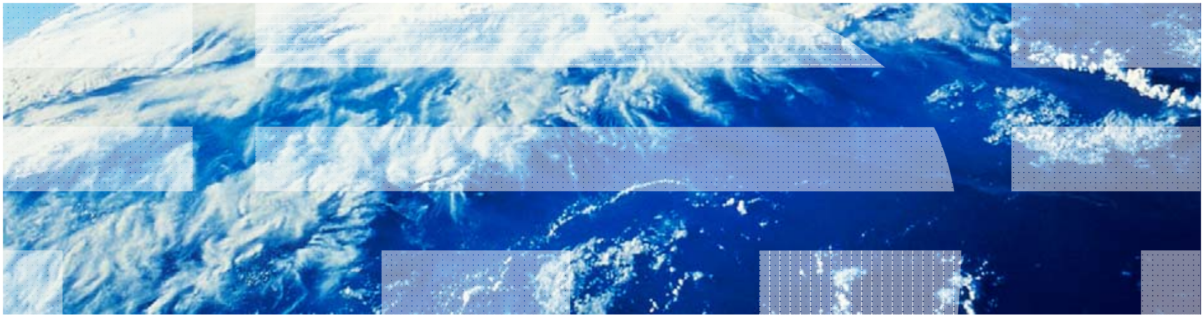
IBM Worklight V5.0.6 Getting Started

JSONStore – Synchronizing client and server databases



15 March 2013

Trademarks

About IBM®

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

Agenda

- Intermediate JSONStore Usage
  - Using the JSONStore wizard
  - Generated Code
  - Loading data from an adapter
  - Get Documents Requiring Push
  - Push All
  - Push Selected
- Sample

Intermediate JSONStore Usage

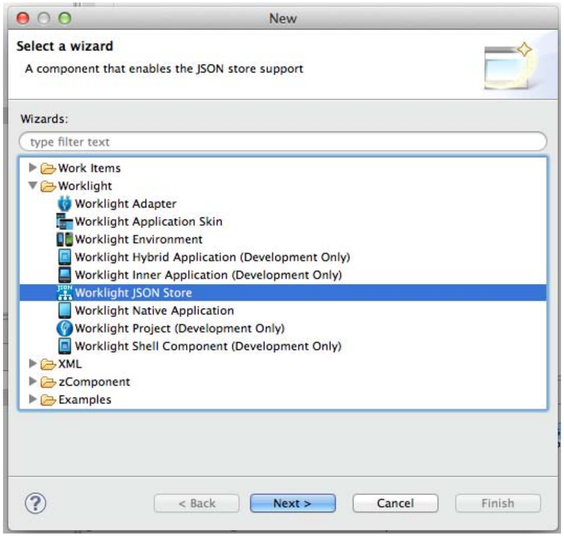
- This module covers intermediate-level tasks that can be performed on a local JSONStore collection, including the use of adapters to connect to a back-end.

Agenda

- Intermediate JSONStore Usage
  - Using the JSONStore wizard
  - Generated Code
  - Loading data from an adapter
  - Get Documents Requiring Push
  - Push All
  - Push Selected
- Sample

Using the JSONStore wizard

- To start the wizard, right-click the app that you want to set up and select **New > Other > Worklight > Worklight JSON Store**


New

Select a wizard

A component that enables the JSON store support

Wizards:

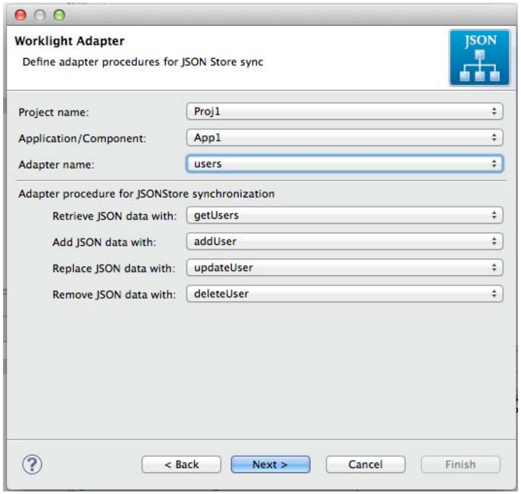
type filter text

- Work Items
- Worklight
  - Worklight Adapter
  - Worklight Application Skin
  - Worklight Environment
  - Worklight Hybrid Application (Development Only)
  - Worklight Inner Application (Development Only)
  - Worklight JSON Store
  - Worklight Native Application
  - Worklight Project (Development Only)
  - Worklight Shell Component (Development Only)
- XML
- zComponent
- Examples

Using the JSONStore wizard – Choosing the adapter

Note: The adapter must already be deployed.

