Red Hat Mobile Application Platform 4.3
Cloud API

For Red Hat Mobile Application Platform 4.3

Red Hat Customer Content Services

Red Hat Mobile Application Platform 4.3 Cloud API

For Red Hat Mobile Application Platform 4.3

Legal Notice

Abstract

This document is a reference of the RHMAP Cloud API.

Table of Contents

CHAPTER 1. $FH.CACHE

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

Store, read or delete a value in the cache layer of your Cloud App.

By default, the cache store has a memory limit of 1GB. Once the limit is reached, the store starts removing data using an LRU (Least Recently Used) algorithm.

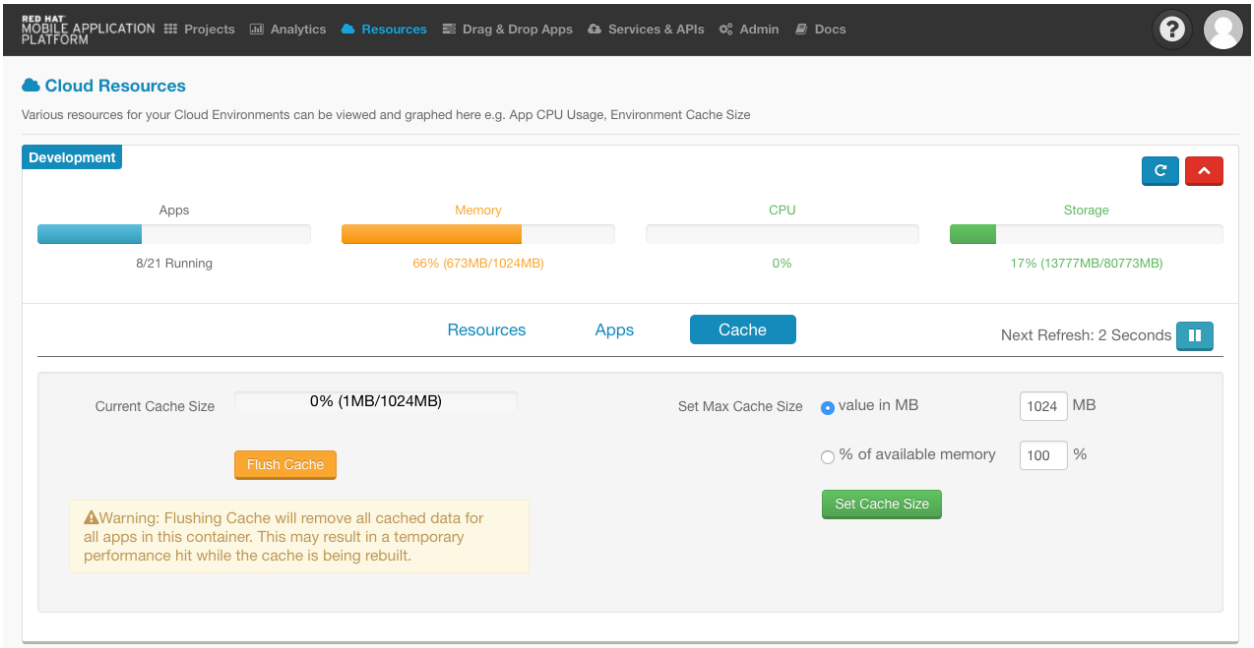
If the option "expire" is not specified when storing a value in the cache, that value remains in the cache until the memory limit is reached. Then, the LRU algorithm determines which value is removed.

1.1. MANAGING THE CACHE OPTIONS

The memory limit for the cache can be managed in the Studio, which also allows you to flush the cache store. These two options are available in **`Resources > [environment] > Cache`**

Note

The Max Cache Size limit setting affects all Cloud Apps for an environment.



1.2. EXAMPLES

Save a value to the cache

```
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());
});
```

Load a value from the cache

```
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());
});
```

Remove a value from the cache

```
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());
});
```

CHAPTER 2. $FH.DB

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

Access to hosted data storage. It supports CRUDL (create, read, update, delete, list) and index operations. It also supports deleteall, which deletes all the records in the specified entity.

