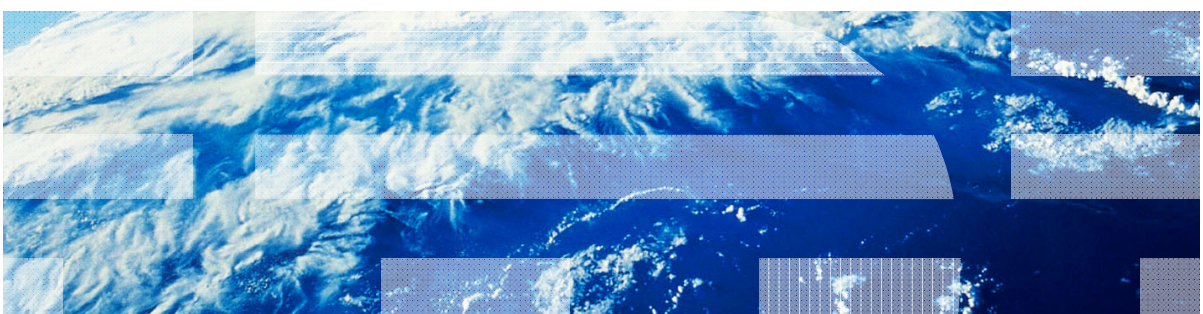
IBM Worklight V5.0.5
Getting Started

Module 7.10 – Using JSONStore



18 January 2013

Trademarks

- IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Worklight is a trademark or registered trademark of Worklight, an IBM Company. 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.
- 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

- Basic JSONStore Usage – Local Data
  - Initialization
  - Adding a document
  - Removing a document
  - Find document with query
  - Find all documents
  - Retrieving document count
  - Closing a collection
  - Destroying a collection
- Sample

Introduction to JSONStore

- This module covers basic tasks that you can perform on a local JSONStore collection.
- Adapter connectivity is covered in a later module.

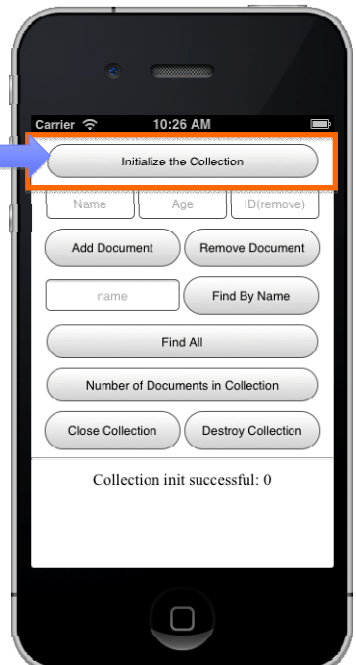
Agenda

- Basic JSONStore Usage – Local Data
  - Initialization
  - Adding a document
  - Removing a document
  - Find document with query
  - Find all documents
  - Retrieving document count
  - Closing a collection
  - Destroying a collection
- Sample

Initialization - UI

First, initialize a collection instance to be used by the rest of the app.

If the collection exists it is opened, otherwise an empty collection is created.



Initialization – The Code

```
WLJQ('button#initCollection').bind('click', function () {

    WL.Logger.debug('Called button#initCollection');

    var collectionName = 'users',
        searchFields = {name: 'string', age: 'number'};

    var win =   function (data) {
        logMessage('Collection init successful: ' + data);
    };

    var options = {onSuccess: win, onFailure: genericFailureCallback};

    usersCollection = WL.JSONStore.initCollection(collectionName, searchFields, options);

});
```

Name and Search Fields

Define your collection's name and your search fields.

By defining the search fields, you can query by them later in the app.

Initialization – The Code

```
WLJQ('button#initCollection').bind('click', function () {

    WL.Logger.debug('Called button#initCollection');

    var collectionName = 'users',
        searchFields = {name: 'string', age: 'number'};

    var win =   function (data) {
        logMessage('Collection init successful: ' + data);
    };

    var options = {onSuccess: win, onFailure: genericFailureCallback};

    usersCollection = WL.JSONStore.initCollection(collectionName, searchFields, options);

});
```

Options

Define the OnSuccess callback. The variable “data” will contain a return code. You cannot use the collection until you get an OnSuccess callback.