- Select the **user** adapter that is included in the sample project.
- In the adapter, the following sync operation procedures are defined:
  - Retrieve (required)
  - Add
  - Replace
  - Remove



Worklight Adapter

Define adapter procedures for JSON Store sync

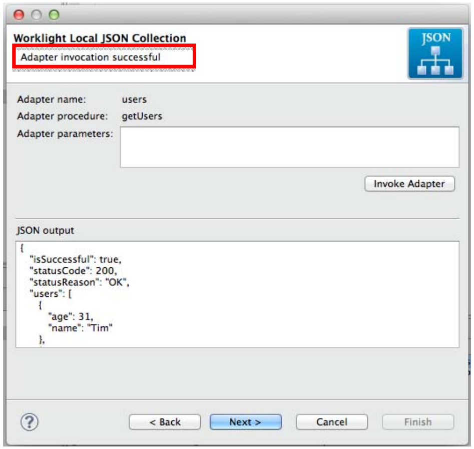
Project name:	Proj1
Application/Component:	App1
Adapter name:	users

Adapter procedure for JSONStore synchronization

Retrieve JSON data with:	getUsers
Add JSON data with:	addUser
Replace JSON data with:	updateUser
Remove JSON data with:	deleteUser

Using the JSONStore wizard – Testing the adapter

- In this step, you test the adapter invocation.
- After you click **Invoke Adapter**, you see the JSON output. A message is displayed that indicates whether the adapter invocation is a success or a failure.



Adapter invocation successful

Adapter name: users
Adapter procedure: getUsers
Adapter parameters:

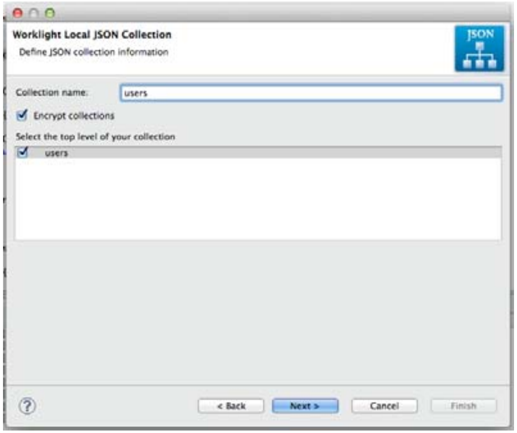
JSON output

```
{
  "isSuccessful": true,
  "statusCode": 200,
  "statusReason": "OK",
  "users": [
    {
      "age": 31,
      "name": "Tim"
    },
```

Using the JSONStore wizard – Naming the Collection

- In this step, you define the JSON collection information

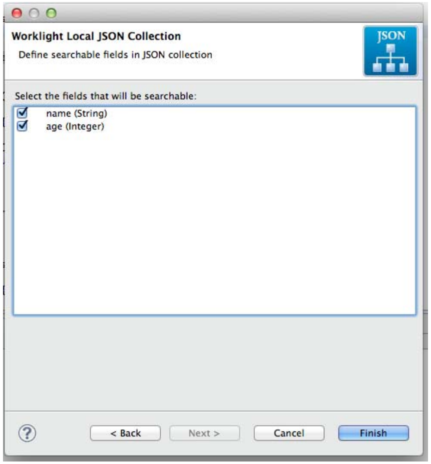
1. Set **Collection name** to `users`.
2. Select **Encrypt collections**.
3. Select the check box that is next to **users**.