2.1. EXAMPLE

Create a single entry (row)

```
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": "4e563ea44fe8e7fc19000002", // unique id for this data
entry
        "type": "myFirstEntity"
      }
    */
  }
});
```

Create multiple records in one call

```
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
       }
    */
  }
});
```

Read a single entry

```
var options = {
  "act": "read",
  "type": "myFirstEntity", // Entity/Collection name
  "guid": "4e563ea44fe8e7fc19000002" // 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",
          "fistName": "Joe",
          "lastName": "Bloggs",
          "phone": "555-123456"
        },
        "guid": "4e563ea44fe8e7fc19000002",
        "type": "myFirstEntity"
      }
    */
  }
});
```

Update an entire entry

```
// 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": "4e563ea44fe8e7fc19000002", // Row/Entry ID
  "fields": {
    "fistName": "Jane"
  }
};
$fh.db(options, function (err, data) {
  if (err) {
    console.error("Error " + err);
  } else {
    console.log(JSON.stringify(data));
    /* Output:
      {
        "fields": {
          "fistName": "Jane"   //only one field now
        },
        "guid": "4e563ea44fe8e7fc19000002",
        "type": "myFirstEntity"
      }
    */
  }
});
```

Update a single field

```
var options = {
  "act": "read",
  "type": "myFirstEntity", // Entity/Collection name
  "guid": "4e563ea44fe8e7fc19000002" // Row/Entry ID
};
$fh.db(options, function (err, entity) {
  var entFields = entity.fields;
  entFields.firstName = 'Jane';

  options = {
    "act": "update",
    "type": "myFirstEntity",
    "guid": "4e563ea44fe8e7fc19000002",
    "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"
        }
      */
    }
  });
});
```

Delete an entry (row)

```
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"
      }
    */
  }
});
```

Delete all entity (collection) entries

```
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
      }
    */
  }
});
```

List

```
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"
      }]
    }
    */
  }
});
```

List with sorting

```
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
  }
};
```

List with pagination

```
var options = {
  "act": "list",
  "type": "myFirstEntity", // Entity/Collection name
  "skip": 20,    1
  "limit": 10    2
};
$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",
          "fistName": "Joe",
          "lastName": "Bloggs",
          "phone": "555-123456"
        },
        "guid": "4e563ea44fe8e7fc19000002",
        "type": "myFirstEntity"
      }, ...
      ]
    }
    */
  }
});
```

1

returns the third page of results

2

the size of the returned page is 10

List with Geo search

```
var options = {
  act: "list",
```

```
  type: "collectionName", // Entity/Collection name
  "geo": {
    "employeeLocation": { //emplyeeLocation 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",
          "fistName": "Joe",
          "lastName": "Bloggs",
          "phone": "555-123456"
        },
        "guid": "4e563ea44fe8e7fc19000002",
        "type": "myFirstEntity"
      }]
    }
    */
  }
});
```

List with multiple restrictions

```
/* 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",
          "fistName": "Joe",
          "lastName": "Bloggs",
          "phone": "555-123456"
        },
        "guid": "4e563ea44fe8e7fc19000002",
        "type": "myFirstEntity"
      }]
    }
    */
  }
});
```

List with restricted fields returned

```
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": "4e563ea44fe8e7fc19000002",
        "type": "myFirstEntity"
      }]
    }
    */
  }
});
```

Adding an index

```
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"
      }
    */
  }
});
```

CHAPTER 3. $FH.FORMS

3.1. $FH.FORMS.GETFORMS

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

Return an array of JSON objects with form summary information.

Note

These form summary JSON objects do not contain the full form definition. Use the $fh.forms.getForm function to get a full form definition.

3.1.1. Example

```
//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. Details

Retrieve specific form model based on form ID. Form IDs can be obtained using the $fh.forms.getForms function.

3.2.2. Example

```
$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. Details

Retrieve form models based on a list of form IDs.

3.3.2. Example

```
$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. Details

List all Submissions based on filtering criteria

3.4.2. submissionsObject

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

3.4.3. Example

```
$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. Details

Get A Single Form Submission

3.5.2. Example

```
$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. Details

Stream a single file contained in a submission. Files are accessed using the fileGroupId field in a submission file field.

3.6.2. fileStreamObject

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

3.6.3. Example

```
$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. FORM JSON DEFINITION

A form JSON object contains all of the information needed to process a form.

```
{
   "_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. Page

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

3.7.2. Field

```
{
  "_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. Page Rule

This JSON object describes a Page Rule created in the Studio.

```
{
   "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. Field Rule

This JSON object describes a Field Rule created in the Studio.

