Kony Fabric

Sync ORM API Guide

Release V8

Document Relevance and Accuracy

This document is considered relevant to the release stated on this title page and the document version stated on the Revision History page. Remember to always view and download the latest document version relevant to the software release you are using.

September, 2017

Revision History

Date	Document Version	Description of Modifications/Release
09/08/2017	1.0	Document updated for release V8. Worked on rebranding of MobileFabric to Kony Fabric.

Table of Contents

1. Preface

Kony Fabric Sync Cloud and On-Premises Client ORM API Guide (JavaScript) enables developers to access underlying SQLite device database eliminating developer overhead of writing explicit queries to access the data. There are various APIs to help the developer meet their requirements.

The APIs are categorized as follows:

- Application Level
- Object Level
- Scope Level

Supported Platforms

iOS, Android, and Windows.

For detailed information on supported platforms, refer to *kony_sync_on_premises_server_planning_guide.*

1.1 Purpose

This document enables developers to access underlying SQLite device database eliminating developer overhead of writing explicit queries to access the data.

1.2 Intended Audience

This document is intended for developers who are responsible for performing device side operations.

1.3 Formatting Conventions

The following formatting conventions are used throughout the document:

Convention	Explanation
`Monospace`	■ User input text, system prompts, and responses ■ File path ■ Commands ■ Program code ■ File Names.
Italic	■ Emphasis ■ Names of books, and documents ■ New terminology.
Bold	■ Windows ■ Menus ■ Buttons ■ Icons ■ Fields ■ Tabs.
URL	Active link to a URL
Note	Provides helpful hints or additional information
Important	Highlights actions or information that might cause problems to systems or data

1.4 Contact Us

We welcome your feedback on our documentation. Write to us at techpubs@kony.com. For technical questions, suggestions, comments or to report problems on Kony product line, contact support@kony.com.

2. Application Level

This set of APIs help the developer to perform various ORM operations at Application level.

2.1 sync.init

This function initializes the creation of device database as well sync environment. On successful sync init, the database structure without data (schema) gets created in the device database. Once the client database is created, subsequent calls to *sync.init* just initialize the sync environment.

> ***Note:*** You should always call this function after application re-start and before you perform any Sync operation. You need to perform the *sync.init* function only once in the life span of an application.

2.1.1 Signature

*sync.init (**initSuccessCallback ,initFailCallback**)*

2.1.2 Parameters

- initSuccessCallback[function] - Optional

  Specifies the function that gets invoked on success

- initFailCallback[function] - Optional

  Specifies the function that gets invoked on failure

2.1.3 Platform Availability

Available on all platforms mentioned.

2.1.4 Example

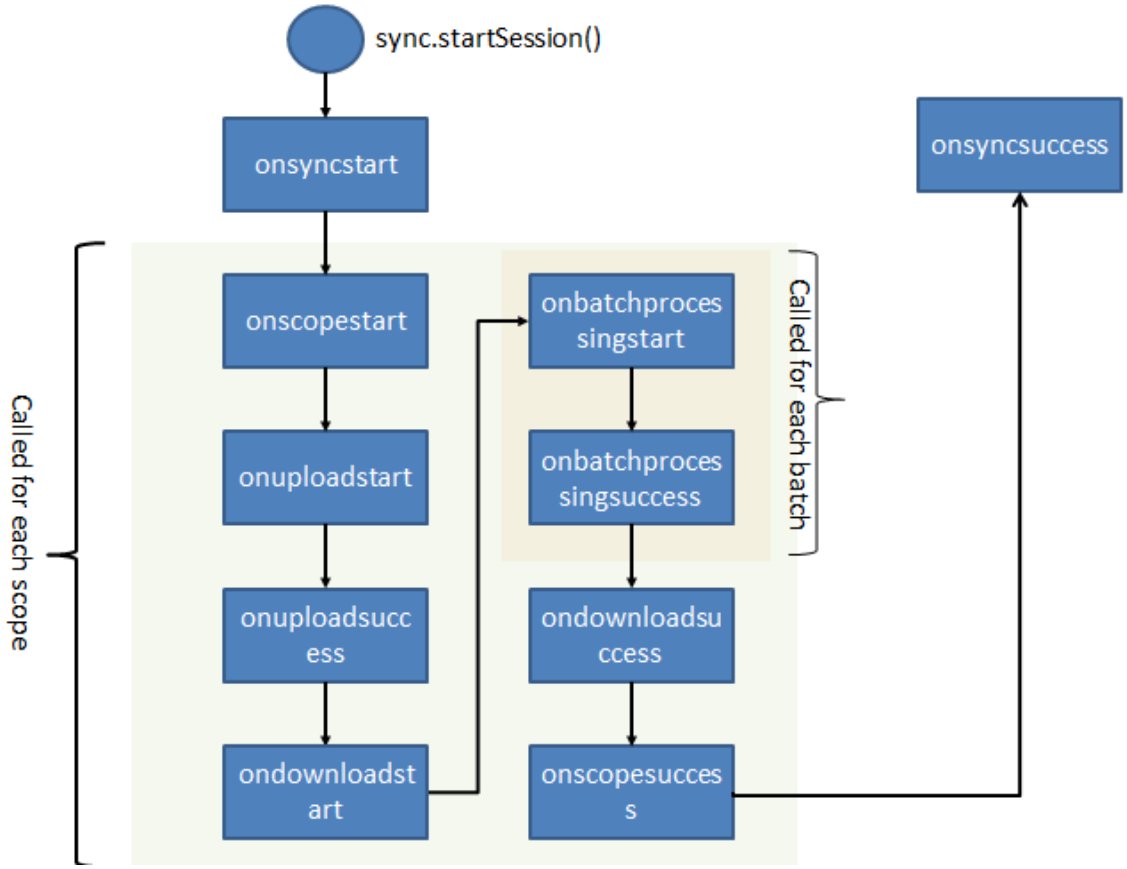
```
function syncInit(){
sync.init(initSuccessCallback , initFailCallback);
}

function initSuccessCallback(){
alert("Sync Initialized");
}
function initFailCallback(res){
alert("Sync Failed" + " with Error Code:" + res.errorCode + ", error
message:"+ res.errorMessage + ", error information:" +
JSON.stringify(res.errorInfo));
}
```

2.2 sync.startSession

This function triggers the synchronization cycle. It receives a set of configuration as input that helps to fine tune the synchronization process to specific application needs.

2.2.1 Client Side Lifecycle Events



2.2.2 Configuration Parameters

Parameter Name [Mandatory/Optional]	Parameter Type	Description
user id [Mandatory]	String	Specifies the user identifier to use to authenticate the user for accessing the Kony Fabric Sync Services
password [Mandatory]	String	Specifies the password to use to authenticate the user to access the Kony Fabric Sync Services

Parameter Name [Mandatory/Optional]	Parameter Type	Description
appid [Mandatory]	String	Application identifier for which the data needs to be synchronized
serverhost [Mandatory]	String	Identifies the server on which the Kony Fabric Sync Services are deployed. Typically a DNS (For Example: `sync.kony.net`) or a String in IP Address format (10.10.10.4). The serverhost is 'undefined' by default.
issecure[Optional]	boolean	Identifies the protocol you should use to connect to server; http or https. If `issecure` is not specified, 'false' is taken by default and `config.issecure` is configured to 'false' by default.
serverport [Optional]	String	Identifies the port on which the Kony Fabric Sync Services are deployed. If port is not specified, port no: 80 is taken as default.

Parameter Name [Mandatory/Optional]	Parameter Type	Description

sessiontasks [Optional]	JSObject/table	You can specify various operations scope level using session tasks:
		
		1. **doupload /dodownload**: Allows the user to control Sync Scopes that you should consider for upload and download. It typical follows the below structure.
		
		`{`
		
		`syncscopename : {doupload:true, dodownload:true}`
		
		`}`
		
		Setting `dodownload` to *true* indicates that user wants to download the latest changes from the Kony Fabric Sync Server and vice-versa.
		
		Setting `doupload` to *true* indicates that the user wants to upload the latest changes in the local tables to the Kony Fabric Sync Server and vice-versa.
		
		2. **`uploaderrorpolicy`**: This indicates what should happen when error occurs while uploading changes to server. There are two options available:
		
		a. `abortonerror`: Abort the upload and stop the sync process for that particular scope.
		
		b. `continueonerror`: Ignore the error, continue the upload and sync process for that particular scope.
		
		It has to follow structure:
		
		`{`
		`syncscope1: {uploaderrorpolicy: "abortonerror"},`
		`syncscope2: {uploaderrorpolicy:`

Parameter Name [Mandatory/Optional]	Parameter Type	Description

sessiontasks [Optional]	JSObject/table	For **Example 1**:
		`config.sessiontasks = { Scope1 : { doupload : true, dodownload : true, uploaderrorpolicy : "continueonerror" }, Scope2 : { doupload : true, dodownload : true, uploaderrorpolicy : "abortonerror" } };`
		For **Example 2**:
		`config.sessiontasks = { Scope1 : { doupload : true, dodownload : false, uploaderrorpolicy : "abortonerror" }, Scope2 : { doupload : true, dodownload : false, uploaderrorpolicy: "continueonerror" } }`

Parameter Name [Mandatory/Optional]	Parameter Type	Description
batchsize	String	Indicates the number of rows that get downloaded in one batch from the server. By default, this value is set to 500. ***Note:*** If you have more than one record with the same lastupdatetimestamp, then irrespective of the batchsize, all the records that have the same lastupdatetimestamp are downloaded in the same batch depending upon the batch cut time.

Parameter Name [Mandatory/Optional]	Parameter Type	Description

filterparams	JSObject/table	Indicates the filter params or values that are used to filter data for a specific user/device. This helps to ensure that only data relevant to a specific user is sent to the device and thus making the whole process more optimized. It typically follows the below structure
		`{ScopeName : [{ SyncObjectName : [{ FilterName : "ProductName", Value1 : "" , Value2 : "" }] }] }`
		For Example:
		`config.filterparams = { PersistentSyncScope : [ { Products : [ { Name : "ProductName", Value1 : "%Cha%" } ] } ] } ;`

Parameter Name [Mandatory/Optional]	Parameter Type	Description
removeafterupload [Optional]	JSObject/table	Indicates that you should delete specified records from local database after successful upload. **Usage**: Suppose that there are three scopes: 1. scope1 with two sync objects syncObject1 and syncObject2) 2. scope2 with two sync objects syncObject3 and syncObject4 3. scope3 with three sync objects syncObject4, syncObject5, and syncObject6. Now, we want to 1. Delete all records in all sync objects in scope1 after upload. 2. Delete none of the records in sync objects in scope2 after upload. 3. Delete only records of syncObject4 and syncObject6 in scope3 after upload. We pass config param to `syn.startsession` like this: `config.removeafterupload = {"scope1":[ ],"scope3": ["syncObject4", "syncObject6"]};` P.S. Currently it is not available at record level.

Parameter Name [Mandatory/Optional]	Parameter Type	Description

onscopestart [Optional]	Function	Indicates the callback that is called before synchronization for a specific scope is started. It is called for each Sync Scope defined in the application. It receives a context object parameter with the below keys:
		`uploadurl [String]`
		`downloadurl [String]`
		`currentScope [String]`
		`lastsynctimestamp [String]`
		`uploadsequencenumber [String]`

onuploadstart [Optional]	Function	Indicates a callback that is called before starting the upload for a specific SyncScope. It is called for each SyncScope defined in the application (unless the specific scope is set, the property `doupload=false` or there is nothing to upload for that scope). It receives a context object with the below keys
		`uploadurl [String]`
		`downloadurl [String]`
		`currentScope [String]`
		`lastsynctimestamp [String]`
		`uploadsequencenumber [String]`
		`uploadRequest [JSObject/table]`

Parameter Name [Mandatory/Optional]	Parameter Type	Description

onuploadsuccess [Optional]	Function	Indicates a callback that is called after the changes for all the objects within a scope are uploaded to the Kony Fabric Sync Server. It receives a context object with the below keys:
		`uploadurl [String]`
		`downloadurl [String]`
		`currentScope [String]`
		`lastsynctimestamp [String]`
		`uploadsequencenumber [String]`
		`uploadcontext [JSObject/table]`
		a. `rowsuploaded [Number]` - total number of rows uploaded
		b. `rowsinserted [Number]` - number of rows inserted
		c. `rowsupdated [Number]` - number of rows updated
		d. `rowsdeleted [Number]` - number of rows deleted
		e. `oobjectlevelinfo [JSObject/table]` - Contains following info for each sync object:
		`syncobject [JSObject/table] : {`
		`rowsuploaded [Number],`
		`rowsinserted[Number],`
		`rowsupdated [Number],`
		`rowsdeleted[Number],`
		`rowsfailedtoupload[Number],`