Worklight Local JSON Collection
Define JSON collection information

Collection name: users
☑ Encrypt collections
Select the top level of your collection
☑ users

Using the JSONStore wizard – Search fields

- Choose the fields that you want to make searchable.
- The searchable fields define the keys in JavaScript objects that are indexed, and determine what you can query in a collection.



Worklight Local JSON Collection

Define searchable fields in JSON collection

Select the fields that will be searchable:

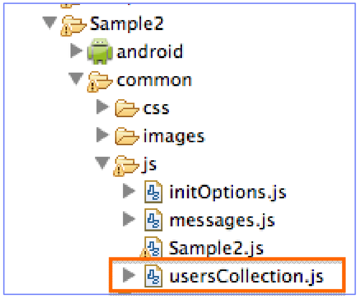
- ✓ name (String)
- ✓ age (Integer)

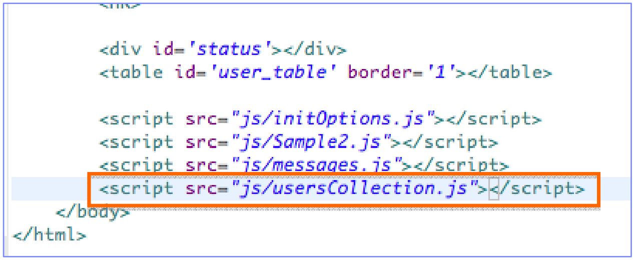
Agenda

- Intermediate JSONStore Usage
  - Using the JSONStore wizard
  - Generated Code
  - Loading data from an adapter
  - Get Documents Requiring Push
  - Push All
  - Push Selected
- Sample

Generated Code

- When the wizard completes, the `usersCollection.js` file is generated in the `common/js` directory. This file contains the necessary code to initialize the local store and to connect it to the adapter.
- To link this file to your app, first add script tags to the generated file in the *`app_name`*`.html` file.
  - *(continued on the next slide)*



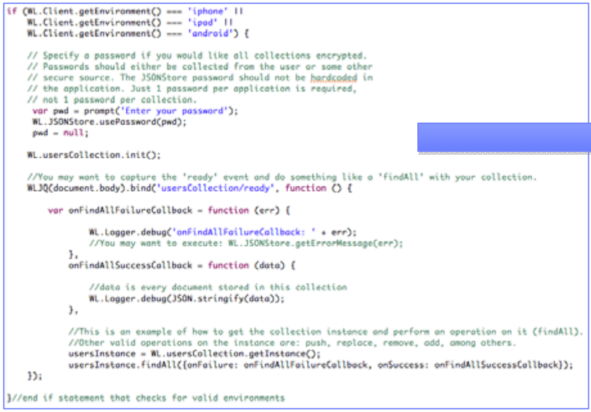


```
        <div id='status'></div>
        <table id='user_table' border='1'></table>

        <script src="js/initOptions.js"></script>
        <script src="js/Sample2.js"></script>
        <script src="js/messages.js"></script>
        <script src="js/usersCollection.js"></script>
    </body>
</html>
```

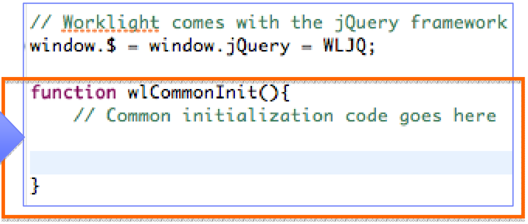
Generated Code

- Next, copy the last part of the generated code into the **wlCommonInit()** method inside the `app_name.js` file.
- This part of the code is called when the collection is ready for use and performs a **findAll** on the data.

