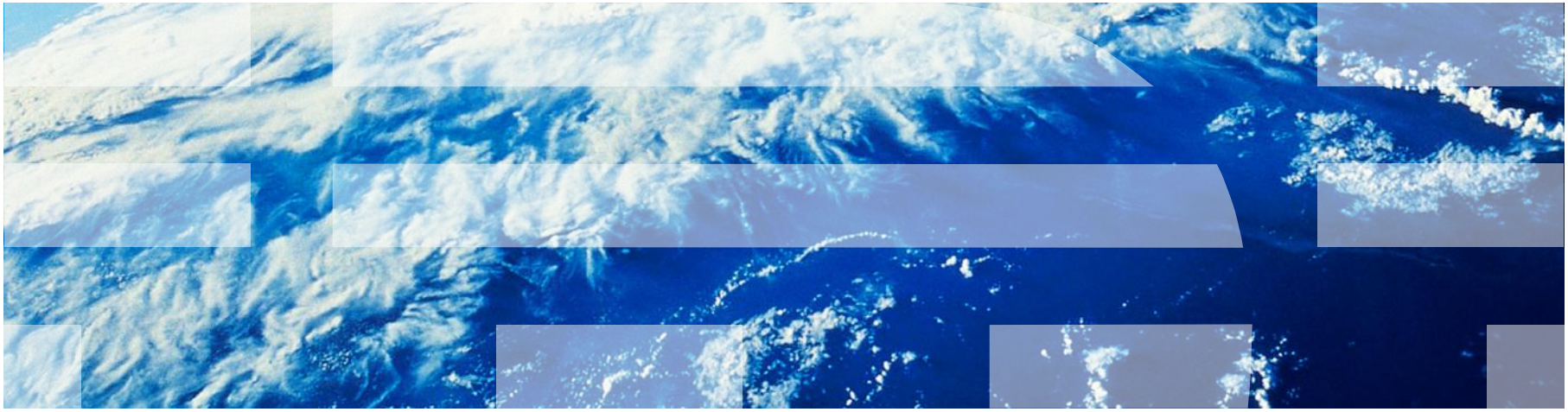
IBM Worklight Foundation V6.2.0
Getting Started

JSONStore – JavaScript API



5 Sep 2014

Trademarks

- IBM, the IBM logo, ibm.com, and Worklight are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.
- Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.
- Other company products or service names may be trademarks or service marks of others.
- This document may not be reproduced in whole or in part without the prior written permission of IBM.

About IBM®

- See http://www.ibm.com/ibm/us/en/

Agenda

- What is JSONStore?
- Follow along with the code sample
- Code sample walkthrough
- Expected output
- For more information

What is JSONStore?

- Optional client-side API with the following key features:
  - Data indexing for efficient searching.
  - Data encryption in production environments.
  - Mechanism for tracking local-only changes to the stored data.
  - Support for multiple users.
- Available in the following environments:
  - Native: Android and iOS
  - Hybrid: Android, iOS, Windows Phone 8, and Windows 8
  - Preview Common Resources / Mobile Browser Simulator (not for production use)

Note: This module shows how to get started with the JSONStore API. Some features such as data encryption are beyond the scope of this module. All features are documented in detail in the IBM Worklight® Foundation user documentation.

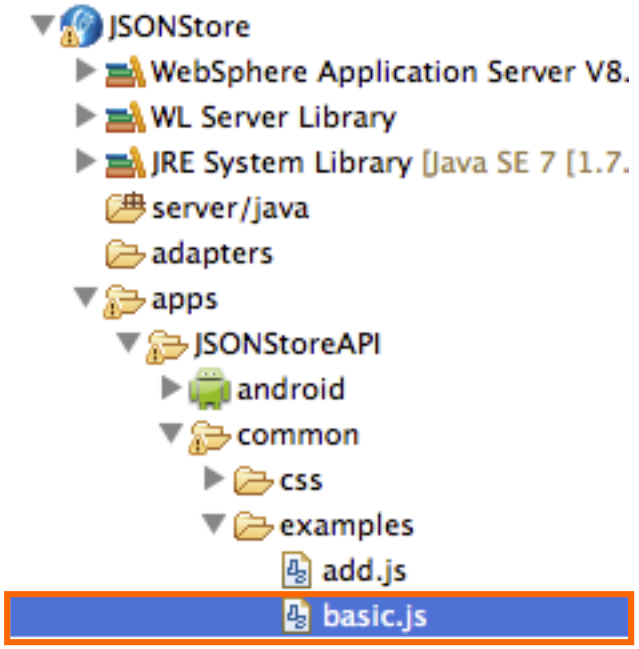
Agenda

- What is JSONStore?
- Follow along with the code sample
- Code sample walkthrough
- Expected output
- For more information