Parameter Name [Mandatory/Optional]	Parameter Type	Description

ondownloadstart [Optional]	Function	Indicates a callback that is called before downloading the data changes for a specific scope. It receives a context object populated with the below keys:
		`uploadurl [String]`
		`downloadurl [String]`
		`currentScope [String]`
		`lastsynctimestamp [String]`
		`downloadRequest [JSObject/table]`

onbatchprocessingstart [Optional]	Function	Indicates a callback that is called before downloading a batch of changes from the Kony Fabric Sync Server. It is called for each batch that the Server sends.
		`uploadurl [String]`
		`downloadurl [String]`
		`currentScope [String]`
		`lastsynctimestamp [String]`
		`downloadRequest [JSObject/table]`

Parameter Name [Mandatory/Optional]	Parameter Type	Description

onbatchprocessingsuccess [Optional]	Function	Indicates the callback that is called after the client processes the batch. It receives a context object with the below keys:
		`uploadurl [String]`
		`downloadurl [String]`
		`currentScope [String]`
		`lastsynctimestamp [String]`
		1. `batchcontext [JSObject/table]`
		a. `batchrowsdownloaded [Number]-total number of rows uploaded`
		b. `rowsinserted [Number] - number of rows inserted`
		c. `rowsupdated [Number] - number of rows updated`
		d. `rowsdeleted [Number] - number of rows deleted`
		e. `objectlevelinfo [JSObject/table] - Contains following info for each sync object:`
		`syncobject [JSObject/table] :`
		`{ rowsuploaded [Number],`
		`rowsinserted[Number],`
		`rowsupdated[Number],`
		`rowsdeleted[Number],`
		`rowsfailedtoupload [Number],`

Parameter Name [Mandatory/Optional]	Parameter Type	Description
onbatchprocessingsuccess [Optional]	Function	[Following are populated only for Scopes with Sync Strategy OTASync] `ackinsetedrows [Number], ackdeletedrows[Number], ackupdatedrows[Number], acktotalrows[Number], }` `acktotalrows [Number]` - total number of rows acknowledged by the server. Are populated only for Scopes with Sync Strategy OTASync. `ackinsertedrows [Number]` - number of inserted rows acknowledged by the server. Are populated only for Scopes with Sync Strategy OTASync. `ackupdatedrows [Number]` - number of updated rows acknowledged by the server. Are populated only for Scopes with Sync Strategy OTASync.

onscopesuccess [Optional]	Function	Indicates a callback that is called once a specific SyncScope is processed. It is called for each SyncScope defined in the application. It receives a context object with the below keys:
		
		`uploadurl [String]`
		
		`downloadurl [String]`
		
		`currentScope [String]`
		
		`lastsynctimestamp [String]`

Parameter Name [Mandatory/Optional]	Parameter Type	Description

onsyncsuccess [Optional]	Function	Indicates a callback that is called once all the Sync Scopes are successfully processed by the Sync client side libraries. It receives a context object with the below keys:
		`uploadurl [String]`
		`downloadurl [String]`
		`currentScope [String]`
		`lastsynctimestamp [String]`

onsyncerror [Optional]	Function	Indicates the callback that is called for in case the Sync process encounters an error in any of scopes. It receives a context object with the below keys: `errorCode [String]` - error Code `errorMessage [String]` - Message `erroInfo [JSObject/table/null]` - Further information about errors in various scopes
onscopecerror [Optional]	Function	Indicates the callback that is called for error occurred at each scope. It receives a context object with the below keys: `errorCode [String]` - error Code `errorMessage [String]` - Message `erroInfo [JSObject/table/null]` - Further information about error.

Parameter Name [Mandatory/Optional]	Parameter Type	Description
passwordhashalgo [Optional]	Function/String	Indicates the hashing algorithm to be used to encrypt password before sending it to server. By default, passwordhashalgo is configured to SHA256. Following are valid values: 1. If some hashing algorithm is to be used: a. **sha1** b. **sha224** c. **sha256** d. **sha384** e. **sha512** f. **md2** g. **md4** h. **md5** For example, `config.passwordhashalgo = "sha256";` P.S. It purely depends upon the algorithms that Kony platform supports. 2. If plain password is to be sent: **none** For example, `config.passwordhashalgo = "none";`

Parameter Name [Mandatory/Optional]	Parameter Type	Description

passwordhashalgo [Optional]	Function/String	If some user defined function is to be used to create hash: given function
		
		For example,
		
		`config.passwordhashalgo = myHashalgo`
		`function myHashalgo(plainText){`
		`//create Hash`
		`return hash;`
		`}`

invokeservicefunction [Optional]	Function	To Override Network Service Calls

Note: For sync to work with iOS9.x, you need to modify value of the parameter, allow self signed certificates to *YES* in the Xcode.

2.2.3 Signature

sync.startSession(config)

2.2.4 Parameters

- config [JSObject or table] - Mandatory

  Specifies the table to set the configuration parameters.

2.2.5 Platform Availability

Available on all platforms.

2.2.6 Example

```
function syncStart()
{
varconfig = {};

//onXXXXDemo is a function that is called on these callbacks
       config.onsyncstart = onsyncstartDemo;
       config.onscopestart = onscopestartDemo;
       config.onscopeerror = onscopeerrorDemo;
       config.onscopesuccess = onscopesuccessDemo;
       config.onuploadstart = onuploadstartDemo;
       config.onuploadsuccess = onuploadsuccessDemo;
       config.ondownloadstart = ondownloadstartDemo;
       config.ondownloadsuccess = ondownloadsuccessDemo;
       config.onbatchprocessingstart = onbatchprocessingstartDemo;
       config.onbatchprocessingsuccess = onbatchprocessingsuccessDemo;
       config.onsyncsuccess = onsyncsuccessDemo;
       config.onsyncerror = onsyncerrorDemo;

//Selected records need to be downloaded to the device database from
the server.
config.filterparams =
{
PerProv1 : [{Categories : [ { Name : "CategoryID", Value1 :
"1,2"}]}]
}

//Remove data in the specific tables after upload
       config.removeafterupload = {"scope1":[],"scope3":["tab4", "tab6"]};

//To achieve the below said 3 usecases, pass configparam to
```

```
sync.startsession() as above

Example : scope1 (table 1, table 2), scope2 (table 3, table 4),
scope3 (table 4, table 5, table 6)
1. delete all records in all tables in scope1 after upload
2. delete none of the records in tables in scope2 after upload
3. delete only records of table4  and table6 in scope3 after upload

//The below values can be passed by the developer
       config.userid= "syncadmin";
       config.password="SyncAdmin123";
       config.appid = "APPIDNorthwind";
       config.serverhost ="10.10.10.10";
       config.serverport = "8080";

//Configure the size of the batch to be downloaded
config.batchsize= "1000"// Default value is 500

-- Session level tasks to control at scopes
      -- These parameters help to configure Download or Upload only
for a scope
       config.sessiontasks =
       {
       Scope1 : {doupload : true, dodownload : true,
uploaderrorpolicy:"continueonerror"}
       }

// The sync.startSession command after populating the config object
       sync.startSession(config);
```

2.3 sync.startReconciliation

`sync.startReconciliation` API is used to compare primary keys present in backend with application database primary keys for all objects configured in input params. It involves 2 service calls:

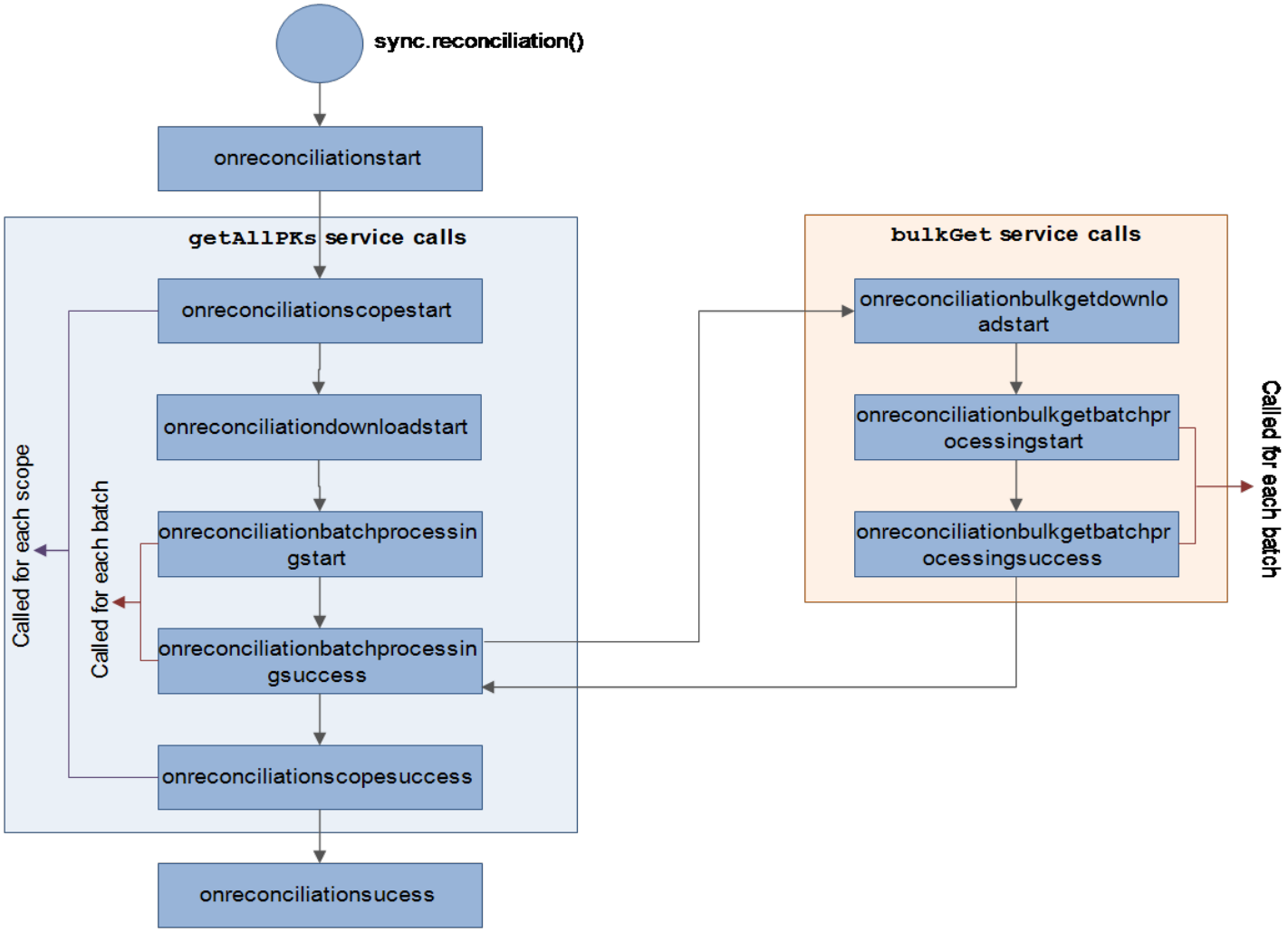
1. `getAllPKs` - Captures all corresponding object PKs from the backend.
2. `bulkGet` - Retrieves the actual record for given PKs.

From SDK, the first service call is `getAllPKs` used to fetch the PK values for corresponding objects. SDK compares the PKs with application database. If any extra records are found in application database, the corresponding record will be deleted. If any record is missing, the actual record will be retrieved through `bulkGet` service call.

> ***Important:*** PK - Primary Key

2.3.1 Workflow

The following diagram illustrates the end-to-end workflow of `sync.startReconciliation` API:


sync.reconciliation()
onreconciliationstart
getAllPKs service calls
onreconciliationscopestart
onreconciliationdownloadstart
onreconciliationbatchprocessingstart
onreconciliationbatchprocessingsuccess
onreconciliationscopesuccess
onreconciliationsucess
Called for each scope
Called for each batch
bulkGet service calls
onreconciliationbulkgetdownloadstart
onreconciliationbulkgetbatchprocessingstart
onreconciliationbulkgetbatchprocessingsuccess
Called for each batch

2.3.2 Configuration Parameters

Callbacks [Mandatory/Optional]	Type	Description
reconciliationc[Mandatory]	[JSObject/ Table]	List of scopes with array of object names which are reconcile. `config.reconciliation = { "<scope1>": ["<object1>", "<object2>"], "<scope2>": [] };` Reconciliation takes place for object1 and object2 in scope1 and all objects present in scope2 as the value is an empty array[ ].
reconciledownloadbatchsize [Optional]	Number	Batch size represents number of primary keys to be downloaded in each batch. ***Note:*** Default value for Blackberry is 50 and for other platforms 500. ***Note:*** Maximum number of parameters to be sent in RPC request for sql server is 2100.So we have to make sure that the `reconciledownloadbatchsize` given in config params should meet this condition.

