

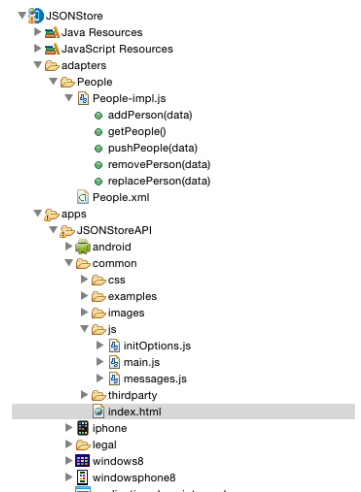
Using JSONStore in Hybrid applications

Overview

This tutorial is a continuation of the JSONStore Overview tutorial.

The tutorial covers the following topics:

- Add JSONStore Feature
- Basic Usage
 - Initialize
 - Get
 - Add
 - Find
 - Replace
 - Remove
 - Remove Collection
 - Destroy
- Advanced Usage
 - Security
 - Multiple User Support
 - MobileFirst Adapter Integration
 - Enhance
- Sample application
- For more information



Add JSONStore Feature

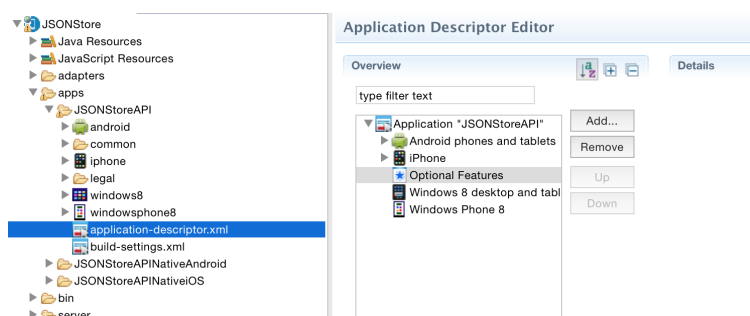
To add JSONStore to your hybrid environment open the `application-descriptor.xml` simply add under the element.