Follow along with the code sample

1. Download the compressed file with the code sample that is associated with this module.
2. Open the `basic.js` file. The sample image provides context.
3. Run the application on one of the environments listed (for example: Android, iPhone). The *"Setting up your environment"* and *"Hello Worklight"* Getting Started Modules cover this procedure in detail.

Note: The code sample uses a JavaScript™ Unit Test framework that is called QUnit (qunitjs.com). Explaining how it works is beyond the scope of this module.



Agenda

- What is JSONStore?
- Follow along with the code sample
- Code sample walkthrough
- Expected output
- For more information

Code sample walkthrough (1 of 9)

The `destroy` API removes all JSONStore content from the application. It is used here to start with no data. Doing it this way ensures that the output is predictable in the code sample.

```
WL.JSONStore.destroy()

.then(function () {

    var collections = {
            people : {
                searchFields: {name: 'string', age: 'integer'}
            }
    };

    return WL.JSONStore.init(collections);
})

.then(function () {

    var data = [{name: 'carlos', age: 20},
                {name: 'mike', age: 30}];

    return WL.JSONStore.get('people').add(data);
})

.then(function () {
    return WL.JSONStore.get('people').findAll();
})

.then(function (res) {

    deepEqual(res, [{_id: 1, json: {name: 'carlos', age: 20}},
                    {_id: 2, json: {name: 'mike', age: 30}}]);

    start();
})

.fail(function (err) {
    ok(false, 'Got failure: ' + err.toString());
    start();
});
```

Code sample walkthrough (2 of 9)

To persist data, you must first define at least one collection. These collections are entities that hold data. Shown here is the definition of a collection that is called `people`.

```
WL.JSONStore.destroy()

.then(function () {

    var collections = {
            people : {
                searchFields: {name: 'string', age: 'integer'}
            }
    };

    return WL.JSONStore.init(collections);
})

.then(function () {

    var data = [{name: 'carlos', age: 20},
                {name: 'mike', age: 30}];

    return WL.JSONStore.get('people').add(data);
})

.then(function () {
    return WL.JSONStore.get('people').findAll();
})

.then(function (res) {

    deepEqual(res, [{_id: 1, json: {name: 'carlos', age: 20}},
                    {_id: 2, json: {name: 'mike', age: 30}}]);

    start();
})

.fail(function (err) {
    ok(false, 'Got failure: ' + err.toString());
    start();
});
```

Code sample walkthrough (3 of 9)

Search fields are fields that are indexed inside a collection. You can use those fields when you search for data that is inside a collection.

You can see here the definition of two search fields:

- `name` (string)
- `age` (integer)

The data types, such as `string`, `integer`, `number`, `Boolean`, are used to better store input data.

```
WL.JSONStore.destroy()

.then(function () {

    var collections = {
            people : {
                searchFields: {name: 'string', age: 'integer'}
            }
    };

    return WL.JSONStore.init(collections);
})

.then(function () {

    var data = [{name: 'carlos', age: 20},
                {name: 'mike', age: 30}];

    return WL.JSONStore.get('people').add(data);
})

.then(function () {
    return WL.JSONStore.get('people').findAll();
})

.then(function (res) {

    deepEqual(res, [{_id: 1, json: {name: 'carlos', age: 20}},
                    {_id: 2, json: {name: 'mike', age: 30}}]);

    start();
})

.fail(function (err) {
    ok(false, 'Got failure: ' + err.toString());
    start();
});
```

Code sample walkthrough (4 of 9)

The `init` API is used to open one or more collections. If the collection was never opened before, a file is created on the file system to persist data that is inside the collection. Before the operation finishes, an accessor to that file is created.