Callbacks [Mandatory/Optional]	Type	Description
reconcilebulkgetbatchsize [Optional]	Number	Batch size represents number of records (actual record) to be downloaded in each batch. Default value for Blackberry is 50 and for other platforms 500. ***Note:*** Maximum number of parameters to be sent in RPC request for sql server is 2100.So we have to make sure that the `reconcilebulkgetbatchsize` given in config params should meet this condition.

onreconciliationscopestart [Optional]	Function	Indicates a callback that is called before starting the `getAllPks` service call (fetching all the Primary keys) for a specific SyncScope. It is called for each SyncScope defined in the application. It receives a context object with the below keys. JSON object with scope name as 'key' and value as arrays of objects which are going to be reconciled.
		`{`
		`<scopename> : ["<object1>", "<object2>"];`
		`}`

onreconciliationscopesuccess [Optional]	Function	Indicates a callback that is called once a specific SyncScope is processed. It is called for each SyncScope defined in the application. It is just a hook if user wants to perform anything after each scope success.

Callbacks [Mandatory/Optional]	Type	Description
onreconciliationscopeerror [Optional]	Function	Indicates a callback that is called for error occurred at each scope. It receives a context object with the below keys: `errorCode [String]` - error Code `errorMessage [String]` - Message `errorInfo [JSObject/table/null]` - Further information about error.
onreconciliationstart [Optional]	Function	Indicates a callback that is called before starting reconciliation process. It receives a context object which is configured in `sync.startReconciliation` API.
onreconciliationbatchprocessingstart [Optional]	Function	Indicates a callback that is called before downloading a batch of primary keys from Kony Fabric Sync Server. It is called for each batch that the Server sends: 1. `downloadRequest` [JSObject/table] 2. `filterContext`[JSObject/table]: is a download request that has a list of objects with primary key columns.

Callbacks [Mandatory/Optional]	Type	Description
onreconciliationbatchprocessingsuccess [Optional]	Function	Indicates a callback that is called after the client processes the batch. It receives a context object with the below keys. Specified number of records downloads / inserts / deletes for that corresponding batch. 1. `reconcilebatchdownloads` [Number] 2. `reconcilebatchinsertions` [Number] 3. `reconcilebatchdeletions` [Number]

Callbacks [Mandatory/Optional]	Type	Description
onreconciliationsuccess [Optional]	Function	Indicates a callback that is called once all the Sync Scopes are successfully processed (`getAllPks` and `bulkGet`) by the Sync client side libraries. It receives a context object with the below keys: • `reconcileobjectlevelinfo` [JSObject/table] - complete reconciled objects information with no of records downloaded / inserted / deleted. • `reconciletotaldownloads` [Number] - Total no of records downloaded for that corresponding object and will be present in each object. • `reconciletotalinsertions` [Number] - Total no of records inserted in app DB for that corresponding object and will be present in each object. • `reconciletotaldeletions` [Number]- Total no of records deleted in app DBfor that corresponding object and will be present in each object. • `Pendingreconcilescopeswithhistorychanges`[Number] - Scopes which are ignored because of pending changes resided in it for that corresponding object and will be present in each object.

Callbacks [Mandatory/Optional]	Type	Description
onreconciliationerror [Optional]	Function	Indicates a callback that is called if the Sync process encounters an error in any of scopes. It receives a context object with the below keys: • `errorCode` [String] - error Code • `errorMessage` [String] - Message • `erroInfo` [JSObject/table/null] - Further information about error.
onreconciliationdownloadstart [Optional]	Function	Indicates a callback that is called before downloading the PKs for a specific scope. It receives a context object populated and configured in `sync.startReconciliation` API.
onreconciliationbulkgetdownloadstart [Optional]	Function	Indicates a callback that is called before starting the `bulkGet` service call (fetching the actual data for missed primary keys). It receives a context object with the below keys. `downloadRequest` [JSON/Table] `filterContext` [JSON] : It is located inside the `downloadRequest`. It has the list of PK values that are to be downloaded.
onreconciliationbulkgetbatchprocessingstart [Optional]	Function	Indicates a callback that is called before downloading a batch `bulkGet` (fetching the actual data for missed primary keys) from the Kony Fabric Sync Server. It is called for each batch that the Server sends and receives a context object with the below keys.

Callbacks [Mandatory/Optional]	Type	Description
onreconciliationbulkgetbatchprocessingsuccess [Optional]	Function	Indicates a callback that is called after the client processes the `bulkGet` batch. It receives a context object with the below keys: `reconcilebatchinsertions`[Number]: No. of records after processing the batch.

Limitation: If there are any pending changes in any object, then corresponding scope will be ignored in reconciliation process.

> ***Note:*** If any error / failure occurs, reconciliation starts from the scratch and does not have an option to resume its process.

> ***Note:*** For parent child relationship in sync config, it is recommended that the order of passing object names in `config.reconcile` param of `sync.startReconciliation` is first the child object has to be passed and then the parent object .

> ***Note:*** `getAllPKs` is a new operation which is not supported in IDE, so customer needs to map the operation manually.

2.3.3 Example

```
config.reconciliation = {
"scope1": [{ "Account": {} },{"Contact": {}}, {"Opportunity": {}}],
"scope2": [{"Lead": {}}]
};
 config.reconcilebulkgetbatchsize = "2"; // Used for bulkgetService
 config.reconciledownloadbatchsize = "100"; //Used for getAllPk
service

 //The below values can be passed by the developer
```

```
        config.userid= "syncadmin";
        config.password="SyncAdmin123";
        config.appid = "APPIDNorthwind";
        config.serverhost ="10.10.10.10";
        config.serverport = "8080";

         //User defined callbacks as per the requirement
        config.onreconciliationscopestart =
onReconciliationScopeStartCallback;
        config.onreconciliationscopesuccess =
onReconciliationScopeSuccessCallback;
        config.onreconciliationscopeerror =
onEeconciliationScopeErrorCallback;
        config.onreconciliationstart = onReconciliationStartCallback;
        config.onreconciliationbatchprocessingstart =
onReconciliationBatchProcessingStartCallback;
        config.onreconciliationbatchprocessingsuccess =
onReconciliationBatchProcessingSuccessCallback;
        config.onreconciliationsuccess = onReconciliationSuccessCallback;
        config.onreconciliationerror = onRconciliationErrorCallback;
        config.onreconciliationdownloadstart =
onreconciliationdownloadstart;
        config.onreconciliationbulkgetdownloadstart =
onreconciliationbulkgetdownloadstart;
        config.onreconciliationbulkgetbatchprocessingstart =
onreconciliationbulkgetbatchprocessingstart;
config.onreconciliationbulkgetbatchprocessingsuccess =
onreconciliationbulkgetbatchprocessingsuccess
```

2.4 sync.stopSession

You can stop the sync process using the `sync.stopSession` API. If you perform `sync.stopsession` and then `sync.startsession`, sync starts from where it stopped before.

2.4.1 Signature

sync.stopSession(callback)

2.4.2 Platform Availability

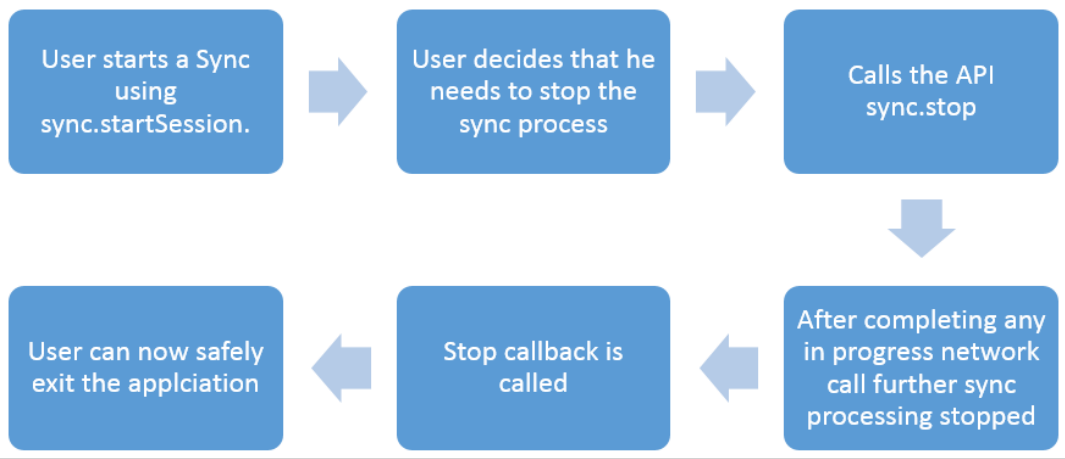
The Stop Sync API is supported on the following platforms:

- iOS
- Android
- Windows

2.4.3 Example

```
sync.stopSession( function ()
{
alert("Sync has been stopped");
});
```

Following illustration shows the end-to-end workflow for the stop sync API:



2.4.4 Requirements

Following are some of the key requirements that need to be enabled for the stop sync process:

- You must be able to call `sync.stopSession` to cancel any in progress sync.
- If the network call is in progress, stop sync will not cancel the process right away. Instead it will wait for the network call to complete and stops triggering further sync calls (next batches or scopes).
- If you call the `sync.stopSession` API multiple times, an error `Session is in Progress` appears and the sync continues.
- If you call the `sync.stopSession` API when sync is not in progress, a sync cycle is triggered and makes network calls in background.
- After the sync is stopped first the pending callbacks (like `onbatchsuccess`, `onscopesuccess` will get called before invoking the stop callback).

2.5 sync.reset

This method resets the device database by removing all the data and restoring it to a state after first sync.init.

2.5.1 Signature

sync.reset(successCallback, errorCallback)

2.5.2 Parameters

- `successCallback[function]` - Optional

  Specifies the function that gets invoked on success

- `errorCallback[function]` - Optional

  Specifies the function that gets invoked on error

2.5.3 Platform Availability

Available on all platforms mentioned.

2.5.4 Example

```
function syncReset(){
sync.reset(resetSuccessCallback,resetFailCallback );
}

function resetSuccessCallback(){
alert("Sync reset successfully.");
}

function resetFailCallback(res){
alert("Sync reset failed" + " with Error Code:" + res.errorCode + ",
error message:"+ res.errorMessage + ", error information:" +
JSON.stringify(res.errorInfo));
}
```

2.6 sync.rollbackPendingLocalChanges

This method resets the data to last synchronized state. You can use this method to undo the all the client side changes.

2.6.1 Signature

sync.rollbackPendingLocalChanges(successCallback, errorCallback)

2.6.2 Parameters

- successCallback**[function] - Optional**

  Specifies the function that gets invoked on success

- errorCallback**[function] - Optional**

  Specifies the function that gets invoked on error

2.6.3 Platform Availability

Available on all platforms mentioned.

2.6.4 Example

```
function syncRollBack(){
sync.rollbackPendingLocalChanges
(rollbackSuccessCallback,rollbackFailCallback);
}

function rollbackSuccessCallback(){
alert("All recent changes rollbacked successfully");
}
```

```
function rollbackFailCallback(res)
{
alert("Problem occurred in rollbacking recent changes." + " with
Error Code:" + res.errorCode + ",
error message:"+ res.errorMessage + ", error information:" +
JSON.stringify(res.errorInfo));
}
```

2.7 sync.getPendingAcknowledgement

This method retrieves all the rows across all the SyncObjects that are waiting for acknowledgement from the Sync Server. You use the method for persistent sync strategy.

> ***Note:*** For OTA strategy, sync.getPendingAcknowledgement API always returns *0*.

2.7.1 Signature

sync.getPendingAcknowledgement*(successCallback, errorCallback)*

2.7.2 Parameters

- successCallback[function] - Optional

  Specifies the function that gets invoked on success

- errorCallback[function] - Optional

  Specifies the function that gets invoked on error

2.7.3 Platform Availability

Available on all platforms mentioned.

2.7.4 Example

```
function fetchPendingAcknowledgement(){
sync.getPendingAcknowledgement(ackSuccessCallback ,ackFailCallback)
}

function ackSuccessCallback(res){
/* res contains all information regarding the Pending
Acknowledgements.
It contains the total number of Pending Acknowledgements and
information regarding the tables whose Acknowledgment is pending */
     alert("Total Number of Pending Acknowledgements: " + res.
totalCount);
     alert("Data Regarding Tables waiting for Acknowledgement:"
     +res.totalData);
alert("Sync Pending Acknowledgement Success");
}

function ackFailCallback(res){
alert("Sync Pending Acknowledgement Failed" + " with Error Code:" +
res.errorCode + ", error message:"+ res.errorMessage + ", error
information:" + JSON.stringify(res.errorInfo));
}
```

2.8 sync.getPendingUpload

This method retrieves all the rows across all the SyncObjects that are pending to be uploaded to the Kony Fabric SyncServer.

2.8.1 Signature

sync.getPendingUpload*(successCallback, errorCallback)*

2.8.2 Parameters

- successCallback**[function] - Optional**

  Specifies the function that gets invoked on success

- errorCallback**[function] - Optional**

  Specifies the function that gets invoked on error

2.8.3 Platform Availability

Available on all platforms mentioned.