Initialization – The Code

```
WLJQ('button#initCollection').bind('click', function () {

    WL.Logger.debug('Called button#initCollection');

    var collectionName = 'users',
        searchFields = {name: 'string', age: 'number'};

    var win =   function (data) {
        logMessage('Collection init successful: ' + data);
    };

    var options = {onSuccess: win, onFailure: genericFailureCallback};

    usersCollection = WL.JSONStore.initCollection(collectionName, searchFields, options);
});
```

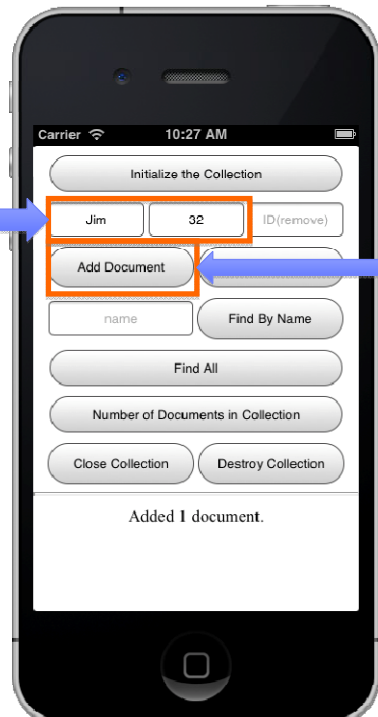
Initialize Collection

Finally, call the init function.

Agenda

- Basic JSONStore Usage – Local Data
  - Initialization
  - Adding a document
  - Removing a document
  - Find document with query
  - Find all documents
  - Retrieving document count
  - Closing a collection
  - Destroying a collection
- Sample

Adding a Document - UI



Specify a name and age.

Insert data into the *users* collection, creating a new document.

Adding a Document – The Code

```
WLJQ('button#add').bind('click', function () {

    WL.Logger.debug('Called button#store');

    var user_name = nameTag.val(),
        user_age = ageTag.val();

    if (user_name.length < 1 || user_age.length < 1 ||
        checkUndefOrNull(usersCollection)) {

        logMessage('You must init the collection and provide valid values');

    } else {

        var win =   function (data) {
            logMessage('Added ' + data + ' document.');
            nameTag.val('');
            ageTag.val('');
        };

        var options = {onSuccess: win, onFailure: genericFailureCallback};

        usersCollection.add({name: user_name, age: user_age},options);
    }
});
```

User Inputs

Get the values that were input and assign them to variables.

Adding a Document – The Code

```
WLJQ('button#add').bind('click', function () {

    WL.Logger.debug('Called button#store');

    var user_name = nameTag.val(),
        user_age = ageTag.val();

    if (user_name.length < 1 || user_age.length < 1 ||
        checkUndefOrNull(usersCollection)) {

        logMessage('You must init the collection and provide valid values');

    } else {

        var win =   function (data) {
            logMessage('Added ' + data + ' document.');
            nameTag.val('');
            ageTag.val('');
        };

        var options = {onSuccess: win, onFailure: genericFailureCallback};

        usersCollection.add({name: user_name, age: user_age},options);
    }
});
```

Options

Define the OnSuccess callback. The variable **data** will contain the number of documents successfully added.

Adding a Document – The Code

```
WLJQ('button#add').bind('click', function () {

    WL.Logger.debug('Called button#store');

    var user_name = nameTag.val(),
        user_age = ageTag.val();

    if (user_name.length < 1 || user_age.length < 1 ||
        checkUndefOrNull(usersCollection)) {

        logMessage('You must init the collection and provide valid values');

    } else {

        var win =   function (data) {
            logMessage('Added ' + data + ' document.');
            nameTag.val('');
            ageTag.val('');
        };

        var options = {onSuccess: win, onFailure: genericFailureCallback};

        usersCollection.add({name: user_name, age: user_age},options);
    }
});
```

Add the document

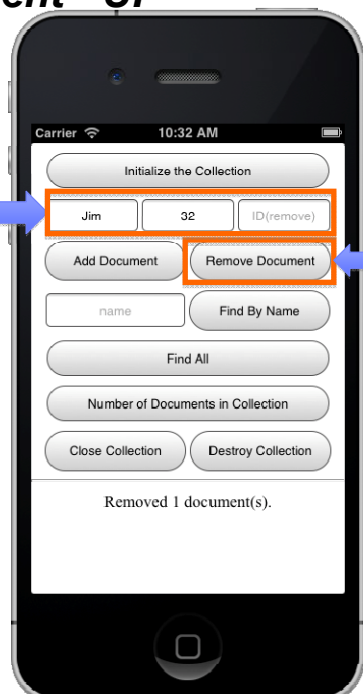
Finally, call the add function and pass in an array of objects including the name and age specified in the inputs.