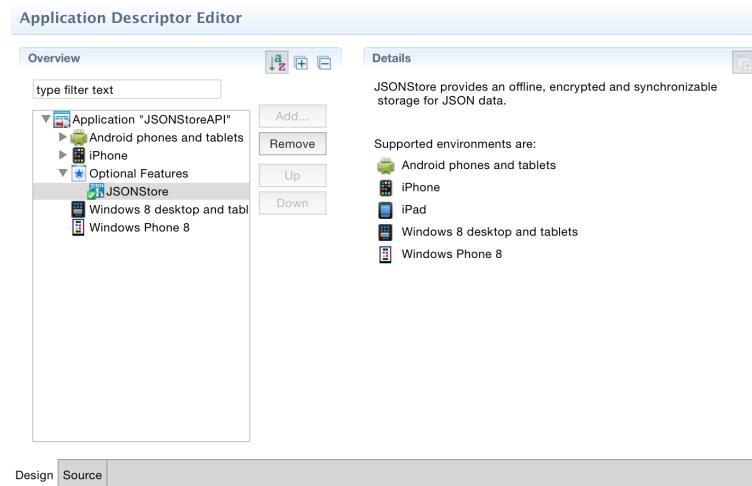
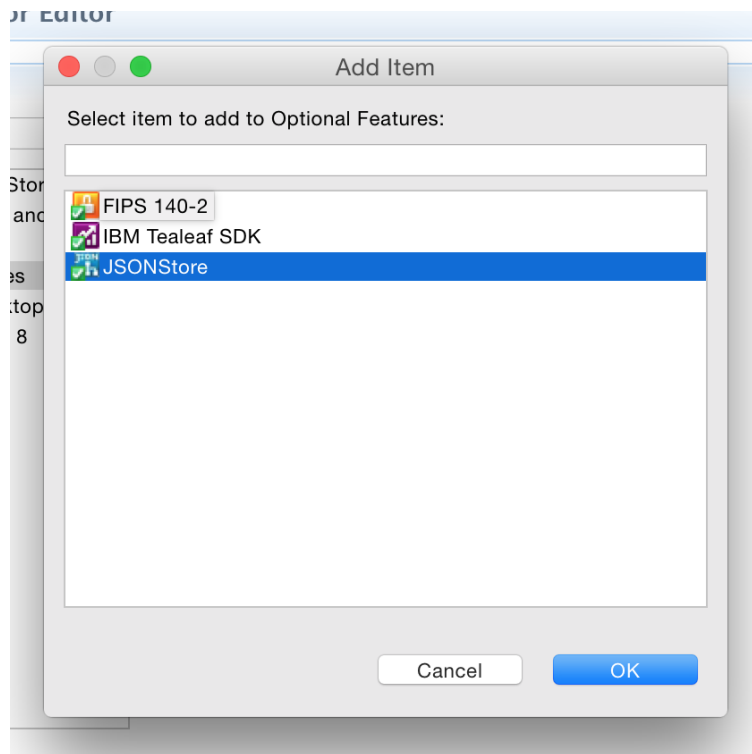
```

1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <application xmlns="http://www.worklight.com/application-descriptor" id="JSONStoreAPI" platformVersion
3   <displayName>JSONStoreAPI</displayName>
4   <description>JSONStoreAPI</description>
5   <author>
6     <name>application's author</name>
7     <email>application author's e-mail</email>
8     <homepage>http://mycompany.com</homepage>
9     <copyright>Copyright My Company</copyright>
10  </author>
11  <mainFile>index.html</mainFile>
12  <features>
13    <JSONStore/>
14  </features>
15  <thumbnailImage>common/images/thumbnail.png</thumbnailImage>
16  <iphone bundleId="com.JSONStoreAPI" version="1.0">
17    <worklightSettings include="false"/>
18    <security>
19      <encryptWebResources enabled="false"/>
20      <testWebResourcesChecksum enabled="false" ignoreFileExtensions="png, jpg, jpeg, gif, mp4, mp
21    </security>
22  </iphone>
23  <android version="1.0">
24    <worklightSettings include="false"/>
25    <security>
26      <encryptWebResources enabled="false"/>
27      <testWebResourcesChecksum enabled="false" ignoreFileExtensions="png, jpg, jpeg, gif, mp4, mp
28      <publicSigningKey>Replace this text with the actual public signing key of the certificate used to sig
29      <packageName>Replace this text with the actual package name of the application, which is the va
30    </security>
31  </android>
32  <windowsPhone8 version="1.0">
33    <uuid>e5eeea5c-4c80-40d4-b250-c8f2e8698138</uuid>
34  </windowsPhone8>
35  <windows8 version="1.0">
36    <uuid>802f8287-a3f7-4dc5-ac17-1da638074763</uuid>
37  </windows8>
38 </application>

```

Alternatively, you can use the **Application Descriptor Editor** click **Optional Features > Add > JSONStore > OK**





Initialize

Use `init` to start one or more JSONStore collections

Starting or provisioning a collections means creating the persistent storage that contains the collection and documents, if it does not exists.

If the persistent storage is encrypted and a correct password is passed, the necessary security procedures to make the data accessible are run.

For optional features that you can enable at initialization time, see **Security**, **Multiple User Support**, and **MobileFirst Adapter Integration** in the second part of this module

```

1  var collections = {
2    people : {
3      searchFields: {name: 'string', age: 'integer'}
4    }
5  };
6
7  WL.JSONStore.init(collections).then(function (collections) {
8    // handle success - collection.people (people's collection)
9  }).fail(function (error) {
10    // handle failure
11  });

```

Get

Use `get` to create an accessor to the collection. You must call `init` before you call `get` otherwise the result of `get` is undefined

```

1  var collectionName = 'people';
2  var people = WL.JSONStore.get(collectionName);

```

The variable `people` can now be used to perform operations on the `people` collection such as `add`, `find`, and `replace`

Add

Use `add` to store data as documents inside a collection

```

1  var collectionName = 'people';
2  var options = {};
3  var data = {name: 'yoel', age: 23};
4  WL.JSONStore.get(collectionName).add(data, options).then(function () {
5    // handle success
6  }).fail(function (error) {
7    // handle failure
8  });