```
{
   "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. Details

Loads a JSON object representing the Theme that is assigned to the Project.

3.8.2. Example

```
//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. Details

Returns a JSON object containing Client Config settings associated with the Project. These are configured in the Studio.

3.8.5. Example

```
//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. Client Config JSON Object

```
{
  "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. Details

Submits a JSON object representing a Form Submission.

3.9.1.1. Example

```
var options = {
  "submission": <<Submission JSON Object>>,
  "appClientId": <<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. Submission JSON Object

```
{
  "formId": "<<ID Of Form Submitting Agains>>",
  "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. Field Entry JSON Object

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

3.9.4. Field Value Entries

This presents the data format required for each type of field submission.

- Text: String
- TextArea: String
- Number: Number
- Radio: String (Must represent one of the Radio Field options defined in the Form)
- Dropdown: String (Must represent one of the Dropdown options represented in the Form)
- Webstite: String
- Email: String (Must be a valid email format)
- DateStamp - Date Only: String (Format: **`DD/MM/YYYY`**)
- DateStamp - Time Only: String (Format: **`HH:SS`**)
- DateStamp - Date & Time: String (Format: **`YYYY-MM-DD HH:SS`**)

Check boxes

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

Location (And Map Field) - Latitude & Longitude

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

Location - Northings & Eastings

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

File Based Fields - File, Photo, Signature

```
{
  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 identifer for the fole.
NOTE: Must begin with "filePlaceHolder"
}
```

Note

All **hashName** parameters must begin with **`filePlaceHolder`** or the submission will be rejected.

3.10. $FH.FORMS.GETSUBMISSIONSTATUS

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

3.10.1. Details

Returns the current status of the submission.

3.10.2. Example

```
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. Submission Status JSON Object

Note

A submission is only marked as complete after the $fh.forms.completeSubmission function has been called. Therefore it is possible that the **`pendingFiles`** array can be empty and the **`status`** set as pending.

```
{
  "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. Details

Uploads a file to the submission.

Note

The file must already be added to the Submission JSON Object and submitted using the $fh.forms.submitFormData function.

Note

The file must already exist on the local file system to upload it to the submission.

Warning

If the **`keepFile`** parameter is not set to **`true`**, the file uploaded to the submission will be deleted from the file system when upload is complete.

3.11.2. Example

```
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 Object

```
{
  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. Details

Mark the submission as complete. This will check that all of the files submitted as part of the Submission JSON have been uploaded.

If the submission has completed successfully, the **`completeResult`** JSON object will contain

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

If submitted files have not been uploaded the **`completeResult`** JSON object will contain

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

3.12.2. Example

```
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. Details

The $fh.forms.createSubmissionModel function is an alternative method of creating and submitting a form.

The Submission Model provides some convenience functions to make the process of creating a submission easier.

3.13.2. Example

```
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

Note

The version of **`fh-mbaas-api`** in your **`package.json`** file must be at least **`4.8.0`**.

3.14.1. Details

The **`$fh.forms.registerListener`** function allows you to register an EventEmitter object to listen for submission events that occur.

Note

The **$fh.forms.registerListener** and **$fh.forms.deregisterListener** functions only accept EventEmitter objects as parameters.

```
//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. Event: submissionStarted

This event is emitted whenever a submission has been submitted, validated and saved to the database. This occurs **before** any files are uploaded as part of the submission.

The object passed to the **submissionStarted** event contains the following parameters:

```
{
    "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. Event: submissionComplete

This event is emitted whenever a submission has been submitted, has been validated and saved to the database, all files have been saved to the database and the submission has been verified. The

submission is now ready for processing using the $fh.forms.getSubmission Cloud API.

```
//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.
});
```

The **params** object passed to the event contains:

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

3.14.4. Event: submissionError

This event is emitted whenever an error has occurred when making a submission.

```
//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. Submission Error Types

Submission errors fall into different types depending on the reason for the error.

