック関数
 - タイプ: 関数

10.11.3. 例

```
$fh.sync.handleRead("todo", function(dataset_id, uid, meta_data, cb) {  
  // The dataset identifier that this function was defined for  
  console.log(dataset_id);  
  
  // Unique Identifier for row to read  
  console.log(uid);  
  
  // Sample back-end storage call  
  var data = readData(uid);  
  /* sample response  
    {  
      "userid": "1234",  
      "name": "Jane Bloggs",  
      "age": 27  
    }  
  */  
}
```

```
// The callback into the sync service to return the row of data
return cb(null, data);
});
```

10.11.4. \$fh.sync.handleRead (バージョン 7.0.0 以前の fh-mbaas-api 向け)

```
$fh.sync.handleRead(dataset_id, callback)
```

バックエンドから単一行のデータを読み取るハンドラー関数を定義します。データセットが初期化された後に呼び出してください。

10.11.4.1. 例

```
// data source for a dataset
$fh.sync.handleRead(dataset_id, function(dataset_id, uid, cb, meta_data) {
  // The dataset identifier that this function was defined for
  console.log(dataset_id);

  // Unique Identifier for row to read
  console.log(uid);

  // Sample back-end storage call
  var data = readData(uid);
  /* sample response
  {
    "userid": "1234",
    "name": "Jane Bloggs",
    "age": 27
  }
  */

  // The callback into the sync service to return the row of data
  return cb(null, data);
});
```

10.12. \$FH.SYNC.GLOBALHANDLEREAD

注記: RHMAP 3.19 には fh-mbaas-api バージョン 7 が含まれています。ご使用の package.json ファイルにある fh-mbaas-api のバージョンが 7.0.0 より前の場合は、[以前のバージョンのセクション](#) を参照してください。

データセットバックエンドから単一行のデータを読み取るグローバルのハンドラー関数を定義します。

これは複数のデータセットで使用可能ですが、[handleRead API](#) を使用するデータセットにハンドラーが割り当てられていない場合にのみ使用されます。

10.12.1. 使用法

```
$fh.sync.globalHandleRead(function readHandler(dataset_id, uid, meta_data,
cb){});
```

10.12.2. パラメーター

10.12.2.1. readHandler

- 説明: レコードを読み取る際に呼び出される関数。
- タイプ: 関数
- 呼び出されるパラメーター:
 - **dataset_id**
 - 説明: データセットの ID。
 - タイプ: 文字列
 - **uid**
 - 説明: 読み取るレコードの一意の ID
 - タイプ: 文字列
 - **meta_data**
 - 説明: データに関連付けるメタデータ。null とすることも可能。
 - タイプ: Object
 - **cb**
 - 説明: コールバック関数
 - タイプ: 関数

10.12.3. 例

```
$fh.sync.globalHandleRead(function(dataset_id, uid, meta_data, cb) {  
  // The dataset identifier that this function was defined for  
  console.log(dataset_id);  
  
  // Unique Identifier for row to read  
  console.log(uid);  
  
  // Sample back-end storage call  
  var data = readData(uid);  
  /* sample response  
  {  
    "userid": "1234",  
    "name": "Jane Bloggs",  
    "age": 27  
  }  
  */  
  
  // The callback into the sync service to return the row of data  
  return cb(null, data);  
});
```

10.12.4. \$fh.sync.globalHandleRead (バージョン 7.0.0 以前の fh-mbaas-api 向け)

-

```
$fh.sync.globalHandleRead(callback)
```

`$fh.sync.handleRead` と似ていますが、ハンドラーをグローバルに設定することで複数のデータセットが同一のハンドラー関数を使用できるようになります。グローバルハンドラーは、`$fh.sync.handleRead` でデータセットにハンドラーが割り当てられていない場合にのみ使用されます。

10.12.4.1. 例

```
$fh.sync.globalHandleRead(function(dataset_id, uid, cb, meta_data){
  //read data for the specified dataset_id and uid
});
```

10.13. \$FH.SYNC.HANDLEUPDATE

注記: RHMAP 3.19 には `fh-mbaas-api` バージョン 7 が含まれています。ご使用の `package.json` ファイルにある `fh-mbaas-api` のバージョンが 7.0.0 より前の場合は、[以前のバージョンのセクション](#) を参照してください。

特定のデータセットでデータセットバックエンドでの単一行のデータを更新するハンドラー関数を定義します。

同期サーバーは現行の値がバックエンドの値と一致することをチェックして、データが更新可能であることを確認します。値が一致しない場合は、更新操作は実行されず、競合が発生します。

10.13.1. 使用法

```
$fh.sync.handleUpdate(dataset_id, function updateHandler(dataset_id, uid,
data, meta_data, cb){});
```

10.13.2. パラメーター

10.13.2.1. dataset_id

- 説明: データセットの ID。
- タイプ: 文字列

10.13.2.2. updateHandler

- 説明: レコードを更新する際に呼び出される関数。
- タイプ: 関数
- 呼び出されるパラメーター:
 - **dataset_id**
 - 説明: データセットの ID。
 - タイプ: 文字列
 - **uid**

- 説明:更新するレコードの一意の ID
- タイプ: 文字列
- **data**
 - 説明: 新規の日付フィールド
 - タイプ: Object
- **meta_data**
 - 説明: データに関連付けるメタデータ。null とすることも可能。
 - タイプ: Object
- **cb**
 - 説明: コールバック関数
 - タイプ: 関数