```

Find

Use `find` to locate a document inside a collection by using a query. Use `findAll` to retrieve all the documents inside a collection. Use `findById` to search by the document unique identifier. The default behavior for `find` is to do a "fuzzy" search

```

1  var query = {name: 'yoel'};
2  var collectionName = 'people';
3  var options = {
4      exact: false, //default
5      limit: 10 // returns a maximum of 10 documents, default: return every document
6  };
7
8  WL.JSONStore.get(collectionName).find(query, options).then(function (results) {
9      // handle success - results (array of documents found)
10 })fail(function (error) {
11     // handle failure
12 });
13 <p>

```

Replace

Use `replace` to modify documents inside a collection. The field that you use to perform the replacement is `_id`, the document unique identifier.

```

1  var document = {
2      _id: 1, json: {name: 'chevy', age: 23}
3  };
4  var collectionName = 'people';
5  var options = {};
6  WL.JSONStore.get(collectionName).replace(document, options).then(function (numberOfDocsReplaced)
7      // handle success
8  }).fail(function (error) {
9      // handle failure
10 });
11 <p>

```

This examples assumes that the document `{_id: 1, json: {name: 'yoel', age: 23}}` is in the collection

Remove

Use `remove` to delete a document from a collection

Documents are not erased from the collection until you call `push`. For more information, see the **MobileFirst Adapter Integration** section later in this tutorial

```

1  var query = { _id: 1 };
2  var collectionName = 'people';
3  var options = { exact: true };
4  WL.JSONStore.get(collectionName).remove(query, options).then(function (numberOfDocsRemoved) {
5      // handle success
6  }).fail(function (error) {
7      // handle failure
8  });

```

Remove Collection

Use `removeCollection` to delete all the documents that are stored inside a collection. This operation is similar to dropping a table in database terms

```

1  var collectionName = 'people';
2  WL.JSONStore.get(collectionName).removeCollection().then(function (removeCollectionReturnCode) {
3      // handle success
4  }).fail(function (error) {
5      // handle failure
6  });

```

Destroy

Use `destroy` to remove the following data:

- All documents
- All collections
- All Stores (see "**Multiple User Support**" later in this tutorial)
- All JSONStore metadata and security artifacts (see "**Security**" later in this tutorial)

```

1  var collectionName = 'people';
2  WL.JSONStore.destroy().then(function () {
3      // handle success
4  }).fail(function (error) {
5      // handle failure
6  });

```

Security

You can secure all the collections in a store by passing a password to the `init` function. If no password is passed, the documents of all the collections in the store are not encrypted.

Data encryption is only available on Android, iOS, Windows Phone 8, and Windows 8 environments.

Some security metadata are stored in the keychain (iOS), shared preferences (Android), isolated storage (Windows 8 Phone), or the credential locker (Windows 8).

The store is encrypted with a 256-bit Advanced Encryption Standard (AES) key. All keys are strengthened with Password-Based Key Derivation Function 2 (PBKDF2).

Use `closeAll` to lock access to all the collections until you call `init` again. If you think of `init` as a login function you can think of `closeAll` as the corresponding logout function.

Use `changePassword` to change the password.

```
1  var collections = {
2    people: {
3      searchFields: {name: 'string'}
4    }
5  };
6
7  var options = {password: '123'};
8  WL.JSONStore.init(collections, options).then(function () {
9    // handle success
10  }).fail(function (error) {
11    // handle failure
12  });
```

Multiple User Support

You can create multiple stores that contain different collections in a single MobileFirst application. The `init` function can take an options object with a username. If no username is given, the default username is **jsonstore**

```
1  var collections = {
2    people: {
3      searchFields: {name: 'string'}
4    }
5  };
6
7  var options = {username: 'yoel'};
8  WL.JSONStore.init(collections, options).then(function () {
9    // handle success
10  }).fail(function (error) {
11    // handle failure
12  });
```

MobileFirst Adapter Integration

This section assumes that you are familiar with MobileFirst adapters. MobileFirst Adapter Integration is optional and provides ways to send data from a collection to an adapter and get data from an adapter into a collection.