3.14.4.1.1. Validation Error

The submitted data is not valid. The response will be in the following format:

Note

For repeating fields, the error messages will be in the same order as the values entered for the field.

```
{
  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. Other Errors Saving The JSON Definition Of The Submission

There was an unexpected error saving the JSON definition of the submission. This event covers all errors other than validation that can occur when attempting to save the submission (For example, an error occurred when saving the submission to the database).

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

3.14.4.1.3. File Upload Error

There was an error uploading a file for a submission.

```
{
  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

Note

The version of **`fh-mbaas-api`** in your **`package.json`** file must be at least **`4.8.0`**.

3.15.1. Details

Removes a listener from the $fh.forms Cloud API.

```
//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

Note

The version of **`fh-mbaas-api`** in your **`package.json`** file must be at least **`5.10.0`**.

3.16.1. Details

Export CSV files from the $fh.forms Cloud API. The export returns a zip file of several CSV files. To filter then use the input value, you can filter by:

- *projectId:* The GUID of a project to filter by.
- *submissionId:* A single Submission ID or an array of submission IDs to filter by.
- *formId:* A single Form ID or and array of form IDs to filter by.
- *fieldHeader:* Header text to use in the exported CSV files. The options are *name* for the name of the field or *fieldCode* to use the field code defined in the Form Builder.

```
// 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

Note

The version of **`fh-mbaas-api`** in your **`package.json`** file must be at least **`5.12.0`**.

Note

This API is compatible with MBaaS version >=4.1.0 .

3.17.1. Details

Export one PDF file for a given submission from the $fh.forms Cloud API. The export returns a stream file.

```
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();
});
```

CHAPTER 4. $FH.HASH

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

Generate the hash value of a given input.

4.1. EXAMPLE

```
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);
  }
});
```

CHAPTER 5. $FH.HOST

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

Fetch the public host name of the MBaaS. Useful for configuring callback URLs in various authentication libraries.

5.1. EXAMPLE

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

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

CHAPTER 6. $FH.PUSH

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

Send a push message from the cloud to registered clients.

6.1. EXAMPLE

Push a message to all devices in all Client Apps of the associated project

```
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);
    }
  });
```

Push a message for specific deviceType in a specific Client App

```
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. PARAMETERS

6.2.1. Notification

- **message** Object

- **`alert`** String — The main message
- **`sound`** String — (iOS only) The name of a sound file in the app bundle, or **`default`**
- **`badge`** String — The number to display as the badge of the app icon
- **`userData`** Object — Any extra user data to be passed

6.2.2. iOS-specific

- **`message.apns`** Object — Options specific to the Apple Push Notification Service
  - **`title`** String — A short string describing the purpose of the notification
  - **`action`** String — The label of the action button
  - **`urlArgs`** Array — (Safari only) Values that are paired with the placeholders inside the urlFormatString value of your website.json file
  - **`titleLocKey`** String — (iOS only) The key to a title string in the Localizable.strings file for the current localization
  - **`titleLocArgs`** Array — (iOS only) Variable string values to appear in place of the format specifiers in title-loc-key
  - **`actionCategory`** String — The identifier of the action category for the interactive notification
  - **`contentAvailable`** Number — (iOS only) Informs the application that new content is available by delivering a silent notification. The only possible value is **`1`**.

6.2.3. Windows-specific

- **`message.windows`** Object — Options specific to the Windows platform (*Windows Notification Service* and *Microsoft Push Notification Service*)
  - **`type`** String — The type of message to send. Possible values: **`toast`**, **`raw`**, **`badge`** or **`tile`**.
  - **`duration`** String — Duration a Toast message is displayed. Possible values: **`long`** or **`short`**.
  - **`badge`** String — The glyph to use as the notification badge, instead of a number. Possible values: **`none`**, **`activity`**, **`alert`**, **`available`**, **`away`**, **`busy`**, **`newMessage`**, **`paused`**, **`playing`**, **`unavailable`**, **`error`** or **`attention`**. For more information on badge types, see official documentation. For numeric values, use the **`message.badge`** parameter.
  - **`tileType`** String — The tile template, for example, **`TileSquareText02`** or **`TileWideBlockAndText02`**. See the tile template catalog for all possible values.
  - **`images`** Array — List of images displayed on tiles. Either a path to a local file (for example, **`Assets/image.png`**) or a URL (for example, **`http://host/image.png`**). The number of elements needs to match the number of images required by the chosen tileType.
  - **`textFields`** Array — List of texts displayed on tiles. The number of elements needs to match the number of text fields required by the chosen tileType.

6.2.4. Other configuration

- **`options`** Object
- **`options.config`** Object
  - **`ttl`** Number — (APNS and GCM only) The time to live in seconds.

6.2.5. Selection of Client Apps in project

> **Warning**
>
> One of the options — **`broadcast`** or **`apps`** — must be set manually, there is no default value.

- **`options.broadcast`** Boolean — when set to **`true`**, notification will be sent to all Client Apps in the project which contains the sending Cloud App
- **`options.apps`** Array — list of Client App IDs to send the notification to

6.2.6. Filtering recipients

- **`options.criteria`** Object — Criteria for selection of notification recipients.
  See Sending Notifications for details about these criteria.
  - **`alias`** Array — list of user-specific identifiers
  - **`categories`** Array — list of categories
  - **`deviceType`** Array — list of device types
  - **`variants`** Array — list of variant IDs

6.2.7. Response handling

- **`callback(err, res)`** Function — callback invoked after the message is pushed to the integrated push server. If **`err`** is set, it contains any possible error response. Parameter **`res`** contains the normal server response.

CHAPTER 7. $FH.SEC

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

Key generation, data encryption and decryption.

7.1. EXAMPLE

RSA Key Generation