Agenda

- Basic JSONStore Usage – Local Data
  - Initialization
  - Adding a document
  - Removing a document
  - Find document with query
  - Find all documents
  - Retrieving document count
  - Closing a collection
  - Destroying a collection
- Sample

Removing a Document - UI

Specify a name, age, or id of an existing document in the collection.



Mark a document or documents as removed from the “users” collection.

Removing a Document – The Code

```
WLJQ('button#btn_remove').bind('click', function () {

    WL.Logger.debug('Called button#btn_remove');

    var name = nameTag.val(),
        id = idTag.val(),
        age = ageTag.val();

    var win =   function (data) {
        logMessage('Removed ' + data + ' document(s).' );
        nameTag.val('');
        ageTag.val('');
        idTag.val('');
    };

    var options = {onSuccess: win, onFailure: genericFailureCallback};

    if (age.length > 0 && name.length > 0) { //Remove specific document
        logMessage('Remove Document by name and age.' );
        usersCollection.remove({name: name, age: age}, options);

    } else if (name.length > 0 && age.length < 1) { //Remove All By Name
        logMessage('Remove All by Name.' );
        usersCollection.remove({name: name}, options);

    } else if (id.length > 0) { //Remove document by ID
        logMessage('Remove by ID.' + id );
        usersCollection.remove(parseInt(id), options);
```

User Inputs

Get the values that were input and assign them to variables.

Removing a Document – The Code

```
WLJQ('button#btn_remove').bind('click', function () {

    WL.Logger.debug('Called button#btn_remove');

    var name = nameTag.val(),
        id = idTag.val(),
        age = ageTag.val();

    var win =   function (data) {
        logMessage('Removed ' + data + ' document(s).' );
        nameTag.val('');
        ageTag.val('');
        idTag.val('');
    };

    var options = {onSuccess: win, onFailure: genericFailureCallback};

    if (age.length > 0 && name.length > 0) { //Remove specific document
        logMessage('Remove Document by name and age.' );
        usersCollection.remove({name: name, age: age}, options);

    } else if (name.length > 0 && age.length < 1) { //Remove All By Name
        logMessage('Remove All by Name.' );
        usersCollection.remove({name: name}, options);

    } else if (id.length > 0) { //Remove document by ID
        logMessage('Remove by ID.' + id );
        usersCollection.remove(parseInt(id), options);
```

Options

Define the OnSuccess callback. The variable **data** will contain the number of documents marked as removed.

Note that the documents will not actually be removed until `push` is executed (covered in a subsequent module).

Removing a Document – The Code

```
WLJQ('button#btn_remove').bind('click', function () {

    WL.Logger.debug('Called button#btn_remove');

    var name = nameTag.val(),
        id = idTag.val(),
        age = ageTag.val();

    var win =   function (data) {
        logMessage('Removed ' + data + ' document(s).' );
        nameTag.val('');
        ageTag.val('');
        idTag.val('');
    };

    var options = {onSuccess: win, onFailure: genericFailureCallback};

    if (age.length > 0 && name.length > 0) { //Remove specific document
        logMessage('Remove Document by name and age.' );
        usersCollection.remove({name: name, age: age}, options);

    } else if (name.length > 0 && age.length < 1) { //Remove All By Name
        logMessage('Remove All by Name.' );
        usersCollection.remove({name: name}, options);

    } else if (id.length > 0) { //Remove document by ID
        logMessage('Remove by ID.' + id );
        usersCollection.remove(parseInt(id), options);
```

Removal by 2 fields

If both name and age were specified in the inputs then call remove with an array containing both objects.

All documents that match that name and age will be marked for removal.