The accessor allows the caller to call collection-level APIs such as `add` and `findAll`, which are shown later in this code sample walkthrough.

```
WL.JSONStore.destroy()

.then(function () {

    var collections = {
            people : {
                searchFields: {name: 'string', age: 'integer'}
            }
    };

    return WL.JSONStore.init(collections);
})

.then(function () {

    var data = [{name: 'carlos', age: 20},
                {name: 'mike', age: 30}];

    return WL.JSONStore.get('people').add(data);
})

.then(function () {
    return WL.JSONStore.get('people').findAll();
})

.then(function (res) {

    deepEqual(res, [{_id: 1, json: {name: 'carlos', age: 20}},
                    {_id: 2, json: {name: 'mike', age: 30}}]);

    start();
})

.fail(function (err) {
    ok(false, 'Got failure: ' + err.toString());
    start();
});
```

Code sample walkthrough (5 of 9)

The data that is stored inside the `people` collection is defined here. Notice that the data is a hardcoded array of two JSON objects with key value pairs for `name` and `age`. This data can be acquired from multiple sources (for example: Network Request, File I/O, User Input).

```
WL.JSONStore.destroy()

.then(function () {

    var collections = {
            people : {
                searchFields: {name: 'string', age: 'integer'}
            }
    };

    return WL.JSONStore.init(collections);
})

.then(function () {

    var data = [{name: 'carlos', age: 20},
                {name: 'mike', age: 30}];

    return WL.JSONStore.get('people').add(data);
})

.then(function () {
    return WL.JSONStore.get('people').findAll();
})

.then(function (res) {

    deepEqual(res, [{_id: 1, json: {name: 'carlos', age: 20}},
                    {_id: 2, json: {name: 'mike', age: 30}}]);

    start();
})

.fail(function (err) {
    ok(false, 'Got failure: ' + err.toString());
    start();
});
```

Code sample walkthrough (6 of 9)

The accessor that is created by the completion of the `init` API is accessed via the `get` API. That accessor provides access to store data inside the people collection. The input data must be in JSON format.

```
WL.JSONStore.destroy()

.then(function () {

    var collections = {
            people : {
                searchFields: {name: 'string', age: 'integer'}
            }
    };

    return WL.JSONStore.init(collections);
})

.then(function () {

    var data = [{name: 'carlos', age: 20},
                {name: 'mike', age: 30}];

    return WL.JSONStore.get('people').add(data);
})

.then(function () {
    return WL.JSONStore.get('people').findAll();
})

.then(function (res) {

    deepEqual(res, [{_id: 1, json: {name: 'carlos', age: 20}},
                    {_id: 2, json: {name: 'mike', age: 30}}]);

    start();
})

.fail(function (err) {
    ok(false, 'Got failure: ' + err.toString());
    start();
});
```

Code sample walkthrough (7 of 9)

You can find documents inside a JSONStore collection in different ways. For example: `find`, `findById`.

The easiest way, and the way that is shown here, is by using the `findAll` API. This method returns all the data that is stored inside a collection.

```
WL.JSONStore.destroy()

.then(function () {

    var collections = {
            people : {
                searchFields: {name: 'string', age: 'integer'}
            }
    };

    return WL.JSONStore.init(collections);
})

.then(function () {

    var data = [{name: 'carlos', age: 20},
                {name: 'mike', age: 30}];

    return WL.JSONStore.get('people').add(data);
})

.then(function () {
    return WL.JSONStore.get('people').findAll();
})

.then(function (res) {

    deepEqual(res, [{_id: 1, json: {name: 'carlos', age: 20}},
                    {_id: 2, json: {name: 'mike', age: 30}}]);

    start();
})

.fail(function (err) {
    ok(false, 'Got failure: ' + err.toString());
    start();
});
```

Code sample walkthrough (8 of 9)

Data that is stored inside a collection is called a document.

Documents have `_id` and `json` key value pairs. The `_id` pair is an internal identifier that is added automatically when data is added. The `json` pair contains all the data that was added.

```
WL.JSONStore.destroy()

.then(function () {

    var collections = {
            people : {
                searchFields: {name: 'string', age: 'integer'}
            }
    };

    return WL.JSONStore.init(collections);
})

.then(function () {

    var data = [{name: 'carlos', age: 20},
                {name: 'mike', age: 30}];

    return WL.JSONStore.get('people').add(data);
})

.then(function () {
    return WL.JSONStore.get('people').findAll();
})

.then(function (res) {

    deepEqual(res, [{_id: 1, json: {name: 'carlos', age: 20}},
                    {_id: 2, json: {name: 'mike', age: 30}}]);

    start();
})

.fail(function (err) {
    ok(false, 'Got failure: ' + err.toString());
    start();
});
```

Code sample walkthrough (9 of 9)

If one of the API calls fails (for example: `init`, `add`, `findAll`) the function inside `.fail` is called with an error object.