10.13.3. 例

```
$fh.sync.handleUpdate("todo", function(dataset_id, uid, meta_data, cb) {
  // The dataset identifier that this function was defined for
  console.log(dataset_id);

  // Unique Identifier for row to update
  console.log(uid);

  // Row of data to update
  console.log(data);

  // Sample back-end storage call
  var updatedData = updateData(uid, data);
  /* sample response
  {
    "userid": "1234",
    "name": "Jane Bloggs",
    "age": 27
  }
  */

  // The callback into the sync service to return the updated row of data
  return cb(null, updatedData);
});
```

10.13.4. \$fh.sync.handleUpdate (バージョン 7.0.0 以前の fh-mbaas-api 向け)

```
$fh.sync.handleUpdate(dataset_id, callback)
```

バックエンドから単一行のデータを更新するハンドラー関数を定義します。データセットが初期化された後に呼び出してください。

10.13.4.1. 例

```
// data source for a dataset.
// The sync service will verify that the update can proceed
// (that is, collision detection) before it invokes the update function.
$fh.sync.handleUpdate(dataset_id, function(dataset_id, uid, data, cb,
meta_data) {
    // The dataset identifier that this function was defined for
    console.log(dataset_id);

    // Unique Identifier for row to update
    console.log(uid);

    // Row of data to update
    console.log(data);

    // Sample back-end storage call
    var updatedData = updateData(uid, data);
    /* sample response
    {
        "userid": "1234",
        "name": "Jane Bloggs",
        "age": 27
    }
    */

    // The callback into the sync service to return the updated row of data
    return cb(null, updatedData);
});
```

10.14. \$FH.SYNC.GLOBALHANDLEUPDATE

注記: RHMAP 3.19 には fh-mbaas-api バージョン 7 が含まれています。ご使用の package.json ファイルにある fh-mbaas-api のバージョンが 7.0.0 より前の場合は、[以前のバージョンのセクション](#)を参照してください。

データセットバックエンド上の単一行データを更新するグローバルのハンドラー関数を定義します。

これは複数のデータセットで使用可能ですが、[handleUpdate API](#) を使用するデータセットにハンドラーが割り当てられていない場合にのみ使用されます。

同期サーバーは現行の値がデータセットバックエンドの値と一致することをチェックして、データが更新可能であることを確認します。値が一致しない場合は、更新操作は実行されず、競合が発生します。

10.14.1. 使用法

```
$fh.sync.globalHandleUpdate(function updateHandler(dataset_id, uid, data,
meta_data, cb){});
```

10.14.2. パラメーター

10.14.2.1. updateHandler

- 説明: レコードを更新する際に呼び出される関数。
- タイプ: 関数
- 呼び出されるパラメーター:
 - **dataset_id**
 - 説明: データセットの ID。
 - タイプ: 文字列
 - **uid**
 - 説明: 更新するレコードの一意の ID
 - タイプ: 文字列
 - **data**
 - 説明: 新規の日付フィールド
 - タイプ: Object
 - **meta_data**
 - 説明: データに関連付けるメタデータ。null とすることも可能。
 - タイプ: Object
 - **cb**
 - 説明: コールバック関数
 - タイプ: 関数

10.14.3. 例

```
$fh.sync.globalHandleUpdate(function(dataset_id, uid, meta_data, cb) {  
  // The dataset identifier that this function was defined for  
  console.log(dataset_id);  
  
  // Unique Identifier for row to update  
  console.log(uid);  
  
  // Row of data to update  
  console.log(data);  
  
  // Sample back-end storage call  
  var updatedData = updateData(uid, data);  
  /* sample response  
  {  
    "userid": "1234",  
    "name": "Jane Bloggs",  
    "age": 27  
  }  
  */  
}
```



```
// The callback into the sync service to return the updated row of data
return cb(null, updatedData);
});
```

10.14.4. \$fh.sync.globalHandleUpdate (バージョン 7.0.0 以前の fh-mbaas-api 向け)

```
$fh.sync.globalHandleUpdate(callback)
```

\$fh.sync.handleUpdate と似ていますが、ハンドラーをグローバルに設定することで複数のデータセットが同一のハンドラー関数を使用できるようになります。グローバルハンドラーは、\$fh.sync.handleUpdate でデータセットにハンドラーが割り当てられていない場合にのみ使用されます。

10.14.4.1. 例

```
$fh.sync.globalHandleUpdate(function(dataset_id, uid, data, cb,
meta_data){
  //update data for the specified dataset_id and uid
});
```

10.15. \$FH.SYNC.HANDLEDELETE

注記: RHMAP 3.19 には fh-mbaas-api バージョン 7 が含まれています。ご使用の package.json ファイルにある fh-mbaas-api のバージョンが 7.0.0 より前の場合は、[以前のバージョンのセクション](#) を参照してください。

特定のデータセットでデータセットバックエンドから単一行のデータを削除するハンドラー関数を定義します。同期サーバーは、行が最新であり、競合が発生しないことを確認し、データを削除可能にします。

10.15.1. 使用法

```
$fh.sync.handleDelete(dataset_id, function deleteHandler(dataset_id, uid,
meta_data, cb){});
```