Removing a Document – The Code

```
WLJQ('button#btn_remove').bind('click', function () {

    WL.Logger.debug('Called button#btn_remove');

    var name = nameTag.val(),
        id = idTag.val(),
        age = ageTag.val();

    var win =   function (data) {
        logMessage('Removed ' + data + ' document(s).' );
        nameTag.val('');
        ageTag.val('');
        idTag.val('');
    };

    var options = {onSuccess: win, onFailure: genericFailureCallback};

    if (age.length > 0 && name.length > 0) { //Remove specific document
        logMessage('Remove Document by name and age.' );
        usersCollection.remove({name: name, age: age}, options);

    } else if (name.length > 0 && age.length < 1) { //Remove All By Name
        logMessage('Remove All by Name.' );
        usersCollection.remove({name: name}, options);

    } else if (id.length > 0) { //Remove document by ID
        logMessage('Remove by ID.' + id );
        usersCollection.remove(parseInt(id), options);
```

Removal by 1 field

This scenario is the same as in the previous slide but it uses only a single object.

Removing a Document – The Code

```
WLJQ('button#btn_remove').bind('click', function () {

    WL.Logger.debug('Called button#btn_remove');

    var name = nameTag.val(),
        id = idTag.val(),
        age = ageTag.val();

    var win =   function (data) {
        logMessage('Removed ' + data + ' document(s).' );
        nameTag.val('');
        ageTag.val('');
        idTag.val('');
    };

    var options = {onSuccess: win, onFailure: genericFailureCallback};

    if (age.length > 0 && name.length > 0) { //Remove specific document
        logMessage('Remove Document by name and age.' );
        usersCollection.remove({name: name, age: age}, options);

    } else if (name.length > 0 && age.length < 1) { //Remove All By Name
        logMessage('Remove All by Name.' );
        usersCollection.remove({name: name}, options);

    } else if (id.length > 0) { //Remove document by ID
        logMessage('Remove by ID.' + id );
        usersCollection.remove(parseInt(id), options);
```

Removal by ID

This scenario is the same as the previous slide except that it uses an ID value which will remove only one specific document.

Agenda

- Basic JSONStore Usage – Local Data
  - Initialization
  - Adding a document
  - Removing a document
  - Find document with query
  - Find all documents
  - Retrieving document count
  - Closing a collection
  - Destroying a collection
- Sample

Find Document with Query- UI

Specify a name to search for, that exists in the collection.

An array of all documents that match that name is displayed.

Find Document with Query– The Code

```
WLJQ('button#find_name').bind('click', function () {

    WL.Logger.debug('Called button#find_name');

    if (!checkColInit(usersCollection)) {return;}

    var user_name = nameFindTag.val();

    if (user_name.length < 1) {
        logMessage('You must provide a search name');
    } else {

        var query = {name: user_name};

        var win =  function (data) {
            reloadTable(data);
            nameFindTag.val('');
        };

        var options = {onSuccess: win, onFailure: genericFailureCallback};

        usersCollection.find(query, options);
    }
});
```

Search Query

Construct a simple search query using the value for **name** that was input by the user.

Find Document with Query– The Code

```
WLJQ('button#find_name').bind('click', function () {

    WL.Logger.debug('Called button#find_name');

    if (!checkColInit(usersCollection)) {return;}

    var user_name = nameFindTag.val();

    if (user_name.length < 1) {
        logMessage('You must provide a search name');
    } else {

        var query = {name: user_name};

        var win =  function (data) {
            reloadTable(data);
            nameFindTag.val('');
        };

        var options = {onSuccess: win, onFailure: genericFailureCallback};

        usersCollection.find(query, options);
    }
});
```

Options

Define the OnSuccess callback. The variable **data** will contain the number of documents successfully retrieved.

Find Document with Query– The Code

```
WLJQ('button#find_name').bind('click', function () {

    WL.Logger.debug('Called button#find_name');

    if (!checkColInit(usersCollection)) {return;}

    var user_name = nameFindTag.val();

    if (user_name.length < 1) {
        logMessage('You must provide a search name');
    } else {

        var query = {name: user_name};

        var win =  function (data) {
            reloadTable(data);
            nameFindTag.val('');
        };

        var options = {onSuccess: win, onFailure: genericFailureCallback};

        usersCollection.find(query, options);
    }
});
```

Retrieving the documents

Finally, call the find method and pass in the query and options. If any documents are found, they are displayed in a table.