`usersCollection.js` `app_name.js`



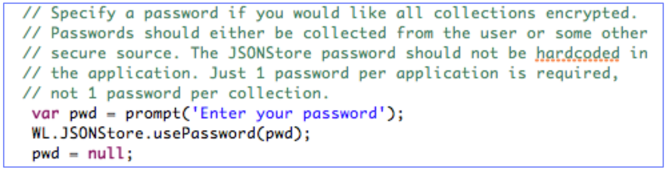
```
if (WL.Client.getEnvironment() === 'iphone' ||
    WL.Client.getEnvironment() === 'ipad' ||
    WL.Client.getEnvironment() === 'android') {

    // Specify a password if you would like all collections encrypted.
    // Passwords should either be collected from the user or some other
    // secure source. The JSONStore password should not be hardcoded in
    // the application. Just 1 password per application is required,
    // not 1 password per collection.
     var pwd = prompt('Enter your password');
     WL.JSONStore.usePassword(pwd);
     pwd = null;

    WL.usersCollection.init();

    //You may want to capture the 'ready' event and do something like a 'findAll' with your collection.
    WLJQ(document.body).bind('usersCollection/ready', function () {

        var onFindAllFailureCallback = function (err) {

                WL.Logger.debug('onFindAllFailureCallback: ' + err);
                //You may want to execute: WL.JSONStore.getErrorMessage(err);
            },
            onFindAllSuccessCallback = function (data) {

                //data is every document stored in this collection
                WL.Logger.debug(JSON.stringify(data));
            },

            //This is an example of how to get the collection instance and perform an operation on it (findAll).
            //Other valid operations on the instance are: push, replace, remove, add, among others.
            usersInstance = WL.usersCollection.getInstance();
            usersInstance.findAll({onFailure: onFindAllFailureCallback, onSuccess: onFindAllSuccessCallback});
    });

}//end if statement that checks for valid environments
```



```
// Worklight comes with the jQuery framework
window.$ = window.jQuery = WLJQ;

function wlCommonInit(){
    // Common initialization code goes here


}
```

Generated Code

- The following options are configured in the `usersCollection.js` file:
  - **userSearchFields** – Search fields that you selected when you defined the searchable fields in the wizard.
  - **userAdapterOptions** – (see next slide)
  - **onSuccess/onFailure Callbacks** – Called after the collection provisioning is attempted.
  - **Encryption** – If you chose to encrypt your data, the following code is also generated to enable encryption.

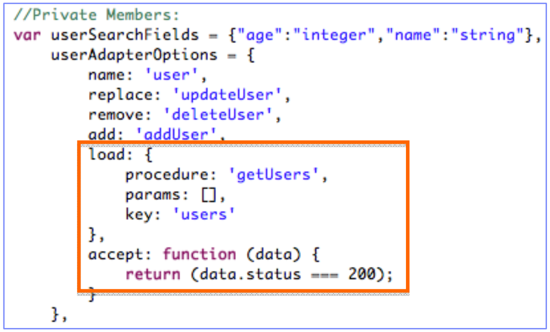


```
// Specify a password if you would like all collections encrypted.
// Passwords should either be collected from the user or some other
// secure source. The JSONStore password should not be hardcoded in
// the application. Just 1 password per application is required,
// not 1 password per collection.
 var pwd = prompt('Enter your password');
 WL.JSONStore.usePassword(pwd);
 pwd = null;
```

- For more information about encryption, see the module **JSONStore – Encrypting sensitive data**.

Generated Code - userAdapterOptions

- userAdapterOptions
  - Contains the name of the adapter that is linked to this collection and the name of various procedures that perform operations when documents are pushed back to the server.
  - In addition to adapter procedures, `load` and `accept` objects are defined:
    - **Load** – determines the procedure that is used to load documents into the collection.
    - **Accept** – called every time an adapter procedure is invoked to push data to the back-end service.



```
//Private Members:
var userSearchFields = {"age":"integer","name":"string"},
    userAdapterOptions = {
        name: 'user',
        replace: 'updateUser',
        remove: 'deleteUser',
        add: 'addUser',
        load: {
            procedure: 'getUsers',
            params: [],
            key: 'users'
        },
        accept: function (data) {
            return (data.status === 200);
        }
    },
```