10.15.2. パラメーター

10.15.2.1. dataset_id

- 説明: データセットの ID。
- タイプ: 文字列

10.15.2.2. deleteHandler

- 説明: レコードを削除する際に呼び出される関数。
- タイプ: 関数
- 呼び出されるパラメーター:
 - dataset_id

- 説明: データセットの ID。
- タイプ: 文字列
- **uid**
 - 説明: 削除するレコードの一意の ID
 - タイプ: 文字列
- **meta_data**
 - 説明: データに関連付けるメタデータ。null とすることも可能。
 - タイプ: Object
- **cb**
 - 説明: コールバック関数
 - タイプ: 関数

10.15.3. 例

```
$fh.sync.handleDelete("todo", function(dataset_id, uid, meta_data, cb) {
  // The dataset identifier that this function was defined for
  console.log(dataset_id);

  // Unique Identifier for row to update
  console.log(uid);

  // Sample back-end storage call
  var deletedData = deleteData(uid);

  /* sample response
  {
    "userid": "1234",
    "name": "Jane Bloggs",
    "age": 27
  }
  */

  // The callback into the sync service to return the deleted row of data
  return cb(null, deletedData);
});
```

10.15.4. \$fh.sync.handleDelete (バージョン 7.0.0 以前の fh-mbaas-api 向け)

```
$fh.sync.handleDelete(dataset_id, callback)
```

バックエンドから単一行のデータを削除するハンドラー関数を定義します。データセットが初期化された後に呼び出してください。

10.15.4.1. 例

```
// data source for a dataset.
// The sync service will verify that the delete can proceed
// (that is, collision detection) before it invokes the delete function.
$fh.sync.handleDelete(dataset_id, function(dataset_id, uid, cb, meta_data)
{
  // The dataset identifier that this function was defined for
  console.log(dataset_id);

  // Unique Identifier for row to update
  console.log(uid);

  // Sample back-end storage call
  var deletedData = deleteData(uid);

  /* sample response
  {
    "userid": "1234",
    "name": "Jane Bloggs",
    "age": 27
  }
  */

  // The callback into the sync service to return the deleted row of data
  return cb(null, deletedData);
});
```

10.16. \$FH.SYNC.GLOBALHANDLEDELETE

注記: RHMAP 3.19 には fh-mbaas-api バージョン 7 が含まれています。ご使用の package.json ファイルにある fh-mbaas-api のバージョンが 7.0.0 より前の場合は、[以前のバージョンのセクション](#) を参照してください。

データセットバックエンドから単一行データを削除するグローバルのハンドラー関数を定義します。

これは複数のデータセットで使用可能ですが、[handleDelete API](#) を使用するデータセットにハンドラーが割り当てられていない場合にのみ使用されます。

同期サーバーは、行が最新であり、競合が発生しないことを確認し、データを削除可能にします。

10.16.1. 使用法

```
$fh.sync.globalHandleDelete(function deleteHandler(dataset_id, uid,
meta_data, cb){});
```

10.16.2. パラメーター

10.16.2.1. deleteHandler

- 説明: レコードを削除する際に呼び出される関数。
- タイプ: 関数
- 呼び出されるパラメーター:

- **dataset_id**
 - 説明: データセットの ID。
 - タイプ: 文字列
- **uid**
 - 説明: 削除するレコードの一意的 ID
 - タイプ: 文字列
- **meta_data**
 - 説明: データに関連付けるメタデータ。null とすることも可能。
 - タイプ: Object
- **cb**
 - 説明: コールバック関数
 - タイプ: 関数

10.16.3. 例

```
$fh.sync.globalHandleDelete(function(dataset_id, uid, meta_data, cb) {
  // The dataset identifier that this function was defined for
  console.log(dataset_id);

  // Unique Identifier for row to update
  console.log(uid);

  // Sample back-end storage call
  var deletedData = deleteData(uid);

  /* sample response
  {
    "userid": "1234",
    "name": "Jane Bloggs",
    "age": 27
  }
  */

  // The callback into the sync service to return the deleted row of data
  return cb(null, deletedData);
});
```

10.16.4. \$fh.sync.globalHandleDelete (バージョン 7.0.0 以前の fh-mbaas-api 向け)

```
$fh.sync.globalHandleDelete(callback)
```

\$fh.sync.handleDelete と似ていますが、ハンドラーをグローバルに設定することで複数のデータセットが同一のハンドラー関数を使用できるようになります。グローバルハンドラーは、\$fh.sync.handleDelete でデータセットにハンドラーが割り当てられていない場合にのみ使用されます。

10.16.4.1. 例

```
$fh.sync.globalHandleDelete(function(dataset_id, uid, cb, meta_data){
  //delete data for the specified dataset_id and uid
});
```

10.17. \$FH.SYNC.HANDLECOLLISION

注記: RHMAP 3.19 には fh-mbaas-api バージョン 7 が含まれています。ご使用の package.json ファイルにある fh-mbaas-api のバージョンが 7.0.0 より前の場合は、[以前のバージョンのセクション](#) を参照してください。

特定のデータセットのデータ競合を処理するハンドラー関数を定義します。

競合はこの関数で解決するか、このまま保持してこれ以降に見直すことができます。

10.17.1. 使用法

```
$fh.sync.handleCollision(dataset_id, function collisionHandler(dataset_id,
hash, timestamp, uid, pre, post, meta_data){});
```

10.17.2. パラメーター

10.17.2.1. dataset_id

- 説明: データセットの ID。
- タイプ: 文字列

10.17.2.2. collisionHandler

- 説明: 競合が発生した際に呼び出される関数。
- タイプ: 関数
- 呼び出されるパラメーター:
 - **dataset_id**
 - 説明: データセットの ID。
 - タイプ: 文字列
 - **hash**
 - 説明: 競合の一意のハッシュ値
 - タイプ: 文字列
 - **timestamp**
 - 説明: クライアント上で変更が発生した際のタイムスタンプ (ミリ秒単位)。
 - タイプ: 数字