Make sure that the error object contains enough information for the user to understand what went wrong. For example, it can contain:

- `src` - source of the error, for example: `find`
- `msg` - message that is related to the error, for example: `BAD_PARAMETER_EXPECTED_STRING`

```
WL.JSONStore.destroy()

.then(function () {

    var collections = {
            people : {
                searchFields: {name: 'string', age: 'integer'}
            }
    };

    return WL.JSONStore.init(collections);
})

.then(function () {

    var data = [{name: 'carlos', age: 20},
                {name: 'mike', age: 30}];

    return WL.JSONStore.get('people').add(data);
})

.then(function () {
    return WL.JSONStore.get('people').findAll();
})

.then(function (res) {

    deepEqual(res, [{_id: 1, json: {name: 'carlos', age: 20}},
                    {_id: 2, json: {name: 'mike', age: 30}}]);

    start();
})

.fail(function (err) {
    ok(false, 'Got failure: ' + err.toString());
    start();
});
```

Agenda

- What is JSONStore?
- Follow along with the code sample
- Code sample walkthrough
- Expected output
- For more information

Expected Output

When the application is executed in one of the supported environments, the output looks similar to the sample image.

The green line indicates that everything is working as expected.

Agenda

- What is JSONStore?
- Follow along with the code sample
- Code sample walkthrough
- Expected output
- For more information

For more information

- For more information about JSONStore, see JSONStore in the product user documentation.
- For more information about JSONStore performance and best practices, see JSONStore performance in the product user documentation.s

Notices

- Permission for the use of these publications is granted subject to these terms and conditions.
- This information was developed for products and services offered in the U.S.A.
- IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.
- IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
  - IBM Director of Licensing
    IBM Corporation
    North Castle Drive
    Armonk, NY 10504-1785
    U.S.A.
- For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:
  - Intellectual Property Licensing
    Legal and Intellectual Property Law
    IBM Japan Ltd.
    1623-14, Shimotsuruma, Yamato-shi
    Kanagawa 242-8502 Japan
- **The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.
- This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.
- Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
- IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.
- Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:
  - IBM Corporation
    Dept F6, Bldg 1
    294 Route 100
    Somers NY 10589-3216
    USA
- Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.
- The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.
- Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

COPYRIGHT LICENSE:

- This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.
- Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:
  - © (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

Privacy Policy Considerations

- IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.
- Depending upon the configurations deployed, this Software Offering may use session cookies that collect session information (generated by the application server). These cookies contain no personally identifiable information and are required for session management. Additionally, persistent cookies may be randomly generated to recognize and manage anonymous users. These cookies also contain no personally identifiable information and are required.
- If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent. For more information about the use of various technologies, including cookies, for these purposes, see IBM’s Privacy Policy at http://www.ibm.com/privacy and IBM’s Online Privacy Statement at http://www.ibm.com/privacy/details the sections entitled “Cookies, Web Beacons and Other Technologies” and the “IBM Software Products and Software-as-a-Service Privacy Statement” at http://www.ibm.com/software/info/product-privacy.

Support and comments

- For the entire IBM Worklight documentation set, training material and online forums where you can post questions, see the IBM website at:
  - http://www.ibm.com/mobile-docs
- **Support**
  - Software Subscription and Support (also referred to as Software Maintenance) is included with licenses purchased through Passport Advantage and Passport Advantage Express. For additional information about the International Passport Advantage Agreement and the IBM International Passport Advantage Express Agreement, visit the Passport Advantage website at:
    - http://www.ibm.com/software/passportadvantage
  - If you have a Software Subscription and Support in effect, IBM provides you assistance for your routine, short duration installation and usage (how-to) questions, and code-related questions. For additional details, consult your IBM Software Support Handbook at:
    - http://www.ibm.com/support/handbook
- **Comments**
  - We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this document. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.
  - For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.
  - When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state.
  - Thank you for your support.
  - Submit your comments in the IBM Worklight Developer Edition support community at:
    - https://www.ibm.com/developerworks/mobile/worklight/connect.html
  - If you would like a response from IBM, please provide the following information:
    - Name
    - Address
    - Company or Organization
    - Phone No.
    - Email address

Thank You