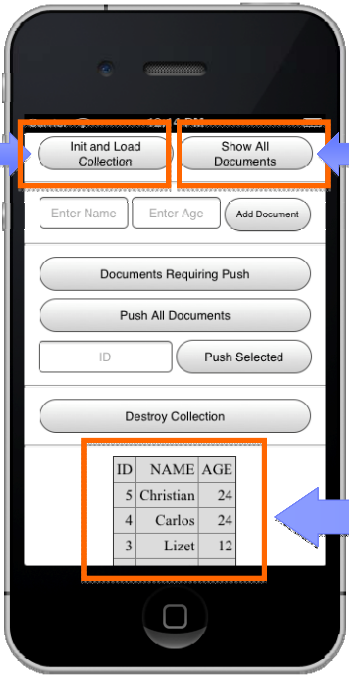
Agenda

- Intermediate JSONStore Usage
  - Using the JSONStore wizard
  - Generated Code
  - Loading data from an adapter
  - Get Documents Requiring Push
  - Push All
  - Push Selected
- Sample

Loading Data from an adapter – UI

Initialize the collection and load data from the `user` adapter into the local store.

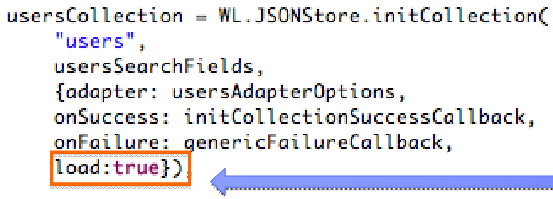
For this module, the previously generated code is wrapped inside a JavaScript event that is fired when you click **init**.



ID	NAME	AGE
5	Christian	24
4	Carlos	24
3	Lizet	12

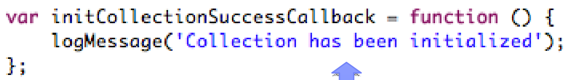
To display the documents, click **Show All**, which performs a findAll on the collection.

Loading Data from an adapter – The Code



```
usersCollection = WL.JSONStore.initCollection(
    "users",
    usersSearchFields,
    {adapter: usersAdapterOptions,
    onSuccess: initCollectionSuccessCallback,
    onFailure: genericFailureCallback,
    load:true})
```

In the generated code, the **initCollection** method has the `load` object set to **true**, meaning that the data is loaded by using the adapter that is linked to the collection.



```
var initCollectionSuccessCallback = function () {
    logMessage('Collection has been initialized');
};
```

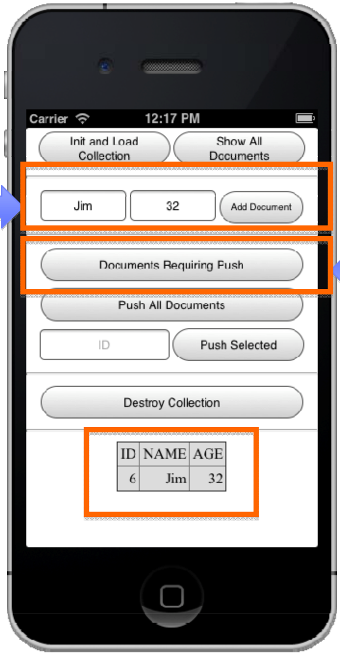
If initialization is successful, the **onSuccess** callback is fired. At this point, the collection is configured and ready.

Agenda

- Intermediate JSONStore Usage
  - Using the JSONStore wizard
  - Generated Code
  - Loading data from an adapter
  - Get Documents Requiring Push
  - Push All
  - Push Selected
- Sample

Get Documents Requiring Push – UI

First, add a document (or documents) to the collection.



ID	NAME	AGE
6	Jim	32

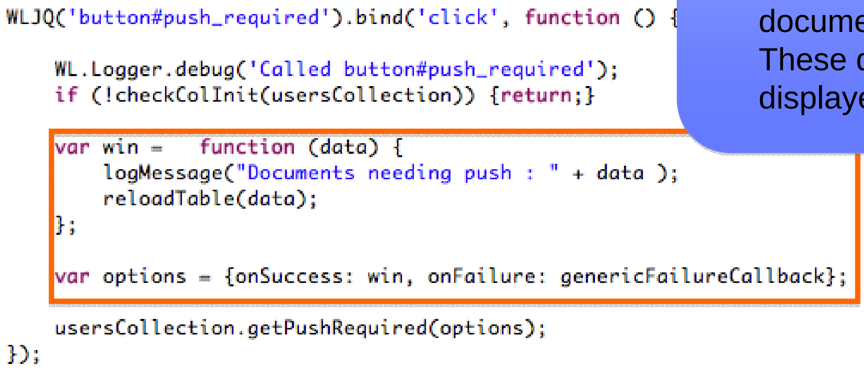
Next, check what documents are currently marked as requiring push.