- **uid**
 - 説明: 保留中レコードの一意の ID
 - タイプ: 文字列
- **pre**
 - 説明: 変更前のデータ
 - タイプ: Object
- **post**
 - 説明: 変更後のデータ
 - タイプ: Object
- **meta_data**
 - 説明: データに関連付けるメタデータ。null とすることも可能。
 - タイプ: Object

10.17.3. 例

```
// Typically a collision handler will write the data record to a
collisions table
// which is reviewed by a user who can manually reconcile the collisions.
$fh.sync.handleCollision("todo", function(dataset_id, hash, timestamp,
uid, pre, post, meta_data, cb) {
  // The dataset identifier that this function was defined for
  console.log(dataset_id);

  // Unique hash value identifying the collision
  console.log(hash);

  // Date / time that update was created on client device
  console.log(timestamp);

  // Unique Identifier for row
  console.log(uid);

  // The data row the client started with
  console.log(pre);

  //The data row the client tried to write
  console.log(post);

  // sample back-end storage call
  saveCollisionData(dataset_id, hash, timestamp, uid, pre, post);
});
```

10.17.4. \$fh.sync.handleCollision (バージョン 7.0.0 以前の fh-mbaas-api 向け)

```
$fh.sync.handleCollision(dataset_id, callback)
```

データ競合 (古い更新) に対処するハンドラー関数を定義します。データセットが初期化された後に呼び出してください。

10.17.4.1. 例

```
// Typically a collision handler will write the data record to a
// collisions table
// which is reviewed by a user who can manually reconcile the collisions.
$fh.sync.handleCollision(dataset_id, function(dataset_id, hash, timestamp,
uid, pre, post, meta_data) {
  // The dataset identifier that this function was defined for
  console.log(dataset_id);

  // Unique hash value identifying the collision
  console.log(hash);

  // Date / time that update was created on client device
  console.log(timestamp);

  // Unique Identifier for row
  console.log(uid);

  // The data row the client started with
  console.log(pre);

  //The data row the client tried to write
  console.log(post);

  // sample back-end storage call
  saveCollisionData(dataset_id, hash, timestamp, uid, pre, post);
});
```

10.18. \$FH.SYNC.GLOBALHANDLECOLLISION

注記: RHMAP 3.19 には fh-mbaas-api バージョン 7 が含まれています。ご使用の package.json ファイルにある fh-mbaas-api のバージョンが 7.0.0 より前の場合は、[以前のバージョンのセクション](#) を参照してください。

データ競合を処理するグローバルハンドラー関数を定義します。

競合はこの関数で解決するか、このまま保持してこれ以降に見直すことができます。

これは複数のデータセットで使用可能ですが、[handleCollision API](#) を使用するデータセットにハンドラーが割り当てられていない場合にのみ使用されます。

10.18.1. 使用法

```
$fh.sync.globalHandleCollision(function collisionHandler(dataset_id, hash,
timestamp, uid, pre, post, meta_data){});
```

10.18.2. パラメーター

10.18.2.1. collisionHandler

- 説明: 競合が発生した際に呼び出される関数。
- タイプ: 関数
- 呼び出されるパラメーター:
 - **dataset_id**
 - 説明: データセットの ID。
 - タイプ: 文字列
 - **hash**
 - 説明: 競合の一意のハッシュ値
 - タイプ: 文字列
 - **timestamp**
 - 説明: クライアント上で変更が発生した際のタイムスタンプ (ミリ秒単位)。
 - タイプ: 数字
 - **uid**
 - 説明: 保留中レコードの一意の ID
 - タイプ: 文字列
 - **pre**
 - 説明: 変更前のデータ
 - タイプ: Object
 - **post**
 - 説明: 変更後のデータ
 - タイプ: Object
 - **meta_data**
 - 説明: データに関連付けるメタデータ。null とすることも可能。
 - タイプ: Object

10.18.3. 例

```
// Typically a collision handler will write the data record to a
collisions table
// which is reviewed by a user who can manually reconcile the collisions.
$fh.sync.globalHandleCollision(function(dataset_id, hash, timestamp, uid,
pre, post, meta_data) {
  // The dataset identifier that this function was defined for
  console.log(dataset_id);
```



```
// Unique hash value identifying the collision
console.log(hash);

// Date / time that update was created on client device
console.log(timestamp);

// Unique Identifier for row
console.log(uid);

// The data row the client started with
console.log(pre);

//The data row the client tried to write
console.log(post);

// sample back-end storage call
saveCollisionData(dataset_id, hash, timestamp, uid, pre, post);
});
```

10.18.4. \$fh.sync.globalHandleCollision (バージョン 7.0.0 以前の fh-mbaas-api 向け)

```
$fh.sync.globalHandleCollision(callback)
```

\$fh.sync.handleCollision と似ていますが、ハンドラーをグローバルに設定することで複数のデータセットが同一のハンドラー関数を使用できるようになります。グローバルハンドラーは、\$fh.sync.handleCollision でデータセットにハンドラーが割り当てられていない場合にのみ使用されます。

10.18.4.1. 例

```
$fh.sync.globalHandleCollision(function(dataset_id, hash, timestamp, uid,
pre, post, meta_data){
});
```