You can achieve these goals by using functions such as `WL.Client.invokeProcedure` or `jQuery.ajax` if you need more flexibility.

Adapter Implementation

Create a MobileFirst adapter and name it **'People'**. Define its procedures `addPerson`, `getPeople`, `pushPeople`, `removePerson`, and `replacePerson`.

```
1  function getPeople() {
2      var data = { peopleList : [{name: 'chevy', age: 23}, {name: 'yoel', age: 23}] };
3      WL.Logger.debug('Adapter: people, procedure: getPeople called.');
```

4 WL.Logger.debug('Sending data: ' + JSON.stringify(data));

```
5      return data;
6  }
7
8  function pushPeople(data) {
9      WL.Logger.debug('Adapter: people, procedure: pushPeople called.');
```

10 WL.Logger.debug('Got data from JSONStore to ADD: ' + data);

```
11     return;
12 }
13
14 function addPerson(data) {
15     WL.Logger.debug('Adapter: people, procedure: addPerson called.');
```

16 WL.Logger.debug('Got data from JSONStore to ADD: ' + data);

```
17     return;
18 }
19
20 function removePerson(data) {
21     WL.Logger.debug('Adapter: people, procedure: removePerson called.');
```

22 WL.Logger.debug('Got data from JSONStore to REMOVE: ' + data);

```
23     return;
24 }
25 function replacePerson(data) {
26     WL.Logger.debug('Adapter: people, procedure: replacePerson called.');
```

27 WL.Logger.debug('Got data from JSONStore to REPLACE: ' + data);

```
28     return;
29 }
```

Initialize a collection linked to a MobileFirst adapter


```

1  var collections = {
2    people : {
3      searchFields : {name: 'string', age: 'integer'},
4      adapter : {
5        name: 'People',
6        add: 'addPerson',
7        remove: 'removePerson',
8        replace: 'replacePerson',
9        load: {
10         procedure: 'getPeople',
11         params: [],
12         key: 'peopleList'
13       }
14     }
15   }
16 }</p>
17 <p>var options = {};</p>
18 <p>WL.JSONStore.init(collections, options).then(function () {
19   // handle success
20 }).fail(function (error) {
21   // handle failure
22 });

```

Load data from MobileFirst Adapter

When `load` is called, JSONStore uses some metadata about the adapter (**name** and **procedure**), which you previously passed to `init`, to determine what data to get from the adapter and eventually store it.

```

1  var collectionName = 'people';
2  <p>WL.JSONStore.get(collectionName).load().then(function (loadedDocuments) {
3    // handle success
4  }).fail(function (error) {
5    // handle failure
6  });

```

Get Push Required (Dirty Documents)

Calling `getPushRequired` returns an array of so called "dirty documents", which are documents that have local modifications that do not exist on the back-end system. These documents are sent to the MobileFirst adapter when `push` is called.

```

1   var collectionName = 'people';
2   <p>WL.JSONStore.get(collectionName).getPushRequired().then(function (dirtyDocuments) {
3       // handle success
4   }).fail(function (error) {
5       // handle failure
6   });

```

To prevent JSONStore from marking the documents as "dirty", pass the option {markDirty:false} to add, replace, and remove

Push

push sends the documents that changed to the correct MobileFirst adapter procedure (i.e., addPerson is called with a document that was added locally). This mechanism is based on the last operation that is associated with the document that changed and the adapter metadata that is passed to init.

```

1   var collectionName = 'people';
2   <p>WL.JSONStore.get(collectionName).push().then(function (response) {
3       // handle success
4       // response is an empty array if all documents reached the server
5       // response is an array of error responses if some documents failed to reach the server
6   }).fail(function (error) {
7       // handle failure
8   });