```
$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 Encryption

```
$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 Decryption

```
$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 Key Generation

```
$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 Encryption

```
$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 Decryption

```
$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);
});
```

CHAPTER 8. $FH.SERVICE

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

Call an endpoint in an MBaaS Service.

Minimum Requirements

- fh-mbaas-api : v4.9.1

8.1. EXAMPLE

```
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);
  }
});
```

CHAPTER 9. $FH.STATS

Utilise temporary stats counters and timers, which can then be viewed as graphs in the Studio.

9.1. EXAMPLE

```
// 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);
```

CHAPTER 10. $FH.SYNC

The cloud sync framework requires handler functions to be defined which provide access to the back end data & manage data collisions. These are specified in the handleXXX() functions. Each unique dataset being managed is identified by a dataset_id which must be specified as the first parameter when calling any function of the sync framework.

Note

Default implementations of the handler functions are already provided as part of the MBaaS service. They use hosted db service to save data (via $fh.db). If this is enough for your app, you don't need to implement the handler functions anymore, you just need to call the init function and provide sync options.

However, if the default implementations do not meet your app's requirement (e.g you need to save data to somewhere else), you need to provide your own implementations of the handler functions listed below. You can provide implementations for all the CRUDL operations, or you can change the default behaviour of a particular operation by only providing an override for the corresponding handler function (e.g provide an override **`$fh.sync.handleRead`** function will allow you to implement your own read data logic, but keep the rest operations to still use the default MBaaS implementations).

10.1. $FH.SYNC.INIT

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

Initialise cloud data sync service for specified dataset.

10.1.1. Example

```
// Unique Id for the dataset to initialise.
var dataset_id = 'tasks';
// Configuration options
var options = {
  "sync_frequency": 10, // How often to synchronise data with the back
end data store in seconds. Default: 10s
  "logLevel":"info" // The level of logging. Can be usful 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
    });
  }
});
```

10.2. $FH.SYNC.INVOKE

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

Invoke the Sync Server.

10.2.1. Example

```
// 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
});
```

10.3. $FH.SYNC.STOP

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

Stop cloud data sync for the specified dataset_id.

10.3.1. Example

```
// 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.
});
```

10.4. $FH.SYNC.STOPALL

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

Stop cloud data sync service for ALL datasets.

10.4.1. Example

```
// 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);
});
```

10.5. $FH.SYNC.HANDLELIST

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

Defines a handler function for listing data from the back end data source for a dataset. Should be called after the dataset is initialised.

10.5.1. Example

```
$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"
  }
}
*/
```

10.6. $FH.SYNC.GLOBALHANDLELIST

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

Similar to $fh.sync.handleList, but set the handler globally which means the same handler function can be used by multiple datasets. The global handler will only be used if there is no handler assigned to the dataset via $fh.sync.handleList.

10.6.1. Example

```
$fh.sync.globalHandleList(function(dataset_id, params, cb, meta_data){
```

```
  //list data for the specified dataset_id
});
```

10.7. $FH.SYNC.HANDLECREATE

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

Defines a handler function for creating a single row of data in the back end. Should be called after the dataset is inited.