10.19. \$FH.SYNC.LISTCOLLISIONS

注記: RHMAP 3.19 には fh-mbaas-api バージョン 7 が含まれています。ご使用の package.json ファイルにある fh-mbaas-api のバージョンが 7.0.0 より前の場合は、[以前のバージョンのセクション](#) を参照してください。

データセットの現在の競合一覧を返すハンドラー関数を定義します。

10.19.1. 使用法

```
$fh.sync.listCollisions(dataset_id, function
listCollisionsHandler(dataset_id, meta_data, cb){});
```

10.19.2. パラメーター

10.19.2.1. dataset_id

- 説明: データセットの ID。
- タイプ: 文字列

10.19.2.2. listCollisionsHandler

- 説明: 競合を一覧表示する際に呼び出される関数。
- タイプ: 関数
- 呼び出されるパラメーター:
 - **dataset_id**
 - 説明: データセットの ID。
 - タイプ: 文字列
 - **meta_data**
 - 説明: データに関連付けるメタデータ。null とすることも可能。
 - タイプ: Object
 - **cb**
 - 説明: コールバック関数
 - タイプ: 関数

10.19.3. 例

```
// This would usually be used by an administration console where a user is
// manually reviewing & resolving collisions.
$fh.sync.listCollisions("todo", function(dataset_id, meta_data, cb) {
  // The dataset identifier that this function was defined for
  console.log(dataset_id);

  // sample back-end storage call
  var collisions = getCollisions(dataset_id);
  /* sample response:
  {
    "collision_hash_1" : {
      "uid": "<uid_of_data_row>",
      "timestamp": "<timestamp_value_passed_to_handleCollision_fn>",
      "pre": "<pre_data_record_passed_to_handleCollision_fn>",
      "post": "<post_data_record_passed_to_handleCollision_fn>"
    },
    "collision_hash_2" : {
      "uid": "<uid_of_data_row>",
      "timestamp": "<timestamp_value_passed_to_handleCollision_fn>",
      "pre": "<pre_data_record_passed_to_handleCollision_fn>",
      "post": "<post_data_record_passed_to_handleCollision_fn>"
    },
    "collision_hash_2" : {
```

```

        "uid": "<uid_of_data_row>",
        "timestamp": "<timestamp_value_passed_to_handleCollision_fn>",
        "pre": "<pre_data_record_passed_to_handleCollision_fn>",
        "post": "<post_data_record_passed_to_handleCollision_fn>"
    }
}
*/

// The callback into the sync service to return the list of known
collisions
return cb(null, collisions);
});

```

10.19.4. \$fh.sync.listCollisions (バージョン 7.0.0 以前の fh-mbaas-api 向け)

```
$fh.sync.listCollisions(dataset_id, callback)
```

競合の現在の一覧を返すハンドラー関数を定義します。データセットが初期化された後に呼び出してください。

10.19.4.1. 例

```

// This would usually be used by an administration console where a user is
// manually reviewing & resolving collisions.
$fh.sync.listCollisions(dataset_id, function(dataset_id, cb, meta_data) {
    // The dataset identifier that this function was defined for
    console.log(dataset_id);

    // sample back-end storage call
    var collisions = getCollisions(dataset_id);
    /* sample response:
    {
        "collision_hash_1" : {
            "uid": "<uid_of_data_row>",
            "timestamp": "<timestamp_value_passed_to_handleCollision_fn>",
            "pre": "<pre_data_record_passed_to_handleCollision_fn>",
            "post": "<post_data_record_passed_to_handleCollision_fn>"
        },
        "collision_hash_2" : {
            "uid": "<uid_of_data_row>",
            "timestamp": "<timestamp_value_passed_to_handleCollision_fn>",
            "pre": "<pre_data_record_passed_to_handleCollision_fn>",
            "post": "<post_data_record_passed_to_handleCollision_fn>"
        },
        "collision_hash_2" : {
            "uid": "<uid_of_data_row>",
            "timestamp": "<timestamp_value_passed_to_handleCollision_fn>",
            "pre": "<pre_data_record_passed_to_handleCollision_fn>",
            "post": "<post_data_record_passed_to_handleCollision_fn>"
        }
    }
    */

    // The callback into the sync service to return the list of known
    collisions

```

```
return cb(null, collisions);
});
```



注記

"collision_hash" は競合を一意に定義するハッシュ値です。この値は、"handleCollision" 関数で競合が最初に作成される際に、"hash" パラメーターとして渡されています。

10.20. \$FH.SYNC.GLOBALLISTCOLLISIONS

注記: RHMAP 3.19 には fh-mbaas-api バージョン 7 が含まれています。ご使用の package.json ファイルにある fh-mbaas-api のバージョンが 7.0.0 より前の場合は、[以前のバージョンのセクション](#) を参照してください。

現在の競合一覧を返すグローバルハンドラー関数を定義します。

これは複数のデータセットで使用可能ですが、[listCollisions API](#) を使用するデータセットにハンドラーが割り当てられていない場合にのみ使用されます。

10.20.1. 使用法

```
$fh.sync.globalListCollisions(function listCollisionsHandler(dataset_id,
meta_data, cb){});
```

10.20.2. パラメーター

10.20.2.1. listCollisionsHandler

- 説明: 競合を一覧表示する際に呼び出される関数。
- タイプ: 関数
- 呼び出されるパラメーター:
 - **dataset_id**
 - 説明: データセットの ID。
 - タイプ: 文字列
 - **meta_data**
 - 説明: データに関連付けるメタデータ。null とすることも可能。
 - タイプ: Object
 - **cb**
 - 説明: コールバック関数
 - タイプ: 関数