Any added document is initially marked as *requiring push*, meaning that it has not been pushed to the back-end system.

Get Documents Requiring Push – The Code

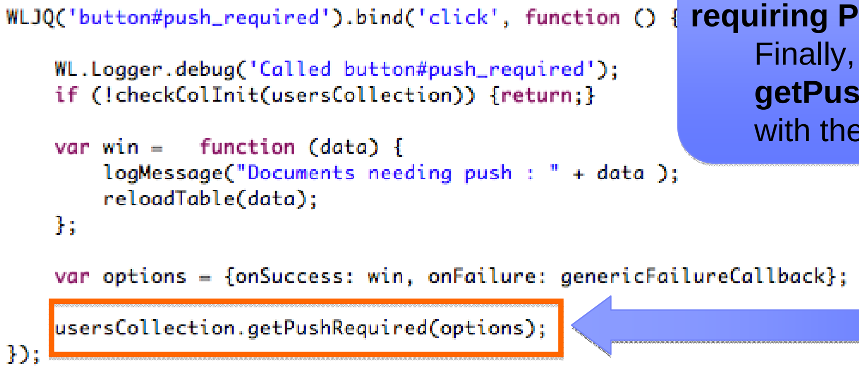
Options

Define the **OnSuccess** callback. The variable **data** will contain an array of all documents that require push. These documents are displayed in a table.



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

    WL.Logger.debug('Called button#push_required');
    if (!checkColInit(usersCollection)) {return;}

    var win =   function (data) {
        logMessage("Documents needing push : " + data );
        reloadTable(data);
    };

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

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

Get Documents Requiring Push – The Code



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

    WL.Logger.debug('Called button#push_required');
    if (!checkColInit(usersCollection)) {return;}

    var win =   function (data) {
        logMessage("Documents needing push : " + data );
        reloadTable(data);
    };

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

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

Get documents that are requiring Push

Finally, call the **getPushRequired** method with the **onSuccess** callback.

Agenda

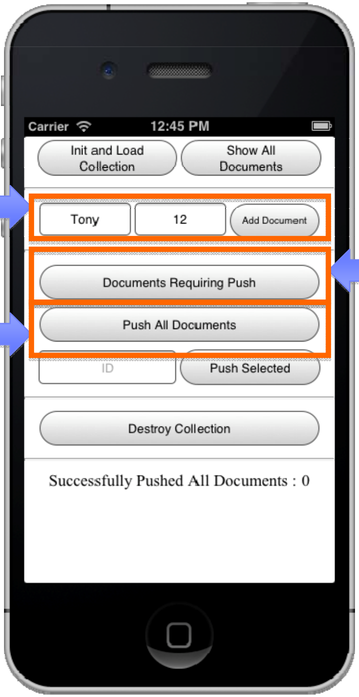
- Intermediate JSONStore Usage
  - Using the JSONStore wizard
  - Generated Code
  - Loading data from an adapter
  - Get Documents Requiring Push
  - Push All
  - Push Selected
- Sample

Push All – UI

First, add a new document to the collection.

Then, check that this document is visible in the list of documents that require push.

Finally, push all documents that are marked as requiring push.

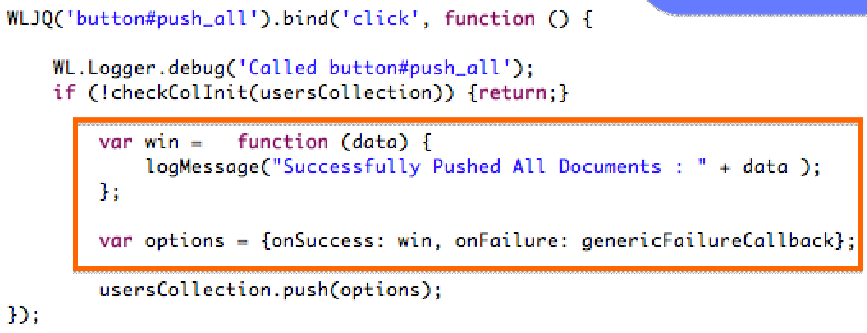


Successfully Pushed All Documents : 0

Push All – The Code

Options

Define the **OnSuccess** callback. The variable **data** will contain a status code (as defined in the JSONStore documentation).