2.8.4 Example

```
function fetchPendingUpload(){
sync.getPendingUpload(pendingUploadSuccessCallback,
pendingUploadFailCallback)
}

function pendingUploadSuccessCallback(res){
     /* res contains all information regarding the Pending Uploads.
It contains the total number of Pending Uploads and information
regarding the tables whose upload is pending */
     alert("Total Number of Pending uploads: " + res. totalCount);
     alert("Data Regarding Tables waiting to get uploaded:"
+res.totalData);
     alert("Sync Pending Upload Success");
}

function pendingUploadFailCallback(res){
alert("Sync Pending Upload Failed" + " with Error Code:" +
res.errorCode + ", error message:"+ res.errorMessage + ", error
```

```
information:" + JSON.stringify(res.errorInfo));
}
```

2.9 sync.getDeferredUpload

This ORM API helps to retrieve the list of rows that are set for hold at sync level.

2.9.1 Signature

sync.getDeferredUpload*(successCallback, errorCallback)*

2.9.2 Parameters

- successCallback**[function] - Optional**

  Specifies the function that gets invoked on success

- errorCallback**[function] - Optional**

  Specifies the function that gets invoked on error

2.9.3 Platform Availability

Available on all platforms mentioned.

2.9.4 Example

```
function GetDeferredUpload()
{
sync.getDeferredUpload(deferredUploadSuccessCallback,
deferredUploadFailCallback)
}

function deferredUploadSuccessCallback(res)
```

```
{
 /* res contains all information regarding the deferred Uploads. It
contains the total number of Deferred Uploads and information
regarding the tables whose upload is deferred */
     alert("Total Number of Deferred Uploads: " + res. totalCount);
     alert("Data Regarding Tables whose upload is deferred:"
+res.totalData);
     alert("Deferred Upload Success");
}

function deferredUploadFailCallback(res){
alert("Deferred Upload Failed" + " with Error Code:" + res.errorCode
+ ", error message:"+ res.errorMessage + ", error information:" +
JSON.stringify(res.errorInfo));
}
```

Rev	Author	Edits
7.0	GS	GS

3. Background Sync

3.1 Introduction

In Kony Sync (5.6.x) if the application goes to background, any network calls or JavaScript execution is stopped. Here you have to ensure that the application is in the foreground if the sync is in progress. If you trigger an initial sync that takes minutes then you have no choice but to observe the progress.

With the Background Sync feature, you can switch to other apps and can expect the sync to progress even when the application is in background. In iOS 7, a background sync feature is added. Refer to the following URL for more information on the background capabilities added in iOS.

https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/BackgroundExecution/BackgroundExecution.html

Following are the supported platforms for Background Sync:

- iOS 7, iOS 8 and iOS 9 (Kony Visualizer Enterprise and Native SDK)
- Android 4.x and above (Kony Visualizer Enterprise and Native SDK)

3.2 NSURLSession API

The main component of background transfers in iOS 7 is the `NSURLSession` API.

The `NSURLSession` API allows you to:

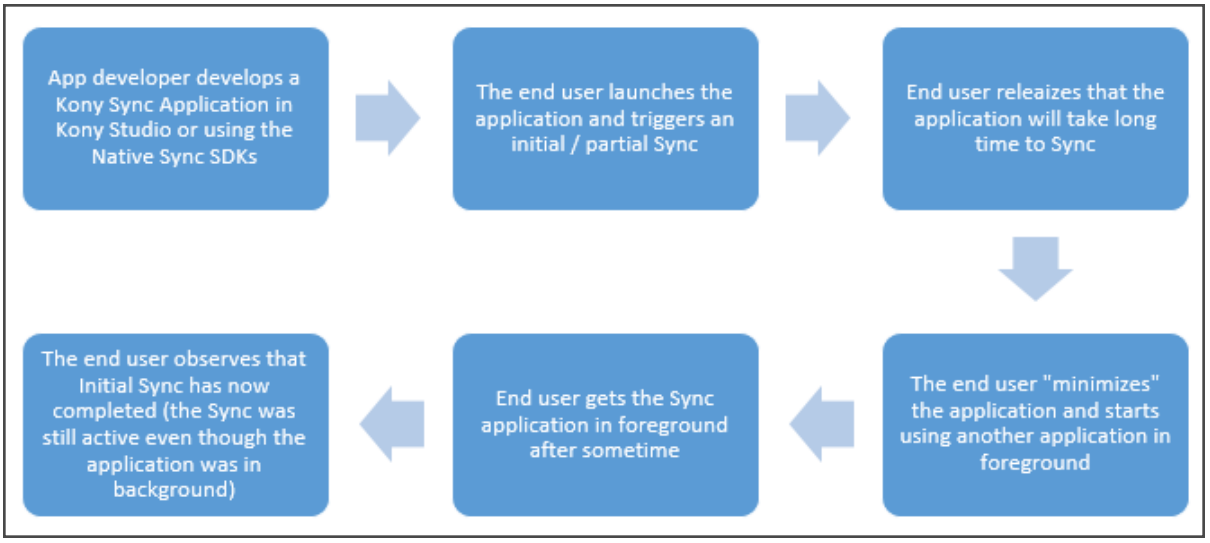
- Transfer content through network and device interruptions.
- Upload and download large files (Background Transfer Service).

The `NSURLSession` API is a powerful API for transferring content over the network. It provides a set of tools to handle the transfer of data through network interruptions and changes in application state.

The `NSURLSession` API creates one or more sessions, that calls tasks to shuttle blocks of related data across the network. Refer to the following links for more information on NSURLSession:

- https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/URLLoadingSystem/Articles/UsingNSURLSession.html
- https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/URLLoadingSystem/NSURLSessionConcepts/NSURLSessionConcepts.html#//apple_ref/doc/uid/10000165i-CH2-SW1

Following illustration shows the background sync process:



3.3 Key Scenarios

The Background Sync capability enables the following scenarios if the application is in background:

3.3.1 sync.startsession Continues to Execute

If you move the application to background, any current network calls made through `sync.startSession` are terminated. With the background sync capability, you must observe that the application continues to download or upload data even if the application is background.

3.3.2 Trigger a Sync Periodically (for iOS 7)

The need to refresh the sync data in background even if the app is not launched or is in background can be achieved using Timer API. The Timer API keeps executing even if the application is in background.

Even though iOS 7 provides a Background Fetch APIs there are several limitations to it. It limits the amount of time the app needs to execute (approximately ten minutes).

3.3.3 Enabling Periodic Sync

You can enable periodic sync using the `kony.backgroundjob.registerBackgroundFetch` API. The developer can override the existing syncStartSession generated from IDE as shown in the following code snippet.

```
function startsSyncCallback()
{
startsSync();
var completionStatus = constants.BACKGROUND_TASK_STATUS_NO_NEW_DATA
}
var fetchInterval = 50;
kony.backgroundjob.registerBackgroundFetch
(startsSyncCallback,fetchInterval);
```

Fetch Interval is an optional parameter with the default value `constants.BACKGROUND_TASK_FETCH_INTERVAL_MINIMUM`, which indicates that the system decides the fetch interval depending on the usage prediction for the app. It is advised to provide a Fetch interval, as decisions made by operating systems are unpredictable.

`kony.backgroundjob.setBackgroundFetchCompletionStatus` API must be called mandatorily for background fetch. This method intimates the system of the completion status of the background fetch job that is scheduled. Executing this call tells the system that it can move the app back to the suspended state and evaluate its power usage.

The `completionStatus` can have three possible values:

1. constants.BACKGROUND_TASK_STATUS_NEW_DATA

   This tells the system that the fetch was successful and internally the system updates the apps user interface (if the fetch resulted in user interface change) in the background. This new user interface is presented after the user brings the app into foreground. The snapshot of the new user interface is also presented when the user tries to switch apps using the app switcher.

2. constants.BACKGROUND_TASK_STATUS_FAILED

   This tells the system that the fetch was unsuccessful. User interface is not updated and the task will be run later based on the available system resources.

3. constants.BACKGROUND_TASK_STATUS_NO_NEW_DATA

   This intimates the system that the fetch did not result in any new data, user interface is not updated and the job is run after any point later in time but not before fetchInterval.

The reason for using `constants.BACKGROUND_TASK_STATUS_NO_NEW_DATA` instead of `constants.BACKGROUND_TASK_STATUS_NEW_DATA` is:

1. The Background Transfer API for long running network calls is asynchronous and the API returns before the callback is completed, hence the network call works anomalously.
2. The background fetch allows only thirty seconds of execution time as specified by iOS documentation. Hence by using no new data you are basically completing the callback in less than thirty seconds and allowing the long running network call (triggered by background transfer to run independently).

This solves two problems, first sync cycle is fired periodically depending on the scheduling frequency decided by the operating system, second the long running network call runs independently of the background fetch allowing the user to view data while sync is in progress.

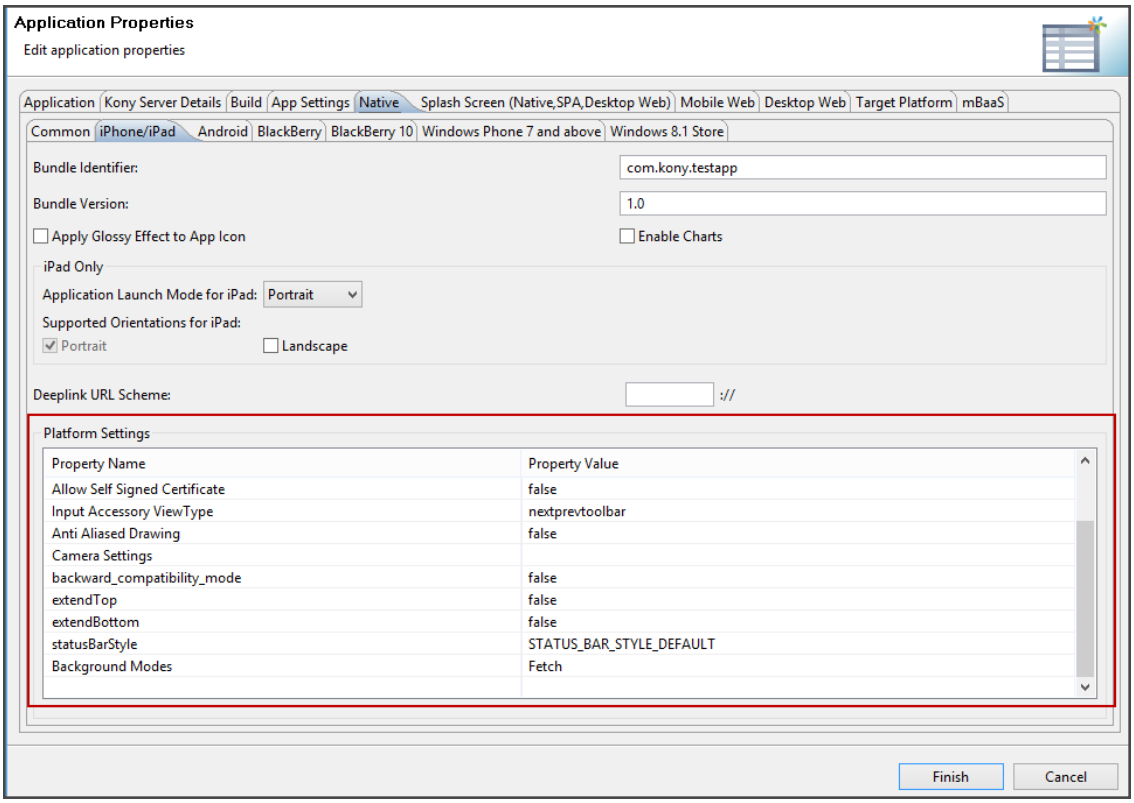
3.3.3.1 Steps to Enable Periodic Sync

To enable periodic sync when the application is in background, following steps must be configured:

1. Kony Visualizer Project Settings
2. Device Settings

Kony Visualizer Project Settings

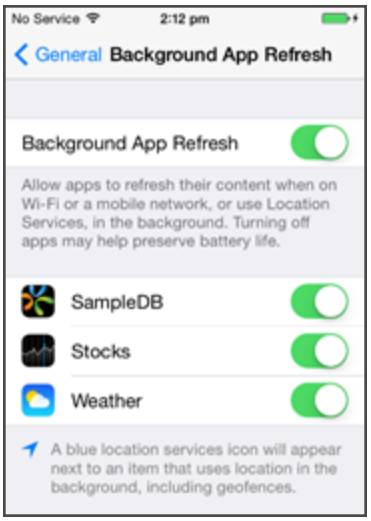
Before building the application, right-click on the application select **Properties > Select Native Tab > Under Native select iPhone/iPad**. Under **Platform Settings**, in **Background Modes** option select **Fetch**. (Default is Disabled)



Device Settings

In iOS 7 and above, go to **Settings > General > BackgroundAppRefresh** and enable the option.

This is required for background fetch to work, as the apps listed under **BackgroundAppRefresh** will be able to use background fetch API.



3.4 Limitations

In iOS, the scheduling of the background jobs is determined by the operating system. Depending upon the execution history, the background jobs can get scheduled. You may observe unusual delays in sync process when running in background. The scheduling of these calls is determined by the operating systems and the nature is unpredictable.