10.20.3. 例

```

// This would usually be used by an administration console where a user is
// manually reviewing & resolving collisions.
$fh.sync.globalListCollisions(function(dataset_id, meta_data, cb) {
  // The dataset identifier that this function was defined for
  console.log(dataset_id);

  // sample back-end storage call
  var collisions = getCollisions(dataset_id);
  /* sample response:
  {
    "collision_hash_1" : {
      "uid": "<uid_of_data_row>",
      "timestamp": "<timestamp_value_passed_to_handleCollision_fn>",
      "pre": "<pre_data_record_passed_to_handleCollision_fn>",
      "post": "<post_data_record_passed_to_handleCollision_fn>"
    },
    "collision_hash_2" : {
      "uid": "<uid_of_data_row>",
      "timestamp": "<timestamp_value_passed_to_handleCollision_fn>",
      "pre": "<pre_data_record_passed_to_handleCollision_fn>",
      "post": "<post_data_record_passed_to_handleCollision_fn>"
    },
    "collision_hash_2" : {
      "uid": "<uid_of_data_row>",
      "timestamp": "<timestamp_value_passed_to_handleCollision_fn>",
      "pre": "<pre_data_record_passed_to_handleCollision_fn>",
      "post": "<post_data_record_passed_to_handleCollision_fn>"
    }
  }
  */

  // The callback into the sync service to return the list of known
  collisions
  return cb(null, collisions);
});

```

10.20.4. \$fh.sync.globalListCollisions (バージョン 7.0.0 以前の fh-mbaas-api 向け)

```
$fh.sync.globalListCollisions(callback)
```

\$fh.sync.listCollisions と似ていますが、ハンドラーをグローバルに設定することで複数のデータセットが同一のハンドラー関数を使用できるようになります。グローバルハンドラーは、\$fh.sync.listCollisions でデータセットにハンドラーが割り当てられていない場合にのみ使用されます。

10.20.4.1. 例

```

$fh.sync.globalListCollisions(function(dataset_id, cb, meta_data){
  });

```

10.21. \$FH.SYNC.REMOVECOLLISION

注記: RHMAP 3.19 には fh-mbaas-api バージョン 7 が含まれています。ご使用の package.json ファイルにある fh-mbaas-api のバージョンが 7.0.0 より前の場合は、[以前のバージョンのセクション](#) を参照してください。

データセットの競合を削除するハンドラー関数を定義します。

10.21.1. 使用法

```
$fh.sync.removeCollision(dataset_id, function  
removeCollisionHandler(dataset_id, collision_hash, meta_data, cb){});
```

10.21.2. パラメーター

10.21.2.1. dataset_id

- 説明: データセットの ID。
- タイプ: 文字列

10.21.2.2. removeCollisionHandler

- 説明: 競合を削除する際に呼び出される関数。
- タイプ: 関数
- 呼び出されるパラメーター:
 - **dataset_id**
 - 説明: データセットの ID。
 - タイプ: 文字列
 - **collision_hash**
 - 説明: 競合の一意のハッシュ
 - タイプ: 文字列
 - **meta_data**
 - 説明: データに関連付けるメタデータ。null とすることも可能。
 - タイプ: Object
 - **cb**
 - 説明: コールバック関数
 - タイプ: 関数

10.21.3. 例

```
// This would usually be used by an administration console where a user is  
// manually reviewing & resolving collisions.
```

```
$fh.sync.removeCollision("todo", function(dataset_id, collision_hash,
meta_data, cb) {
  // The dataset identifier that this function was defined for
  console.log(dataset_id);

  // sample back-end storage call
  removeCollision(collision_hash);

  // The callback into the sync service to return the delete row of data
  return cb(null);
});
```

10.21.4. \$fh.sync.removeCollision (バージョン 7.0.0 以前の fh-mbaas-api 向け)

```
$fh.sync.removeCollision(dataset_id, callback)
```

競合一覧からある競合を削除するハンドラー関数を定義します。データセットが初期化された後に呼び出してください。

10.21.4.1. 例

```
// This would usually be used by an administration console where a user is
// manually reviewing & resolving collisions.
$fh.sync.removeCollision(dataset_id, function(dataset_id, collision_hash,
cb, meta_data) {
  // The dataset identifier that this function was defined for
  console.log(dataset_id);

  // sample back-end storage call
  removeCollision(collision_hash);

  // The callback into the sync service to return the delete row of data
  return cb(null);
});
```

10.22. \$FH.SYNC.GLOBALREMOVECOLLISION

注記: RHMAP 3.19 には fh-mbaas-api バージョン 7 が含まれています。ご使用の package.json ファイルにある fh-mbaas-api のバージョンが 7.0.0 より前の場合は、[以前のバージョンのセクション](#) を参照してください。

競合を削除するグローバルハンドラー関数を定義します。

これは複数のデータセットで使用可能ですが、[removeCollision API](#) を使用するデータセットにハンドラーが割り当てられていない場合にのみ使用されます。

10.22.1. 使用法

```
$fh.sync.globalRemoveCollision(function removeCollisionHandler(dataset_id,
collision_hash, meta_data, cb){});
```