Agenda

- Basic JSONStore Usage – Local Data
  - Initialization
  - Adding a document
  - Removing a document
  - Find document with query
  - Find all documents
  - Retrieving document count
  - Closing a collection
  - Destroying a collection
- Sample

Find All Documents - UI

Add a document into the `users` collection if not done already.

Retrieve all documents in the `users` collection.

Display documents in a table.

Find All Documents – The Code

```
WLJQ('button#find_all').bind('click', function ()

    WL.Logger.debug('Called button#find_all');

    if (!checkColInit(usersCollection)) {return;}

    var win =   function (data) {
        reloadTable(data);
    };

    var options = {onSuccess: win, onFailure: genericFailureCallback};

    usersCollection.findAll(options);
});
```

Options

Define the OnSuccess callback. The variable **data** will contain an array of all the documents contained in the `users` collection.
In this case there is only one document to display.

Find All Documents – The Code

Find the documents

Finally, call the findAll method and pass in the options.

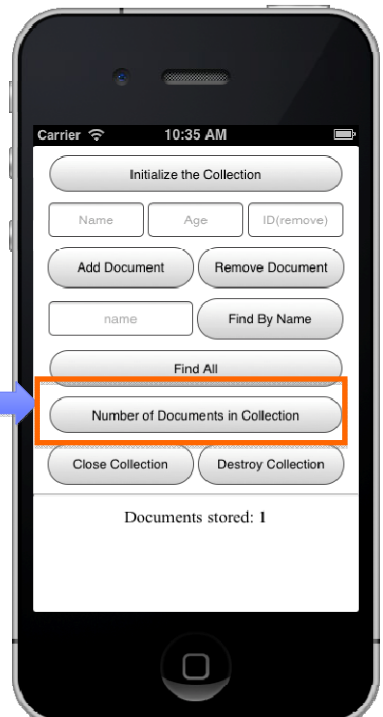
```
WLJQ('button#find_all').bind('click', function () {

    WL.Logger.debug('Called button#find_all');

    if (!checkColInit(usersCollection)) {return;}

    var win =   function (data) {
        reloadTable(data);
    };

    var options = {onSuccess: win, onFailure: genericFailureCallback};

    usersCollection.findAll(options);
});
```

Agenda

- Basic JSONStore Usage – Local Data
  - Initialization
  - Adding a document
  - Removing a document
  - Find document with query
  - Find all documents
  - Retrieving document count
  - Closing a collection
  - Destroying a collection
- Sample

Retrieving Document Count - UI

Get the number of individual documents currently stored in the `users` collection.



Retrieving Document Count – The Code

```
WLJQ('button#count').bind('click', function () {

    WL.Logger.debug('Called button#count');

    if (checkUndefOrNull(usersCollection)) {
        logMessage('You must init the collection first
    } else {

        var win =   function (data) {
            logMessage('Documents stored: ' + data);
        };

        var options = {onSuccess: win, onFailure: genericFailureCallback};

        usersCollection.count(options);
    }
});
```

Options

Define the OnSuccess callback. The variable **data** will contain the number of documents stored in the `users` collection.

This number does not reflect those documents marked *removed*.

Retrieving Document Count – The Code

```
WLJQ('button#count').bind('click', function () {

    WL.Logger.debug('Called button#count');

    if (checkUndefOrNull(usersCollection)) {
        logMessage('You must init the collection first');
    } else {

        var win =   function (data) {
            logMessage('Documents stored: ' + data);
        };

        var options = {onSuccess: win, onFailure: genericFailureCallback};

        usersCollection.count(options);
    }
});
```

Find the documents

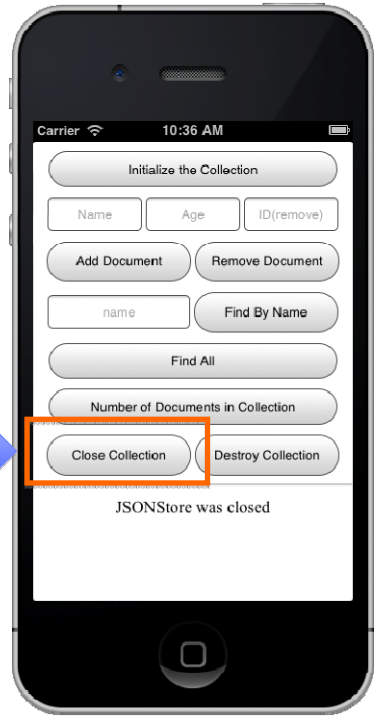
Finally, call the count method and pass in the options.