Following are some observations while performing integration testing on the device:

- Background fetch will only invoke the app when it is background, if the app is in foreground it does not fire.
- When the app is used for first time on a device, the time taken to fire background fetch is unknown (depends on the operating system) but it is a significant amount of time. Frequency of successive fetches fired from the app is decided by the operating system.
- **Time it takes to complete the entire Sync**: When the application is background the NSURLSession API (as defined in iOS 7.x) is used to ensure that network calls are not disconnected when the app moves to background.

    - However in background it is the operating system's privilege to schedule the network calls. Depending upon the operating system or device you can see that sync running in background takes hours of time to complete (the same will take few minutes when running in foreground). The operating system also looks at historical data to schedule the network calls requested by the application.

- **Network timeout is not triggered when running in background**: When the network call times out when running in background the operating system does not raise a notification until you get the app in foreground.
- **iOS 8.1 and network caveats**: In general, if network is turned off while a network call is in progress the operating system does not notify of an error in iOS. From the application's perspective the application seems to be stuck and no error is thrown back the user.

4. Object Level

This set of APIs helps the developer to perform various ORM operations at a SyncObject level in the application.

To take advantage of object oriented programming in JavaScript, some of the APIs are mentioned in object oriented way. So, as far as JavaScript is concerned, you can classify ORMs as:

- Instance APIs
- Static APIs

1. Instance APIs

APIs that involve operations on single row can be invoked by standard JavaScript object as well as traditional way.

- Object Oriented Way:

For Example;

```
var product = new Product();
product.productId = 123
product.name = "test";
product. updateByPK (successcallback, error callback);
```

- Traditional Way:

For Example;

```
Product. updateByPK ({productId :123}, { name:"test" },
successcallback, error callback);
```

2. Static APIs

APIs that involve operations on multiple rows can only be invoked using traditional way.

For Example;

```
Product. getAll ( successcallback, error callback);
```

> ***Note:*** Some of the ORMs described below have an optional markForUpload parameter that you can use to allow or defer the change made by the ORM from being uploaded to the server. If the parameter is not specified, it is treated as true, by default and the changes are reflected in the server after sync is performed. If parameter is specified as *false*, then the change that the ORM made is deferred from upload to server until marked for upload using either the markForUpload or markForUploadByPK ORM.

The convention <object> below indicates the GlobalName of the SyncObject as specified in the Kony Fabric Sync IDE / SyncConfig file.

4.1 <object>.getAll (Static)

This API retrieves all instances of SyncObject present in local database with given limit and offset. Limit indicates the number of records to be retrieved and offset indicates number of rows to be ignored before returning the records.

4.1.1 Signature

*<object>.getAll(**successCallback, errorCallback,orderByMap,limit,offset**)*

4.1.2 Parameters

- successCallback[function] - Optional

Specifies the function that gets invoked on success

- errorCallback[function] - Optional

Specifies the function that gets invoked on error

- orderByMap [Array] - Optional

Specifies the way the data should be arranged in ascending/descending order of column name.

Refer to the Example

- limit [Parameter] - Optional

Specifies the limit of number of records to be displayed (used for pagination/lazy loading).

Refer to the Example

- offset[Parameter] - Optional

Specifies the number of rows to be ignored before returning the records used for pagination/lazy loading

Refer to the Example

4.1.3 Platform Availability

Available on all platforms mentioned.

4.1.4 Examples

getAll

(General) Example:

```
function syncGetAll(){
Account.getAll(showAccSuccessCallback,showAccFailCallback)
}

function showAccSuccessCallback(res){
    //res contains the data of all records of Account table
      alert("Get all Accounts success");
}

function showAccFailCallback(res){
```

```
alert("Get all Accounts failed" + " with Error Code:" +
res.errorCode + ", error message:"+ res.errorMessage + ", error
information:" + JSON.stringify(res.errorInfo));
}
```

getAll

(orderByMap) Example:

```
/* Define a orderByMap array to arrange the data in Categories table
in descending order of CategoryID and ascending order of
CategoryName */
 function CategoryGetAll(){
 var orderByMap = []
 orderByMap [0] = {};
 orderByMap [0].key = "CategoryID";
 orderByMap [0].sortType ="desc";
 orderByMap [1] = {};
 orderByMap [1].key = "CategoryName";
 orderByMap [1].sortType ="asc";
 Categories.getAll
(showCatSuccessCallback,showCatFailCallback,orderByMap);
}
function showCatSuccessCallback(res){
//res contains the data of Categories arranged in the way specified
alert("Get all Categories success");
//do something with the data contained in res
}
function showCatFailCallback(res){
  alert("Get all Accounts failed" + " with Error Code:" +
res.errorCode + ", error message:"+ res.errorMessage + ", error
information:" + JSON.stringify(res.errorInfo));
}
```

getAll

(limit and offset) Example:

```
/*assume that Categories table has 30 records. Specifying limit 20
and offset 5 returns 20 records starting from 5th record. Thus,
records from 5-25 are returned */
function CategoryGetAll(){
//specify limit as 20
var limit = 20;
//specify offset as 5
var offset = 5;
Categories.getAll
(showCatSuccessCallback,showCatFailCallback,null,limit,offset);
}
function showCatSuccessCallback(res){
//res contains the data of Categories table specified by limit and
offset
alert("Get all Categories success");
//do something with the data contained in res
}
function showCatFailCallback(res){
alert("Get all Accounts failed" + " with Error Code:" +
res.errorCode + ", error message:"+ res.errorMessage + ", error
information:" + JSON.stringify(res.errorInfo));
}
```

4.2 <object>.getAllDetailsByPK (Instance)

This method retrieves a row using primary key from the local database.

4.2.1 Signature

*<object>getAllDetailsByPK(**pk, successCallback, errorCallback**)*

4.2.2 Parameters

- pk[number/JSObject/Table]- Mandatory

  Specify the primary key of the object using which the respective row data needs to be fetched from the database on the particular object. For composite primary keys it has to be a table/JSObject.

- successCallback[function] - Optional

  Specifies the function that gets invoked on success

- errorCallback[function] - Optional

  Specifies the function that gets invoked on error

4.2.3 Platform Availability

Available on all platforms mentioned.

4.2.4 Example

```
//Static Way
function SyncGetAllDetailsByPK(){
       Product.getAllDetailsByPK
({ProductId:123},successCallback,errorFailCallback);
}

//OOP Way
function SyncGetAllDetailsByPK(){
product = new Product();
product.ProductId = 123;
product.getAllDetailsByPK(successCallback,errorFailCallback);
}
function successCallback(res){
```

```
    //res contains the details of product record specified by pk
alert("Get All Product Details By Primary Key Success");

   //do something with data contained in res
}

function errorFailCallback (res){
alert("Get All Details By Primary Key Failed + " with Error Code:" +
res.errorCode + ", error message:"+
res.errorMessage + ", error information:" + JSON.stringify
(res.errorInfo));
}
```

4.2.5 Example 2

```
//Static Way with single PRIMARY KEY
    function SyncGetAllProductDetailsByPK(){
        Product.getAllDetailsByPK
({ProductId:123},successCallback,errorFailCallback);
                // OR it can be used as
        Product.getAllDetailsByPK
(123,successCallback,errorFailCallback);
    }

//OOP Way  with single PRIMARY KEY
    function SyncGetProductDetailsByPK(){
        product = new Product();
        product.ProductId = 123;
        product.getAllDetailsByPK
(successCallback,errorFailCallback);
    }
```

```
//Static Way with composite PRIMARY KEY
    var pks = { ProductId :123,
                OrderID : 543 };

    function SyncGetAllOrderDetailsByPK(){
        OrderDetails.getAllDetailsByPK
(pks,successCallback,errorFailCallback);
    }

//OOP Way with composite PRIMARY KEY
    function SyncGetOrderDetailsByPK(){
        mOrderDetails = new OrderDetails();
        mOrderDetails.ProductId = 123;
        mOrderDetails.OrderID = 543;
        mOrderDetails.getAllDetailsByPK
(successCallback,errorFailCallback);
    }

 // CallBacks for getAllDetailsByPK calls
    //success CallBack
    function successCallback(res){
            //res contains the details of product record specified
by pk
        alert("Get All Details By Primary Key Success");
            //do something with data contained in res
    }
    //failure CallBack
    function errorFailCallback (res){
        alert("Get All Details By Primary Key Failed + " with Error
Code:" + res.errorCode + ", error message:"+
        res.errorMessage + ", error information:" + JSON.stringify
(res.errorInfo));
    }
```

4.3 <object>.markForUpload (Static)

This API marks instance(s) of sync object matching given where clause for upload. This enables deferred records to merge with the enterprise data source in the next Sync.

4.3.1 Signature

*<object>.markForUpload(**whereClause, successcallback, errorcallback**)*

4.3.2 Parameters

- whereClause**[String]- Mandatory**

  Specifies the whereclause to markForUpload.

- successCallback**[function] - Optional**

  Specifies the function that gets invoked on success

- errorCallback[function] - Optional

  Specifies the function that gets invoked on error

4.3.3 Platform Availability

Available on all platforms mentioned.

4.3.4 Example

```
function MarkForUpload(whereClause, successCallback,
errorFailCallback){
       Product.markForUpload("where UnitPrice > 10", successCallback ,
```

```
errorFailCallback);
}
function successCallback(){
       alert("Product(s)Marked for Upload");
}
function errorFailCallback (res){
       alert("ProductsMarked for upload Failed" + " with Error Code:" +
res.errorCode + ", error message:"+ res.errorMessage + ", error
information:" + JSON.stringify(res.errorInfo));
}
```

4.4 <object>.markForUploadByPK (Instance)

This ORM API marks instance of sync object with given primary key for upload. This enables deferred records to merge with the enterprise data source in the next Sync.

4.4.1 Signature

*<object>.markForUploadByPK(**pk, successcallback, errorcallback**)*

4.4.2 Parameters

- pk**[number/JSObject/Table] - Mandatory**

  Specifies the whereclause to markForUpload.

- successCallback**[function] - Optional**

  Specifies the function that gets invoked on success

- errorCallback**[function] - Optional**

  Specifies the function that gets invoked on error

4.4.3 Platform Availability

Available on all platforms mentioned.

4.4.4 Example

```
//static way
function markForUploadByPK(){
       Product.markForUploadByPK(123, successCallback, errorFailCallback);
}
//OOP Way
function markForUploadByPK(){
product = new Product();
product.ProductId = 123;
       Product.markForUploadByPK(successCallback, errorFailCallback);
}

function successCallback(){
       alert("Product Marked for Upload);
}
function errorFailCallback(res){
       alert("Products Marked for upload Failed" + " with Error Code:" +
res.errorCode + ", error message:"+ res.errorMessage + ", error
information:" + JSON.stringify(res.errorInfo));
}
```

4.5 <object>.create (Instance)

This API creates a new instance of sync object in the local Database. The new record is merged with the enterprise data source in the next Sync.

4.5.1 Signature

*<object>.create (**object, succCallback, errorCallback**)*

4.5.2 Parameters

- Object **[jsobject/table] - Mandatory**

  Specify the object that you need to create in the database

- successCallback**[function] - Optional**

  Specifies the function that gets invoked on success

- errorCallback**[function] - Optional**

  Specifies the function that gets invoked on error

4.5.3 Platform Availability

Available on all platforms mentioned.

4.5.4 Example

```
//Static Way
function CreateProduct(){
var objectProduct = {ProductName:"xyz", UnitPrice:12.34};
       Product.create(objectProduct, successCallback, errorFailCallback,
true);
}
//OOP Way
function CreateProduct(){
product = new Product();
product.ProductName = "xyz";
product.UnitPrice = 12.34;
```

```
Product.create(successCallback, errorFailCallback);
}

function successCallback(){
       alert("Product Created Successfully");
}
function errorFailCallback (res){
       alert("Product Creation Failed" + " with Error Code:" +
       res.errorCode + ", error message:"+ res.errorMessage + ",
error    information:" + JSON.stringify(res.errorInfo));
}
```

4.6 <object>.updateByPK(Instance)

This method updates sync object using primary key in the local Database. The update is merged with the enterprise data source in the next Sync.

4.6.1 Signature

*<object>.updateByPK(**pk, succCallback, errorCallback, markForUpload**)*

4.6.2 Parameters

- pk**[number/jsobject/table] - Mandatory**

  Specify the primary key of the object using which the respective row data needs to be updated in the database. For composite primary keys it has to be a table or JSObject.

- successCallback**[function] - Optional**

  Specifies the function that gets invoked on success

- errorCallback**[function] - Optional**

  Specifies the function that gets invoked on error

4.6.3 Platform Availability

Available on all platforms mentioned.