10.7.1. Example

```
// 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);
});
```

10.8. $FH.SYNC.GLOBALHANDLECREATE

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

Similar to $fh.sync.handleCreate, but set the handler globally which means the same handler function can be used by multiple datasets. The global handler will only be used if there is no handler assigned to the dataset via $fh.sync.handleCreate.

10.8.1. Example

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

10.9. $FH.SYNC.HANDLEREAD

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

Defines a handler function for reading a single row of data from the back end. Should be called after the dataset is initialised.

10.9.1. Example

```
// 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);
});
```

10.10. $FH.SYNC.GLOBALHANDLEREAD

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

Similar to $fh.sync.handleRead, but set the handler globally which means the same handler function can be used by multiple datasets. The global handler will only be used if there is no handler assigned to the dataset via $fh.sync.handleRead.

10.10.1. Example

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

10.11. $FH.SYNC.HANDLEUPDATE

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

Defines a handler function for updating a single row of data from the back end. Should be called after the dataset is initialised.

10.11.1. Example

```
// 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);
});
```

10.12. $FH.SYNC.GLOBALHANDLEUPDATE

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

Similar to $fh.sync.handleUpdate, but set the handler globally which means the same handler function can be used by multiple datasets. The global handler will only be used if there is no handler assigned to the dataset via $fh.sync.handleUpdate.

10.12.1. Example

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

10.13. $FH.SYNC.HANDLEDELETE

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

Defines a handler function for deleting a single row of data from the back end. Should be called after the dataset is initialised.

10.13.1. Example

```
// 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);
});
```

10.14. $FH.SYNC.GLOBALHANDLEDELETE

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

Similar to $fh.sync.handleDelete, but set the handler globally which means the same handler function can be used by multiple datasets. The global handler will only be used if there is no handler assigned to the dataset via $fh.sync.handleDelete.

10.14.1. Example

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

10.15. $FH.SYNC.HANDLECOLLISION

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

Defines a handler function for dealing with data collisions (that is, stale updates). Should be called after the dataset is initialised.

10.15.1. Example

```
// 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);
});
```

10.16. $FH.SYNC.GLOBALHANDLECOLLISION

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

Similar to $fh.sync.handleCollision, but set the handler globally which means the same handler function can be used by multiple datasets. The global handler will only be used if there is no handler assigned to the dataset via $fh.sync.handleCollision.

10.16.1. Example

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

});
```

10.17. $FH.SYNC.LISTCOLLISIONS

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

Defines a handler function for returning the current list of collisions. Should be called after the dataset is initialised.

10.17.1. Example

```
// 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);
});
```

Note

"collision_hash" is the hash value which uniquely identifies a collision. This value will have been passed as the "hash" parameter when the collision was originally created in the "handleCollision" function.

10.18. $FH.SYNC.GLOBALLISTCOLLISIONS

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

Similar to $fh.sync.listCollisions, but set the handler globally which means the same handler function can be used by multiple datasets. The global handler will only be used if there is no handler assigned to the dataset via $fh.sync.listCollisions.

10.18.1. Example

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

});
```

10.19. $FH.SYNC.REMOVECOLLISION

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

Defines a handler function for deleting a collision from the collisions list. Should be called after the dataset is initialised.

10.19.1. Example

```
// 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);
});
```

10.20. $FH.SYNC.GLOBALREMOVECOLLISION

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

Similar to $fh.sync.removeCollision, but set the handler globally which means the same handler function can be used by multiple datasets. The global handler will only be used if there is no handler assigned to the dataset via $fh.sync.removeCollision.

10.20.1. Example

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

});
```

10.21. $FH.SYNC.INTERCEPTREQUEST

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

Intercept the sync requests for the specified dataset. Can be useful for checking client identities and validating authentication.

10.21.1. Example

```
$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'});
  }
});
```

10.22. $FH.SYNC.GLOBALINTERCEPTREQUEST

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

Similar to $fh.sync.interceptRequest, but set the handler globally which means the same handler function can be used by multiple datasets. The global handler will only be used if there is no handler assigned to the dataset via $fh.sync.interceptRequest.

10.22.1. Example

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

});
```