10.22.2. パラメーター

10.22.2.1. removeCollisionHandler

- 説明: 競合を削除する際に呼び出される関数。
- タイプ: 関数
- 呼び出されるパラメーター:
 - **dataset_id**
 - 説明: データセットの ID。
 - タイプ: 文字列
 - **collision_hash**
 - 説明: 競合の一意のハッシュ
 - タイプ: 文字列
 - **meta_data**
 - 説明: データに関連付けるメタデータ。null とすることも可能。
 - タイプ: Object
 - **cb**
 - 説明: コールバック関数
 - タイプ: 関数

10.22.3. 例

```
// This would usually be used by an administration console where a user is
// manually reviewing & resolving collisions.
$fh.sync.globalRemoveCollision(function(dataset_id, collision_hash,
meta_data, cb) {
  // The dataset identifier that this function was defined for
  console.log(dataset_id);

  // sample back-end storage call
  removeCollision(collision_hash);

  // The callback into the sync service to return the delete row of data
  return cb(null);
});
```

10.22.4. \$fh.sync.globalRemoveCollision (バージョン 7.0.0 以前の fh-mbaas-api 向け)

```
$fh.sync.globalRemoveCollision(callback)
```

`$fh.sync.removeCollision` と似ていますが、ハンドラーをグローバルに設定することで複数のデータセットが同一のハンドラー関数を使用できるようになります。グローバルハンドラーは、`$fh.sync.removeCollision` でデータセットにハンドラーが割り当てられていない場合にのみ使用されま

す。

10.22.4.1. 例

```
$fh.sync.globalRemoveCollision(function(dataset_id, collision_hash, cb,
meta_data){
});
```

10.23. \$FH.SYNC.INTERCEPTREQUEST

注記: RHMAP 3.19 には fh-mbaas-api バージョン 7 が含まれています。ご使用の package.json ファイルにある fh-mbaas-api のバージョンが 7.0.0 より前の場合は、[以前のバージョンのセクション](#) を参照してください。

指定されたデータセットの同期リクエストを処理される前にインターセプトします。クライアント ID のチェックや認証を検証する際にこの API を使用します。

10.23.1. 使用法

```
$fh.sync.interceptRequest(dataset_id, function
requestInterceptor(dataset_id, interceptorParams, cb){});
```

10.23.2. パラメーター

10.23.2.1. dataset_id

- 説明: データセットの ID
- タイプ: 文字列

10.23.2.2. requestInterceptor

- 説明: クライアントから同期リクエストを受け付けた際に呼び出される関数。
- タイプ: 関数
- 呼び出されるパラメーター:
 - **dataset_id**
 - 説明: データセットの ID
 - タイプ: 文字列
 - **interceptorParams**
 - 説明: リクエストに関連付けられたパラメーター
 - タイプ: Object
 - サポートされるキー
 - query_params

- 説明: リクエストに関連付けられたクエリーパラメーター
 - タイプ: Object
- meta_data
 - 説明: リクエストに関連付けられたメタデータ
 - タイプ: Object
- cb
 - 説明: コールバック関数
null 以外の応答で呼び出されると、同期リクエストは拒否されます。
 - タイプ: 関数

10.23.3. 例

```
$fh.sync.interceptRequest("todo", function(dataset_id, interceptorParams,
cb){

    var query_params = interceptorParams.query_params; //the query_params
specified in the client $fh.sync.manage
    var meta_data = interceptorParams.meta_data; //the meta_data specified
in the client $fh.sync.manage

    var validUser = function(qp, meta){
        //implement user authentication and return true or false
    };

    if(validUser(query_params, meta_data)){
        return cb(null);
    } else {
        // Return a non null response to cause the sync request to fail.
        return cb({error: 'invalid user'});
    }
});
```

10.23.4. \$fh.sync.interceptRequest (バージョン 7.0.0 以前の fh-mbaas-api 向け)

```
$fh.sync.interceptRequest(dataset_id, callback);
```

指定されたデータセットの同期リクエストをインターセプトします。クライアント ID のチェックや認証の検証に便利です。

10.23.4.1. 例

```
$fh.sync.interceptRequest(dataset_id, function(dataset_id,
interceptorParams, cb){

    var query_params = interceptorParams.query_params; //the query_params
specified in the client $fh.sync.manage
    var meta_data = interceptorParams.meta_data; //the meta_data specified
```

```
in the client $fh.sync.manage
```

```
var validUser = function(qp, meta){
  //implement user authentication and return true or false
};

if(validUser(query_params, meta_data)){
  return cb(null);
} else {
  // Return a non null response to cause the sync request to fail.
  return cb({error: 'invalid user'});
}
});
```

10.24. \$FH.SYNC.SETRECORDHASHFN

単一レコードのハッシュ値を生成する関数を定義します。キャッシュされたレコードをデータセットバックエンドからの最新のものと比較する際には、レコードハッシュが使用されます。

ハッシュが一致すればレコードは同一のものとみなされます (そのため、キャッシュまたはクライアントを更新する必要がありません)。デフォルトの実装では、レコードの並び替え stringify を行い、その sha-1 を計算するという保守的なアプローチを取っています。

ただし、CPU の時間を節約するには、これを上書きしてください。

重要 サーバー側で更新されたフィールドは、ハッシュ計算での使用は避けてください。例えば、データセットバックエンドでの保存時に更新された lastModified フィールドがデータセットにある場合は、ハッシュ計算からこれを省いてください。これを省かないと、この後の更新で競合が発生する可能性が高くなります。データがクライアントと同期されるまでは、クライアントはそのフィールドの新たな値を認識しないためです。