```

Enhance

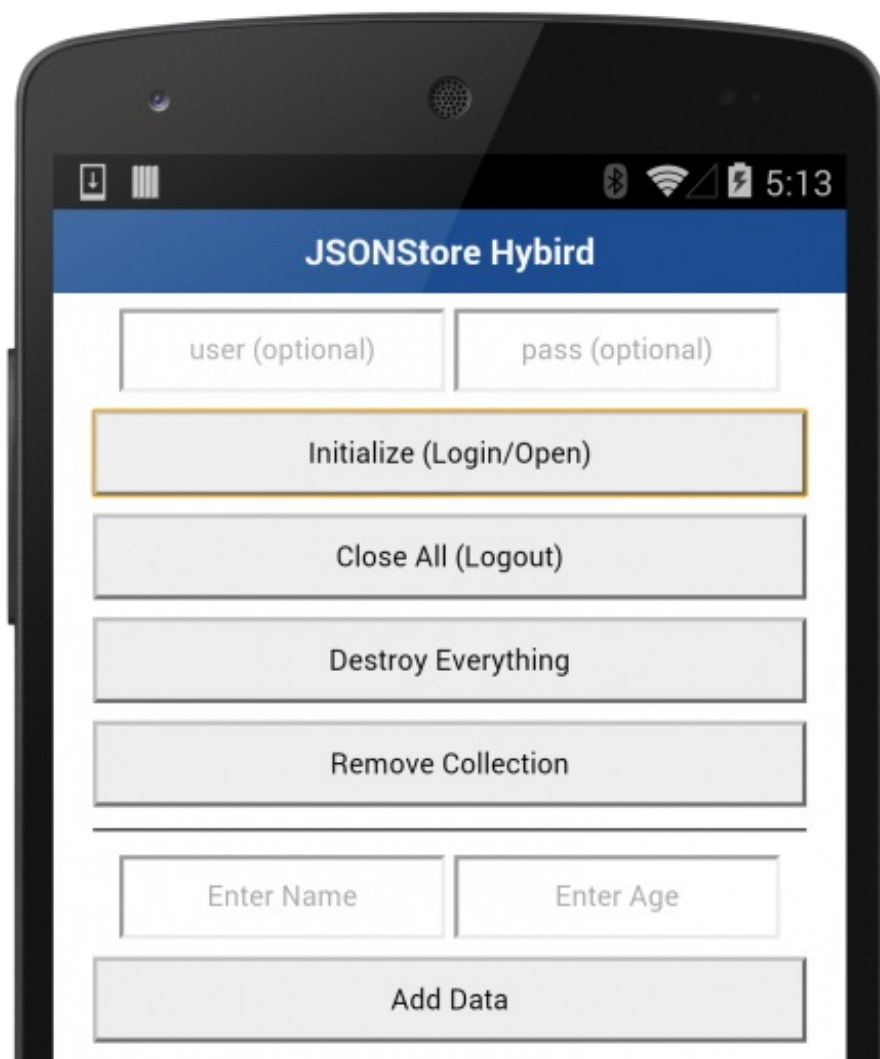
Use enhance to extend the core API to fit your needs, by adding functions to a collection prototype.

This example shows how to use enhance to add the function getVaLue that works on the keyvaLue collection. It takes a key (string) as it's only parameter and returns a single result.

```

1  var collectionName = 'keyvalue';
2
3  WL.JSONStore.get(collectionName).enhance('getValue', function (key) {
4      var deferred = $.Deferred();
5      var collection = this;
6
7      //Do an exact search for the key
8      collection.find({key: key}, {exact:true, limit: 1}).then(deferred.resolve, deferred.reject);
9      return deferred.promise();
10 });
11
12 //Usage:
13 var key = 'myKey';
14 WL.JSONStore.get(collectionName).getValue(key).then(function (result) {
15     // handle success
16     // result contains an array of documents with the results from the find
17 }).fail(function () {
18     // handle failure
19 });

```





Sample application

Click to download (<https://github.com/MobileFirst-Platform-Developer-Center/JSONStore>) the MobileFirst project.

The MobileFirst project contains an application that demonstrates the use of JSONStore in a hybrid environment.

For more information

For more information about JSONStore, see the product user documentation.