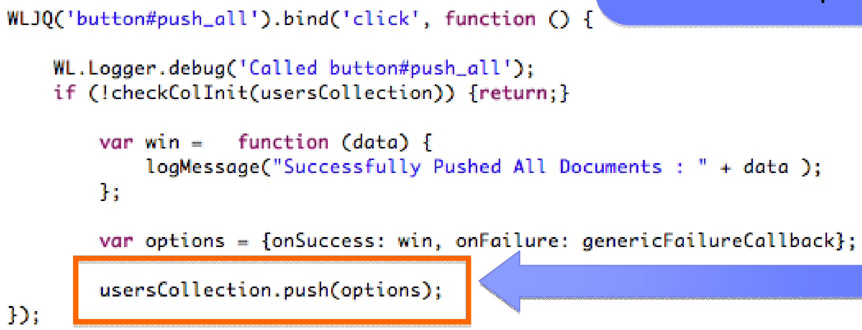


```
WLJQ('button#push_all').bind('click', function () {

    WL.Logger.debug('Called button#push_all');
    if (!checkColInit(usersCollection)) {return;}

        var win =   function (data) {
            logMessage("Successfully Pushed All Documents : " + data );
        };

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

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

Push All – The Code

Push all documents that are requiring Push

Finally, call the **pushAll** method with the options.



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

    WL.Logger.debug('Called button#push_all');
    if (!checkColInit(usersCollection)) {return;}

        var win =   function (data) {
            logMessage("Successfully Pushed All Documents : " + data );
        };

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

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

Agenda

- Intermediate JSONStore Usage
  - Using the JSONStore wizard
  - Generated Code
  - Loading data from an adapter
  - Get Documents Requiring Push
  - Push All
  - Push Selected
- Sample

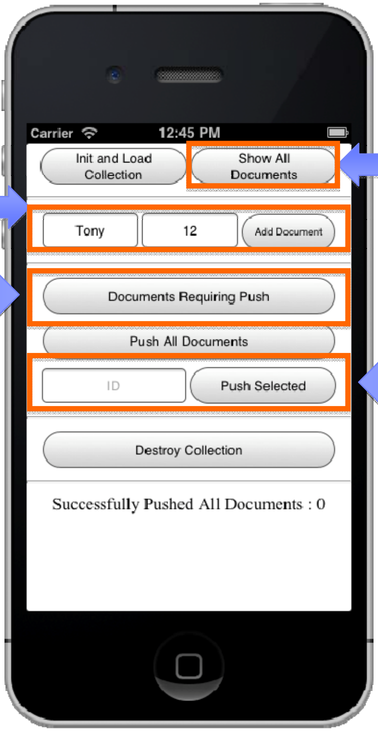
Push Selected – UI

First, add a document to the collection.

Next, show all documents and notice the added document **id**.

Check that this document is visible in the list of documents that require push.

Finally, enter the document **id** and push only this specific document.

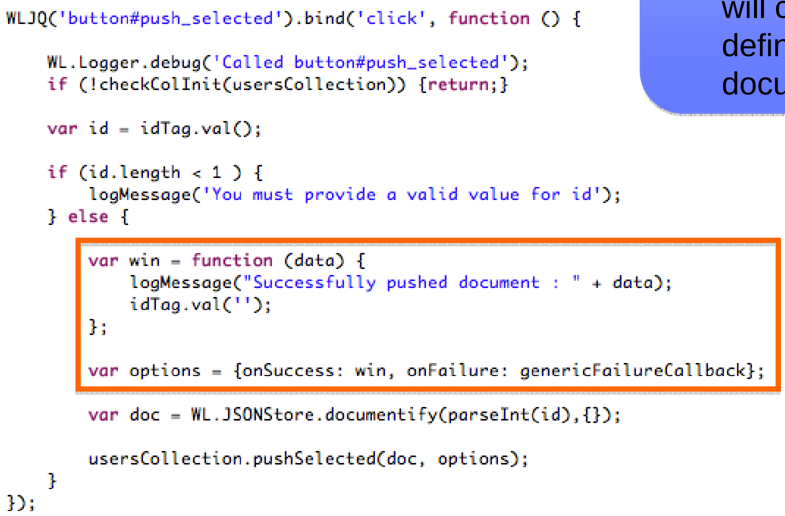


Successfully Pushed All Documents : 0

Push Selected – The Code

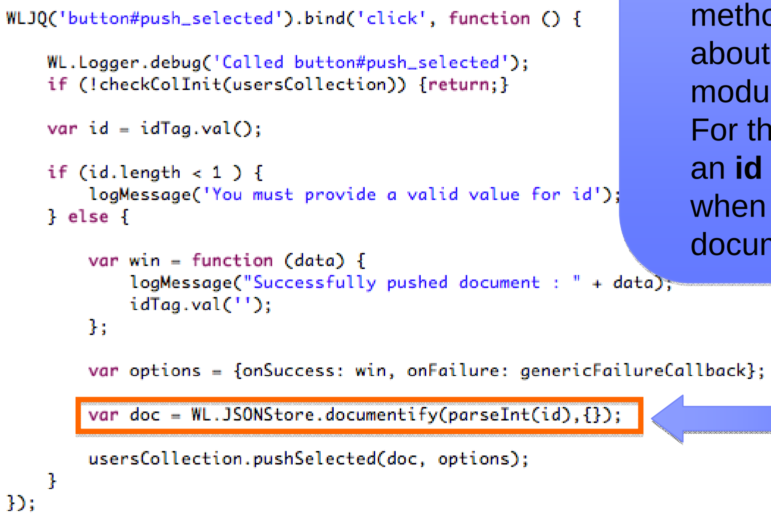
Options

Define the **OnSuccess** callback. The variable **data** will contain a status code (as defined in the JSONStore documentation).



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

    WL.Logger.debug('Called button#push_selected');
    if (!checkColInit(usersCollection)) {return;}

    var id = idTag.val();

    if (id.length < 1 ) {
        logMessage('You must provide a valid value for id');
    } else {

        var win = function (data) {
            logMessage("Successfully pushed document : " + data);
            idTag.val('');
        };

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

        var doc = WL.JSONStore.documentify(parseInt(id),{});

        usersCollection.pushSelected(doc, options);
    }
});
```