この関数を使用すると、デフォルト実装を上書きすることができます。

10.24.1. 使用法

```
$fh.sync.setRecordHashFn(dataset_id, function generateHash(dataset_id,
record){});
```

10.24.2. パラメーター

10.24.2.1. dataset_id

- 説明: データセットの ID。
- タイプ: 文字列

10.24.2.2. generateHash

- 説明: データセットのハッシュ生成に使用される関数
- タイプ: 関数
- 呼び出されるパラメーター:

- **dataset_id**

- 説明: データセットの ID。
- タイプ: 文字列

- **record**

- 説明: レコード
- タイプ: Object

10.24.3. Returns

レコードのハッシュ値。

10.24.4. 例

```
$fh.sync.setRecordHashFn("todo", function(dataset_id, record){  
    return record.lastModified;  
});
```

10.25. \$FH.SYNC.SETGLOBALHASHFN

データセットのグローバルハッシュを計算する関数を定義します。グローバルハッシュは、データセットに変更があったかどうかを判断するために使用されます。

重要 サーバー側で更新されたフィールドは、ハッシュ計算での使用は避けてください。例えば、データセットバックエンドでの保存時に更新された lastModified フィールドがデータセットにある場合は、ハッシュ計算からこれを省いてください。これを省かないと、この後の更新で競合が発生する可能性が高くなります。データがクライアントと同期されるまでは、クライアントはそのフィールドの新たな値を認識しないためです。

10.25.1. 使用法

```
$fh.sync.setGlobalHashFn(dataset_id, function  
generateHash(dataRecordHashes){});
```

10.25.2. パラメーター

10.25.2.1. dataset_id

- 説明: データセットの ID。
- タイプ: 文字列

10.25.2.2. generateHash

- 説明: データセットのハッシュ生成に使用される関数
- タイプ: 関数
- 呼び出されるパラメーター:

- **dataRecordHashes**

- 説明: データセット内の全レコードの全ハッシュの配列
- タイプ: 配列

10.25.3. Returns

データセットのグローバルハッシュ値。

10.25.4. 例

```
$fh.sync.setGlobalHashFn("todo", function(dataRecordHashes){
  return dataRecordHashes.join('');
});
```

10.26. \$FH.SYNC.GLOBALINTERCEPTREQUEST

注記: RHMAP 3.19 には fh-mbaas-api バージョン 7 が含まれています。ご使用の package.json ファイルにある fh-mbaas-api のバージョンが 7.0.0 より前の場合は、[以前のバージョンのセクション](#) を参照してください。

全同期リクエストを処理される前にインターセプトします。クライアント ID のチェックや認証を検証する際にこの API を使用します。

これは複数のデータセットで使用可能ですが、[interceptRequest API](#) を使用するデータセットにハンドラーが割り当てられていない場合にのみ使用されます。

10.26.1. 使用法

```
$fh.sync.globalInterceptRequest(function requestInterceptor(dataset_id,
  interceptorParams, cb){});
```

10.26.2. パラメーター

10.26.2.1. requestInterceptor

- 説明: クライアントから同期リクエストを受け付けた際に呼び出される関数。
- タイプ: 関数
- 呼び出されるパラメーター:
 - **dataset_id**
 - 説明: データセットの ID。
 - タイプ: 文字列
 - **interceptorParams**
 - 説明: リクエストに関連付けられたパラメーター
 - タイプ: Object

- サポートされるキー

- query_params
 - 説明: リクエストに関連付けられたクエリーパラメーター
 - タイプ: Object
- meta_data
 - 説明: リクエストに関連付けられたメタデータ
 - タイプ: Object

- cb

- 説明: コールバック関数
null 以外の応答で呼び出されると、同期リクエストは拒否されます。
- タイプ: 関数

10.26.3. 例

```
$fh.sync.globalInterceptRequest(function(dataset_id, interceptorParams,
cb){

    var query_params = interceptorParams.query_params; //the query_params
specified in the client $fh.sync.manage
    var meta_data = interceptorParams.meta_data; //the meta_data specified
in the client $fh.sync.manage

    var validUser = function(qp, meta){
        //implement user authentication and return true or false
    };

    if(validUser(query_params, meta_data)){
        return cb(null);
    } else {
        // Return a non null response to cause the sync request to fail.
        return cb({error: 'invalid user'});
    }
});
```

10.26.4. \$fh.sync.globalInterceptRequest (バージョン 7.0.0 以前の fh-mbaas-api 向け)

```
$fh.sync.globalInterceptRequest(callback)
```

\$fh.sync.interceptRequest と似ていますが、ハンドラーをグローバルに設定することで複数のデータセットが同一のハンドラー関数を使用できるようになります。グローバルハンドラーは、\$fh.sync.interceptRequest でデータセットにハンドラーが割り当てられていない場合にのみ使用されます。

10.26.4.1. 例

```
$fh.sync.globalInterceptRequest(function(dataset_id, interceptorParams,
cb){

});
```

10.27. \$FH.SYNC.GETEVENTEMITTER

同期サーバーのイベントエミッターを取得します。

fh-mbaas-api からの同期を使用している場合は、**\$fh.events** の場合と同様のエミッターインスタンスが返されます。

10.27.1. 使用法

```
var emitter = $fh.sync.getEventEmitter();
```

10.27.2. パラメーター

なし

10.27.3. 例

```
var emitter = $fh.sync.getEventEmitter();
emitter.once('sync:ready', function(){
  //do something here
});
```