Agenda

- Basic JSONStore Usage – Local Data
  - Initialization
  - Adding a document
  - Removing a document
  - Find document with query
  - Find all documents
  - Retrieving document count
  - Closing a collection
  - Destroying a collection
- Sample

Closing a Collection - UI

Close all the collections in the JSONStore.



Closing a Collection – The Code

Options

Define the OnSuccess callback. If it is successful you can display a meaningful message here.

```
WLJQ('button#close').bind('click', function () {

    WL.Logger.debug('Called button#close');

    var win =   function () {
        logMessage('JSONStore was closed');
    };

    var options = {onSuccess: win, onFailure: genericFailureCallback};

    WL.JSONStore.closeAll(options);

});
```

Closing a Collection – The Code

```
WLJQ('button#close').bind('click', function () {

    WL.Logger.debug('Called button#close');

    var win =   function () {
        logMessage('JSONStore was closed');
    };

    var options = {onSuccess: win, onFailure: genericFailureCallback};

    WL.JSONStore.closeAll(options);

});
```

Close all collections

Finally, we call our closeAll method and pass in our options.

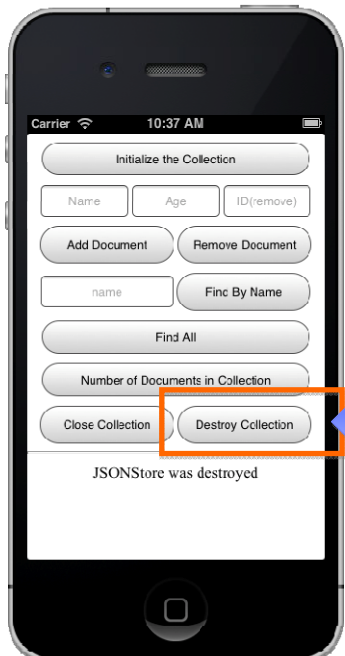
After a closeAll, each collection in the store will need to have WL.JSONStore.initCollection called again before that collection can be used.

Note that if the collections in the persistent store are password protected, the password will need to be specified using WL.JSONStore.usePassword (covered in a subsequent module)

Agenda

- Basic JSONStore Usage – Local Data
  - Initialization
  - Adding a document
  - Removing a document
  - Find document with query
  - Find all documents
  - Retrieving document count
  - Closing a collection
  - Destroying a collection
- Sample

Destroying a Collection - UI



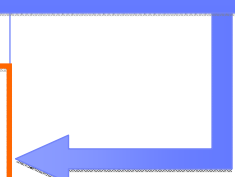
Destroy the internal storage associated with a collection and clear the keychain that stores necessary keys for decrypting the internal storage.

Destroying a Collection – The Code

Options

Define the OnSuccess callback. If it is successful you can display a meaningful message here.

```
WLJQ('button#destroy').bind('click', function () {

    WL.Logger.debug('Called button#destroy');

    var win =   function () {
        logMessage('JSONStore was destroyed');
    };

    var options = {onSuccess: win, onFailure: genericFailureCallback};

    WL.JSONStore.destroy(options);
});
```



Destroying a Collection – The Code

```
WLJQ('button#destroy').bind('click', function () {

    WL.Logger.debug('Called button#destroy');

    var win =   function () {
        logMessage('JSONStore was destroyed');
    };

    var options = {onSuccess: win, onFailure: genericFailureCallback};

    WL.JSONStore.destroy(options);
});
```

Destroy all collections

Finally, call the `destroy` method and pass in the options.

Agenda

- Basic JSONStore Usage – Local Data
  - Initialization
  - Adding a document
  - Removing a document
  - Find document with query
  - Find all documents
  - Retrieving document count
  - Closing a collection
  - Destroying a collection
- Sample

Sample

- The sample for this training module can be found in the Getting Started page of the IBM Worklight documentation website at
  - http://www.ibm.com/mobile-docs

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.

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 forums at:
    - https://www.ibm.com/developerworks/mobile/mobileforum.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