Push Selected – The Code



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

    WL.Logger.debug('Called button#push_selected');
    if (!checkColInit(usersCollection)) {return;}

    var id = idTag.val();

    if (id.length < 1 ) {
        logMessage('You must provide a valid value for id');
    } else {

        var win = function (data) {
            logMessage("Successfully pushed document : " + data);
            idTag.val('');
        };

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

        var doc = WL.JSONStore.documentify(parseInt(id),{});

        usersCollection.pushSelected(doc, options);
    }
});
```

Create a document

Create a document object by using the **documentify** method (you learn more about this method in a later module).

For this module, specify only an **id** and an empty object when you create the document.

Push Selected – The Code

Push the selected document

Finally call the **pushSelected** method with the newly created document and options as parameters.

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

    WL.Logger.debug('Called button#push_selected');
    if (!checkColInit(usersCollection)) {return;}

    var id = idTag.val();

    if (id.length < 1 ) {
        logMessage('You must provide a valid value for id');
    } else {

        var win = function (data) {
            logMessage("Successfully pushed document : " + data);
            idTag.val('');
        };

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

        var doc = WL.JSONStore.documentify(parseInt(id),{});

        usersCollection.pushSelected(doc, options);
    }
});
```

Agenda

- Intermediate JSONStore Usage
  - Using the JSONStore wizard
  - Generated Code
  - Loading data from an adapter
  - Get Documents Requiring Push
  - Push All
  - Push Selected
- 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

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