4.6.4 Example

```
//Static Way
function updateProduct(){
var objectProduct = {ProductName:"xyz", UnitPrice:12.34};
       Product.updateByPK(123, objectProduct, successCallback,
errorFailCallback, true);
}
// Object Oriented Way
function updateProduct (){
product = new Product();
product.ProductName = "xyz";
product.UnitPrice = 12.34;
product.ProductId = 123;
Product.updateByPK(successCallback, errorFailCallback);
}

function successCallback(){
       alert("Product updated Successfully");
}

function errorFailCallback (res){
       alert("Product updating failed" + " with Error Code:"
+ res.errorCode + ", error message:"+ res.errorMessage + ", error
information:" + JSON.stringify(res.errorInfo));
}
```

Note: An Object which was created with markForUpload as false, can not be updated with `markForUpload` value as true. If you want to make `markForUpload` value as true, `markForUploadByPK` API is used to change the value.

4.7 <object>.update(Static)

This method updates sync object using primary key in the local Database. The update is merged with the enterprise data source in the next Sync.

4.7.1 Signature

*<object>.update (**whereClause, succCallback, errorCallback, markForUpload**)*

4.7.2 Parameters

- whereclause**[String] - Mandatory**

  Specify the string using which the data from database to be fetched.

- successCallback**[function] - Optional**

  Specifies the function that gets invoked on success

- errorCallback**[function] - Optional**

  Specifies the function that gets invoked on error

4.7.3 Platform Availability

Available on all platforms mentioned.

4.7.4 Example

```
varobjectProduct = {ProductName:"xyzUpdated", UnitPrice:15.34};
function UpdateProduct(){
Product.update("where UnitPrice> 10", objectProduct, successCallback
, errorFailCallback, markForUpload)
}
function successCallback(){
       alert("Update Product Success");
}
function errorFailCallback(res){
       alert("Update Product Failed" + " with Error Code:"
+ res.errorCode + ", error message:"+ res.errorMessage + ", error
information:" + JSON.stringify(res.errorInfo));
}
```

> ***Note:*** An Object which was created with `markForUpload` as `false`, can not be updated with `markForUplad` value as `true`. If you want to make `markForUpload` value as `true`, `markForUpload` API is used to change the value.

4.8 <object>.deleteByPK (Instance)

This API deletes sync objects using primary key from the local Database. The record is deleted from the enterprise data source in the next Sync.

4.8.1 Signature

*<object>.deleteByPK(**pk, succCallback, errorCallback, markForUpload**)*

4.8.2 Parameters

- pk**[number/jsobject/table] - Mandatory**

  Specify the primary key of object using which the respective row data needs to be deleted in the database.

- successCallback**[function] - Optional**

  Specifies the function that gets invoked on success

- errorCallback**[function] - Optional**

  Specifies the function that gets invoked on error

4.8.3 Platform Availability

Available on all platforms mentioned.

4.8.4 Example

```
//Static Way
function deleteProduct(){
Product. deleteByPK (123, successCallback, errorFailCallback, true);
}
//OOP Way
function updateProduct (){
product = new Product();
product.ProductName = "xyz";
Product.deleteByPK (successCallback, errorFailCallback);
}

function successCallback(){
       alert("Product deleted Successfully");
}
```

```
function errorFailCallback(res){
       alert("Deleted Product By Primary Key Failed + " with Error Code:"
+ res.errorCode + ", error message:"+ res.errorMessage + ", error
information:" + JSON.stringify(res.errorInfo));
}
```

4.9 <object>.remove(Static)

This API deletes sync object(s) using where clause from the local Database. The record(s) are deleted from the enterprise datasource in the next Sync.

4.9.1 Signature

*<object>.remove(**whereclause, succCallback, errorCallback, markForUpload**)*

4.9.2 Parameters

- whereclause**[String] - Mandatory**

  Specify the string using which the data from database to be fetched.

- successCallback**[function] - Optional**

  Specifies the function that gets invoked on success

- errorCallback**[function] - Optional**

  Specifies the function that gets invoked on error

4.9.3 Platform Availability

Available on all platforms mentioned.

4.9.4 Example

```
function DeleteProduct(){
Product.remove(("where ProductId>100", successCallback ,
errorFailCallback, markForUpload)
}

function successCallback(){
       alert("Product Deleted Successfully");
}

function errorFailCallback (res){
       alert("Product Delete Failed" + " with Error Code:" + res.errorCode
+ ", error message:"+ res.errorMessage + ", error information:" +
JSON.stringify(res.errorInfo));
}
```

4.10 <object>.getPendingUpload(Static)

This API retrieves instance(s) of sync object(s) pending for upload. Records are marked for pending upload if they are updated or created locally and the changes are merged with enterprise data source.

4.10.1 Signature

*<object>.getPendingUpload(**succCallback, errorCallback**)*

4.10.2 Parameters

- successCallback**[function] - Optional**

  Specifies the function that gets invoked on success

- errorCallback**[function] - Optional**

  Specifies the function that gets invoked on error

4.10.3 Platform Availability

Available on all platforms mentioned.

4.10.4 Example

```
function getPendingUpload(){
       Product.getPendingUpload(successCallback, errorFailCallback)
}

function successCallback(){
       alert("Get Pending Upload Success");
}

function errorFailCallback (res){
       alert("Get Pending Upload Failed" + " with Error Code:"
+ res.errorCode + ", error message:"+ res.errorMessage + ", error
information:" + JSON.stringify(res.errorInfo));
}
```

4.11 <object>.getPendingAcknowledgement(Static)

This API Retrieves instance(s) of sync object pending for acknowledgement. This is relevant when the Sync Object is part of the SyncScope whose SyncStrategy is PersistentSync. In persistent Sync, the records in the local database are put into a pending acknowledgement state after an upload.

4.11.1 Signature

*<object>.getPendingAcknowledgement(**succCallback, errorCallback**)*

4.11.2 Parameters

- successCallback**[function] - Optional**

  Specifies the function that gets invoked on success

- errorCallback**[function] - Optional**

  Specifies the function that gets invoked on error

4.11.3 Platform Availability

Available on all platforms mentioned.

4.11.4 Example

```
function getPendingAcknowledgement(){
	Product.getPendingAcknowledgement(successCallback ,
errorFailCallback)
}

function successCallback(){
	alert("Get Pending Acknowledgement Success");
}

function errorFailCallback (res)
{
	alert("Get Pending Acknowledgement Failed" + " with Error Code:" +
res.errorCode + ", error message:"+ res.errorMessage + ", error
information:" + JSON.stringify(res.errorInfo));
}
```

4.12 <object>.rollbackPendingLocalChanges(Static)

This API rollbacks all changes to sync object in local database to last synced state.

4.12.1 Signature

*<object>.rollbackPendingLocalChanges(**succCallback, errorCallback**)*

4.12.2 Parameters

- successCallback**[function] - Optional**

  Specifies the function that gets invoked on success

- errorCallback**[function] - Optional**

  Specifies the function that gets invoked on error

4.12.3 Platform Availability

Available on all platforms mentioned.

4.12.4 Example

```
function RollbackPendingLocalChanges(){
       Product.rollbackPendingLocalChanges(successCallback,
errorFailCallback)
}

function successCallback(){
       alert("Rollback Pending Local Changes Success");
}

function errorFailCallback(res){
       alert("Rollback Pending Local Changes Failed" + " with Error
 Code:" + res.errorCode + ", error message:"+ res.errorMessage + ",
error information:" + JSON.stringify(res.errorInfo));
}
```

4.13 <object>.rollbackPendingLocalChangesByPK(Instance)

This API rollbacks changes to sync object with given primary key in local database to last synced state.

4.13.1 Signature

*<object>.rollbackPendingLocalChangesByPK(**pk, succCallback, errorCallback**)*

4.13.2 Parameters

- pk[**Number/jsobject/table**] - **Mandatory**

  Specify the primary key of the object using which the respective row data needs to be rollback in the database.

- successCallback**[function] - Optional**

  Specifies the function that gets invoked on success

- errorCallback**[function] - Optional**

  Specifies the function that gets invoked on error

4.13.3 Platform Availability

Available on all platforms mentioned.

4.13.4 Example

```
//Static Way
function rollbackPendingLocalChangesByPK (){
Product.rollbackPendingLocalChangesByPK(123, successCallback,
errorFailCallback);
}
//OOP Way
```

```
function rollbackPendingLocalChangesByPK (){
product = new Product();
product.productId = 123;
Product.rollbackPendingLocalChangesByPK (successCallback,
errorFailCallback);
}

function successCallback(){
       alert("Product roll backed Successfully");
}

function errorFailCallback(res){
       alert("Rollback Pending Local Changes By PK Failed "+ "
with Error Code:" + res.errorCode + ", error message:"+
res.errorMessage + ", error information:" + JSON.stringify
(res.errorInfo));
}
```

4.14 <object>.find(Static)

This API retrieves sync object(s) using where clause from the local Database.

4.14.1 Signature

*<object>.find(**wrcondition, succCallback, errorCallback**)*

4.14.2 Parameters

- wrconditon**[String] - Mandatory**

  where clause to find the object in the database

- successCallback**[function] - Optional**

  Specifies the function that gets invoked on success

- errorCallback**[function] - Optional**

  Specifies the function that gets invoked on error

4.14.3 Platform Availability

Available on all platforms mentioned.

4.14.4 Example

```
function loadEmployeeTerritories (){
       var wcs = "where EmployeeID > 30";
       EmployeeTerritories.find(wcs, loadEmpTerritoriesSuccCallback,
loadEmpTerritoriesErrCallback);
}

function loadEmpTerritoriesSuccCallback(res){
       if((null != res && res.length> 0)){
               for ( i in res ){
                //process the row
               }
       }

}

function loadEmpTerritoriesErrCallback(res)
{
alert("failed to retrieve records" + " with Error Code:" +
res.errorCode + ", error message:"+ res.errorMessage + ", error
```

```
information:" + JSON.stringify(res.errorInfo));
}
```

4.15 <object>.getXXXwithYYY(Instance)

This API helps to retrieve all the records from the target object (XXX) corresponding to the foreign key attribute (YYY) primary key. In this case, <object> should have some relationship (One-To-Many, Many-To-One) with XXX using target attribute YYY. In case of Many-To-One, YYY is a source attribute.

4.15.1 Signature

*<object>getXXXwithYYY(**pk, succCallback, errorCallback)***

4.15.2 Parameters

- pk**[Number/jsobject/table] - Mandatory**

  Specify the primary key of the object using which the respective row data needs to be fetched from the database.

- successCallback**[function] - Optional**

  Specifies the function that gets invoked on success

- errorCallback**[function] - Optional**

  Specifies the function that gets invoked on error

4.15.3 Platform Availability

Available on all platforms mentioned.

4.15.4 Example

```
//Static Way
function getOrderDetailsWithOrderID (){
Order.getOrderDetailsWithOrderID (123, successCallback,
errorFailCallback);
}
//OOP Way
function getOrderDetailsWithOrderID (){
order = new Order ();
order.orderId = 123;
order.getOrderDetailsWithOrderID(successCallback,
errorFailCallback);
}
function successCallback(res){
       //process the result
}

function errorFailCallback(res){
       alert("failed" + " with Error Code:" + res.errorCode + ",
error message:"+ res.errorMessage + ", error information:" +
JSON.stringify(res.errorInfo));
}
```

4.16 <object>.getCountOfXXXwithYYY(Instance)

This API helps to retrieve number of records from the target object (XXX) corresponding to the foreign key attribute (YYY) primary key. In this case, <object> should have some relationship (One-To-Many, Many-To-One) with XXX using target attribute YYY. In case of Many-To-One, YYY is a source attribute.

4.16.1 Signature

*<object>getXXXwithYYY(**pk, succCallback, errorCallback)***

4.16.2 Parameters

- pk**[Number/jsobject/table] - Mandatory**

  Specify the primary key of the object using which the respective row data needs to be fetched from the database.

- successCallback**[function] - Optional**

  Specifies the function that gets invoked on success

- errorCallback**[function] - Optional**

  Specifies the function that gets invoked on error

4.16.3 Platform Availability

Available on all platforms mentioned.

4.16.4 Example

```
//Static Way
function getOrderDetailsWithOrderID (){
Order.getCountOfOrderDetailsWithOrderID (123, successCallback,
errorFailCallback);
}
Object Oriented Way
function getOrderDetailsWithOrderID (){
order = new Order ();
order.orderId = 123;
order.getCountOfOrderDetailsWithOrderID(successCallback,
```

```
errorFailCallback);
}
function successCallback(res){
       alert("Count=" + res.count);
}

function errorFailCallback(res){
       alert("failed"+ " with Error Code:" + res.errorCode + ",
error message:"+ res.errorMessage + ", error information:" +
JSON.stringify(res.errorInfo));
}
```

4.17 <object>.getDeferredUpload(Static)

This API helps to retrieve the list of rows that are deferred for upload at object level.

4.17.1 Signature

*sync.getDeferredUpload(**successcallback, errorcallback**)*

4.17.2 Parameters

- successCallback**[function] - Optional**

  Specifies the function that gets invoked on success

- errorCallback**[function] - Optional**

  Specifies the function that gets invoked on error

4.17.3 Platform Availability

Available on all platforms mentioned.

4.17.4 Example

```
function GetDeferredUpload(){
	Product.getDeferredUpload(successCallback ,
errorFailCallback)
}

function successCallback(res){
	//process the result
}

function errorFailCallback (res){
	alert("Get Deferred Upload Failed" + " with Error Code:" +
res.errorCode + ", error message:"+ res.errorMessage + ", error
information:" + JSON.stringify(res.errorInfo));
}
```

4.18 <object>.removeDeviceInstanceByPK(Instance)

This API deletes sync objects using primary key from the local Database. This does not have any effect in enterprise data source in subsequent sync cycles.

4.18.1 Signature

*<object>.removeDeviceInstanceByPK(**pk, successcallback, errorcallback**)*

4.18.2 Parameters

- pk**[Number/jsobject/table] - Mandatory**

  Specify the primary key of the object using which the respective row data needs to be fetched from the datasource.

- successCallback**[function] - Optional**

  Specifies the function that gets invoked on success

- errorCallback**[function] - Optional**

  Specifies the function that gets invoked on error

4.18.3 Platform Availability

Available on all platforms mentioned.

4.18.4 Example

```
//Static Way
function removeDeviceInstanceByPK (){
Order. removeDeviceInstanceByPK (123, successCallback,
errorFailCallback);
}
Object Oriented Way
function removeDeviceInstanceByPK (){
order = new Order ();
order.orderId = 123;
order. removeDeviceInstanceByPK (successCallback,
errorFailCallback);
}
function errorFailCallback(res){
       alert("Removing Order Failed" + " with Error Code:" + res.errorCode
+ ", error message:"+ res.errorMessage + ", error information:" +
JSON.stringify(res.errorInfo));
}
```

> ***Note:*** Cascade delete is implemented for **removeDeviceInstanceByPK** ORM API in 5.5.4 GA. When you use **removeDeviceInstanceByPK** ORM API to delete records in the parent table (client side) along with cascade delete flag true (as highlighted below in Red), the records in the child table are also deleted.
>
> <OneToMany TargetObjectAttribute="CategoryID" TargetObject="Products" SourceObjectAttribute="CategoryID" Cascade="true"/>

4.19 <object>.getAllCount(Static)

This API helps to retrieve total number of sync objects present in local database.

4.19.1 Signature

*<object>. getAllCount (**succCallback, errorCallback**)*

4.19.2 Parameters

- successCallback**[function] - Optional**

  Specifies the function that gets invoked on success

- errorCallback**[function] - Optional**

  Specifies the function that gets invoked on error

4.19.3 Platform Availability

Available on all platforms mentioned.

4.19.4 Example

```
//Static Way
function getAllCount(){
```

```
Order.getAllCount(successCallback, errorFailCallback);
}
function successCallback(res){
       alert("Count=" + res.count);
}

function errorFailCallback(res){
       alert("failed"+ " with Error Code:" + res.errorCode + ",
error message:"+ res.errorMessage + ", error information:" +
JSON.stringify(res.errorInfo));
}
```

4.20 <object>.getCount(Static)

This API helps to retrieves total number of sync objects present in local database matching the where clause.

4.20.1 Signature

*<object>. getAllCount (**succCallback, errorCallback**)*

4.20.2 Parameters

- whereClause[String] - Mandatory

  Specify the where clause.

- successCallback**[function] - Optional**

  Specifies the function that gets invoked on success

- errorCallback**[function] - Optional**

  Specifies the function that gets invoked on error

4.20.3 Platform Availability

Available on all platforms mentioned.

4.20.4 Example

```
//Static Way
function getCount(){
Order.getCount("where OrderId > 230", successCallback,
errorFailCallback);
}
function successCallback(res){
       alert("Count=" + res.count);
}

function errorFailCallback(res){
       alert("failed"+ " with Error Code:" + res.errorCode + ", error
message:"+ res.errorMessage + ", error information:" +
JSON.stringify(res.errorInfo));
}
```

4.21 <object>.createAll

This API helps developer to insert bulk records into the device database in a transaction.

4.21.1 Signature

*<object>.createAll (**array**,**succCallback**, **errorCallback**)*

4.21.2 Parameters

- array - **Mandatory**

  Specify the array of records to be created

- successCallback**[function] - Optional**

  Specifies the function that gets invoked on success

- errorCallback**[function] - Optional**

  Specifies the function that gets invoked on error

4.21.3 Platform Availability

Available on all platforms mentioned.

4.21.4 Example

```
function bulkInsert(){
kony.sync.enableORMValidations = true; // This is Global variable,
works only on this createAll ORM, set this to 'false' if data
validation is not required while doing bulk insert
                var PersonData = [

{FirstName:Paul, LastName:Smith, Age:25},

{FirstName:Brenda, LastName:Flintstone, Age:27}
                ];
                Person. createAll
(PersonData,successcallback,errorcallback)
                // Person. createAll
(PersonData,successcallback,errorcallback,false) //<--Markforupload
'false'
```

```
}

// 5.5 and 5.0 Versions
function successCallback(res){
alert("Created all the records successfully with primary keys" +
JSON.stringify(res));
}

// 5.5
function errorFailCallback(res){
alert("Failed to insert records, Error Code : " + res.errorCode + "
Error Message : " + res.errorMessage);
}

// 5.0
function errorFailCallback(){
       alert("Failed to insert records");
}
```

4.22 <object>.updateAll

This API helps developer to update multiple records into the device database in a transaction.

> ***Note:*** The API is available in Sync 5.0.x above versions only.

4.22.1 Signature

*<object>.updateAll (**array**,**succCallback**, **errorCallback**)*

4.22.2 Parameters

- array - **Mandatory**

  Specify the array of records to be created

- successCallback**[function] - Optional**

  Specifies the function that gets invoked on success

- errorCallback**[function] - Optional**

  Specifies the function that gets invoked on error

4.22.3 Platform Availability

Available on all platforms mentioned.

4.22.4 Example

```
Introduced new ORM API upadateAll in 5.5.4 GA. This ORM helps to
update more than one record at a time.
Example :
function updateAllCategory()
{
 var inputArray = [];
 inputArray[0] = {};
 inputArray[0].changeSet = {
  CategoryName : "UpdateALL_0",
  Description : "CatUpdateALLTest_0"
 }; //contains values to be updated
 inputArray[0].whereClause = "where CategoryID < 1"; //where clause
on which to update

 inputArray[1] = {};
```

```
 inputArray[1].changeSet = {
  CategoryName : "UpdateALL_1",
  Description : "CatUpdateALLTest_1"
 }; //contains values to be updated
 inputArray[1].whereClause = "where CategoryID > 1 AND CategoryID <
4"; //where clause on which to update

 inputArray[2] = {};
 inputArray[2].changeSet = {
  CategoryName : "UpdateALL_2",
  Description : "CatUpdateALLTest_2"
 }; //contains values to be updated
 inputArray[2].whereClause = "where CategoryID IN (5,4,6)"; //where
clause on which to update

 inputArray[3] = {};
 inputArray[3].changeSet = {
  CategoryName : "UpdateALL_3",
  Description : "CatUpdateALLTest_3"
 }; //contains values to be updated
 inputArray[3].whereClause = "where CategoryID > 6 AND CategoryID
!=8"; //where clause on which to update

  Categories.updateAll(inputArray,successCallback,errorCallback);

 function successCallback(res){
  alert("success in updateAll");
  kony.print("success in updateAll"+JSON.stringify(res));
 }

 function errorCallback(res){
  alert("error in updateAll");
  kony.print("error in updateAll"+JSON.stringify(res));
```

```
  }
}
```

4.23 <object>.removeDeviceInstance

This API helps developer to delete the sync objects that use where clause from the local Database and has no effect in enterprise data source in subsequent sync cycles.

4.23.1 Signature

*<object>removeDeviceInstance (**whereclause**, **successcallback**, **errorcallback**)*

4.23.2 Parameters

- whereclause - **Mandatory**

  Specify the primary key of the object using which the respective row data needs to be fetched from the datasource

- successCallback**[function] - Optional**

  Specifies the function that gets invoked on success

- errorCallback**[function] - Optional**

  Specifies the function that gets invoked on error

4.23.3 Platform Availability

Available on all platforms mentioned.

4.23.4 Example

```
//Static Way

function removeDeviceInstance (){
Order.removeDeviceInstance (“where OrderID >= 100”, successCallback,
errorFailCallback);
}
function successCallback(){
alert("Orders Deleted Successfully");
}
function errorFailCallback(res){
alert("Removing Order Failed" + " with Error Code:" +
res.errorCode + ", error message:"+
res.errorMessage + ", error information:" + JSON.stringify
(res.errorInfo));
}
```

5. Scope Level

This set of APIs helps the developer to perform various ORM operations at Sync Scope level.

5.1 <scope>.reset

This method deletes all data in all tables for the particular scope on which it is called.

5.1.1 Signature

*<scope>.reset (**resetSuccessCallback** ,**resetFailCallback**)*

5.1.2 Parameters

- resetSuccessCallback[function] - Optional

  Specifies the function that gets invoked on success

- resetFailCallback[function] - Optional

  Specifies the function that gets invoked on failure

5.1.3 Platform Availability

Available on all platforms mentioned.

5.1.4 Example

```
function scopeReset(){
 myScope.reset(resetSuccessCallback , resetFailCallback);
 }

 function resetSuccessCallback(){
 alert("Scope reset successfully");
```

```
 }
 function resetFailCallback(res){
 alert("Scope reset failed" + " with Error Code:" + res.errorCode +
", error message:"+ res.errorMessage + ", error information:" +
JSON.stringify(res.errorInfo));
 }
```

6. Sync Level

This set of APIs helps the developer to get all pending upload instances. For example, if a record is updated three times with *enableIntermediateUpdates = true*, then you can retrieve all three instances of that record.

This API is at application level and provides comprehensive list of all the instances from all the tables in all the scopes.

You have an option to retrieve only count of the instances instead of actual instances. You are recommended to use this when only count is required as this saves huge data from getting loaded into memory.

6.1 sync.getAllPendingUploadInstances

6.1.1 Signature

sync.getAllPendingUploadInstances (**retrieveOnlyCount**, **successcallback**, **errorcallback**);

6.1.2 Parameters

- retrieveOnlyCount =true/false
- successcallback[function] - Optional

  Specifies the function that gets invoked on success
- errorcallback[function] - Optional
  Specifies the function that gets invoked on failure

6.1.3 Platform Availability

Available on all platforms mentioned.

6.1.4 Example 1

```
/*
Assuming there are two scopes scope1 and scope2.
scope1 has 2 tables; tab1(colA, colB) with 4 changes and tab2(col1,
col2) with 2 changes
scope2has 2 tables; tab1(col1, col2) with2 changes and tab2(col1,
col2) with 2 changes
*/
sync. getAllPendingUploadInstances
(false,mySuccesscallback,myErrorcallback);
function mySuccesscallback(res){
kony.print("Total instances pending for upload:"+res.totalCount);
kony.print("Total pending upload instances for
scope1":+res.scope1.count);
kony.print("Total pending upload instances for tab1":+
res.scope1.tab1.count);
kony.print("Pending upload instance data for tab1":+ JSON.stringify
(res.scope1.tab1.data));
kony.print("Total pending upload instances for tab2":+
res.scope1.tab2.count);
kony.print("Pending upload instance data for tab2":+ JSON.stringify
(res.scope1.tab2.data));
//similarly data can be accessed for scope2
}
function myErrorcallback (res){
alert("Error occurred while fetching pending upload instances");
}
```

6.1.5 Example 2

```
sync.getAllPendingUploadInstances (true,
mySuccesscallback,myErrorcallback);
function mySuccesscallback(res){
kony.print("Total instances pending for upload :" + res.totalCount);
kony.print("Total pending upload instances for scope1": +
res.scope1.count);
kony.print("Total pending upload instances for
tab1":+res.scope1.tab1.count);
kony.print("Total pending upload instances for
tab2":+res.scope1.tab2.count);
//similarly data can be accessed for scope2
//Note that in this case res.scope1.tab1.data and
res.scope1.tab2.data will be null
}
function myErrorcallback (res){
alert("Error occurred while fetching pending upload instances");
}
```

7. SQLite DB Encryption

You can enable encryption using the below two APIs:

- sync.init
- sync.reset

7.1 sync.init

You can access *sync.init* in two ways:

- **Previous way**:

  sync.init(**syncInitSuccesscallback**, **syncInitErrorcallback**);

  > ***Note:*** Encryption is disabled, by default.

- **New way**:

```
var config={};
config.oninitsuccess = syncInitSuccesscallback;
config.oniniterror = syncInitErrorcallback;
config.devicedbencryptionkey = "KonySync";
sync.init(config);
```

> ***Note:*** The database is encrypted using devicedbencryptionkey.
> If config.devicedbencryptionkey=null/undefined/<Empty String>, then encryption is disabled.

7.2 sync.reset

You can access *sync.reset* in two ways:

- **Previous way**:

  sync.reset(**syncResetSuccesscallback**, **syncResetErrorcallback**);

  > ***Note:*** Encryption is disabled, by default.

- **New way**:

```
var config={};
config.onresetsuccess = syncResetSuccesscallback;
config.onreseterror = syncResetErrorcallback;
config.devicedbencryptionkey = "KonySync";
sync.reset(config);
```

  > ***Note:*** The database is encrypted using devicedbencryptionkey.
  > If config.devicedbencryptionkey=null/undefined/<Empty String>, then encryption is disabled.

> ***Note:*** To encrypt SQLite database for iPad/iPhone devices, follow these steps:

1. Extract the `KAR` file with "-sqlcipher" option3

   For example:
   When KAR is not in local machine: perl extract.pl http://ip.add.re.ss:8888/appidr/?type=ipad -sqlcipher XYZ.
   When KAR is in local machine: perl extract.pl <KAR file location>/AppName.KAR -sqlcipher XYZ.

2. Encryption does not take place unless you extract the `KAR` file using "-sqlcipher" option.

> ***Note:*** For Android Devices:

1. Supported Android plug-ins:
    - Tab Plug-in version : ANDT-GA-5.0.44
    - Mobile Plug-in version : AND-GA-5.0.4
2. SQLite encryption dependent libraries are available from KonySync IDE 5.5.9.2 GA. These dependent library files are copied into application workspace when user generates application forms or offline services using KonySync IDE 5.5.9.2 GA.

7.2.1 Manual Instructions to Copy SQLite Encryption Dependent Libraries for Android

For Android mobile devices:

1. Download SQLCipher binaries from https://s3.amazonaws.com/sqlcipher/SQLCipher+for+Android+v3.0.1.zip.
2. Access the `Application Resource` folder from IDE and create the folder structure `resources/customlibs/lib/android`, if already does not exist.
3. Copy the contents of the `libs` folder of SQLCipher into `resources/customlibs/lib/android`, if does not exist already. The SQLCipher `libs` folder contains the two folders (`armeabi`, `x86`) and three jar files. You need to copy all the files to `android`.
4. Copy `konysqlcipher.jar` into the same android folder. You can obtain the latest jar file from the Android build folder [Ctrl + Alt + T] (jar location: `workspace\temp\<Application>\build\luaandroid\extres`).
5. Create the folder structure `resources\mobile\native\android\assets` (for Android mobile device, if does not exist already) and copy the contents of `assets` folder of SQLCipher into `assets` folder location.
6. The SQLCipher `assets` folder contains the file `icudt46l.zip` and you need to copy it to

`resources\mobile\native\android\assets` folder.

7. Build the application (All build modes supported).

For Android TAB devices:

1. Download SQLCipher binaries from https://s3.amazonaws.com/sqlcipher/SQLCipher+for+Android+v3.0.1.zip.
2. Open the `Application Resource` folder from Kony Visualizer IDE and create the folder structure `resources/customlibs/lib/tabrcandroid`, if does not exist already.
3. Copy the contents of the `libs` folder of SQLCipher into the `resources/customlibs/lib/tabrcandroid`, if does not exist already. The SQLCipher `libs` folder contains the two folders (`armeabi`, `x86`) and three jar files, copy all these files to the `tabrcandroid` folder.
4. Copy the attached jar file, `konysqlcipher.jar` also into the `tabrcandroid` folder. You can obtain the latest jar file from the `Android build` folder [Ctrl + Alt + T] (the jar file is located in `workspace\temp\<Application>\build\luatabrcandroid\extres`).
5. Create the folder structure `resources\tablet\native\andriodtab\assets`, if does not exist already and copy the contents of `assets` folder of SQLCipher into above location. SQLCipher `assets` folder contains the `icudt46l.zip` file and you need to copy it to `resources\mobile\native\andriodtab\assets` folder.
6. Build the application (All build modes supported)

7.3 Known Issues

1. Encryption key cannot be changed once enabled.
2. Currently no error callback displays, if encryption is not successful.

3. Applications using SQLite encryption on Android devices may not work properly when you try to encrypt existing non-encrypted SQLite database.
4. Applications using SQLite encryption may not respond properly on Android devices causing data loss when you try to access encrypted SQLite DB without a valid KEY. The behavior depends on the Android version. If the Android version is:
    a. Below 3.0, the corrupted database file is removed and new database is created with the same name.
    b. Below 3.0 and 4.1, since you make sure the database file is not deleted at any cost, the application encounters the StackOverflow exception.
    c. From 4.1 and above, an empty database handle is returned on `openDatabase()` call.

8. Chunking Mechanism

Chunking mechanism enables to support data downloads from Enterprise Datasource to device without interruption on low bandwidth, on / off and toggling between the networks, especially when data is huge.

You can configure the below parameters depending on the environment and requirements along with already existing Kony Fabric Sync configuration parameters in *sync.startSession* API.

1. **chunksize** (**Type**: **Number**): To configure the size of chunk while downloading data. If batchsize configured is more than the chunksize, Sync changes to chunking mode else, the data downloads normally without chunking.
    a. For example, batchsize = 500 (Records) = 2048 KB, chunksize = 512 (KB), Sync divides the batch into four chunks and is downloaded independently to client, and finally consolidates and stores into device database.
    b. For example, batchsize = 500 (Records) = 256 KB, chunksize = 512 (KB), Sync happens normally without chunking and you can download the whole batch to client, in one call and store into device database

    Example to set chunksize: *config.chunksize = 512*. This sets the chunksize to 512 KB.

    > ***Note:*** This parameter is a mandatory attribute for chunking.

2. **onchunkstart** (**Type**: **callback**)

    This is a parameter that has reference to the callback that is called when each chunking call starts. This parameter is optional.

    Input parameters: A map is passed to the function that has the following keys:

    a. chunkcount
    b. payloadid

c. scopename

d. chunkid

e. chunkrequest

For example,

```
config.onchunkstart = chunkstartfn;
function chunkstartfn(res){
kony.print("Chunk started for " + res.chunkid + " where total
number of chunks are : "
+ res.chunkcount);
}
```

> ***Note:*** This parameter is optional.

3. **onchunksuccess** (**Type**: **callback**)

   This is a parameter that has reference to the callback that is called when a chunking call completes successfully.

   Input parameters: A map is passed to the function that has the following keys:

   a. chunkcount

   b. payloadid

   c. scopename

   d. chunkid

   e. pendingchunks

   f. chunksdownloaded

For example:

```
config.onchunksuccess = chunksuccessfn;

function chunksuccessfn(res){
kony.print("Chunk with " + res.chunkid + " is successfully
downloaded where pending chunks are " + res.pendingchunks);
}
```

> ***Note:*** This parameter is optional.

4. **onchunkerror** (**Type**: **callback**)

   This is a parameter that has reference to the callback that is called when a chunking call is not successful.

   Input parameters: A map is passed to the function that has the following keys:

   a. chunkcount
   b. payloadid
   c. scopename
   d. chunkid
   e. pendingchunks
   f. chunksdownloaded
   g. errorCode
   h. errorMessage

   For example:

```
config.onchunkerror = chunkerrorfn;
function chunkerrorfn(res){
kony.print("Error occurred while downloading Chunk with" +
res.chunkid + "Error Code :" + res.errorCode + " and Error
Message : " + res.errorMessage);
}
```

> ***Note:*** This parameter is optional.

5. **onretry** (**Type**: **callback**)

   This is a parameter that has reference to the callback that is called for each retry and is optional.

   Input parameters: A map is passed to the function that has the following keys:

   a. request

   b. errorResponse

   c. retryCount

   For example:

```
config.onretry = function(res){
kony.print("Error occurred while making " + JSON.stringify
(res.request) + ":" +
JSON.stringify(res.errorResponse) );
kony.print("retrying for " + res.retryCount + " times");
}
```

> ***Note:*** This parameter is optional.

6. **retryerrorcodes** (**Type**: **vector/array**)
   Specify for which errorcodes, the application should retry.

For example:

config.retryerrorcodes = [**1000**, **1013**, **1015**]

> ***Note:*** This is an optional parameter, if you do not pass, the error codes, 1000, 1013, 1014 and 1015 are used for retry.

7. **numberofretryattempts** (**Type**: **Number)**

   Configure this parameter to specify the number of attempts that application should retry to make a call successful in case of some specific error codes related to network issues that are specified by retryerrorcodes parameter mentioned above.

   For example:

   config.numberofretryattempts = 10

   > ***Note:*** This parameter is a mandatory attribute for retry.

8. **retrywaittime** (**Type** : **Number)**

   Specify the wait time (in seconds) to retry after a network error has occurred. This is optional,

   default is 1 second.

   For example:

   config.retrywaittime = 5

   > ***Note:*** This is an optional parameter.

9. **networktimeout (Type: Number)**

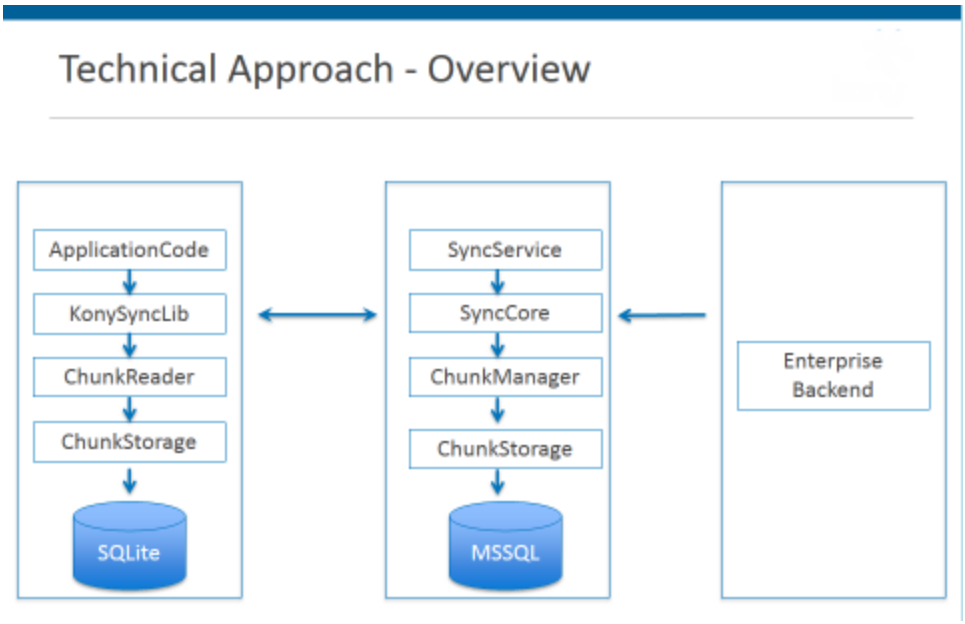
   Configure this parameter to set the network time out while application is waiting for response from the server. This is in seconds. This is an optional parameter, and the default value depends upon the underlying channel

   For example:

   config.networktimeout = 1800; (that is 30 minutes)

   > ***Note:*** This is an optional parameter.

8.1 Technical Architecture Overview



8.1.0.1 Technical Approach Details

Client Side:

- **ChunkReader** first assembles the data before passing the same to the Data layer
- Chunks are also stored in SQLite to take care of network interruptions when reading the chunks

- Incase of network interruptions, the client resumes from the point it failed when downloading chunks.

Server Side:

The **ChunkManager**

- Splits the JSON into chunks less than the size of the **transfer buffer size** (chunksize) specified by the developer
- Stores the chunks offline to take of **network interruptions** as the chunks are downloaded by the device
- Deletes also any chunks that are processed by the Client.

8.2 Working with High Chunk Sizes

While working with chunk sizes, make sure the specified chunk size is less than the network packet size of Syncconsole DB. Else, increase the size of the network packet size accordingly as below,

8.2.1 For MySQL Server

In MySQL, `max_allowed_packet` is the attribute for specifying the network packet size. By default, the value is 4 MB and the maximum allowed for this parameter is 100 GB. For increasing max allowed packet size, edit `<MySQL installation directory>/my.ini` file and increase the `max_allowed_packet` value greater than your chunk size and restart the MySQL server.

8.2.2 For MS SQL Server

In MS SQL Server, set the network packet size by increasing its value. By default, the value is 4096 bytes. The maximum value allowed for this parameter is 32767 bytes.

We can set this by using SQL Server Management Studio as below:

- In **Object Explorer** (left side pane), right-click on the required server and select **Properties**.
- Click the **Advanced** node (on left side pane).

- On the right side pane under **Network**, select the value of “Network Packet Size” box to 32767.
- Click **OK** to confirm.

8.2.3 For Oracle

Set the following in `listener.ora` file

```
LISTENER= (DESCRIPTION= (ADDRESS= (PROTOCOL=tcp) (HOST=konydb_
server) (PORT=1521)
(SEND_BUF_SIZE=32767)(RECV_BUF_SIZE=32767)))
```

Set the following in `sqlnet.ora` file

```
RECV_BUF_SIZE=32767
SEND_BUF_SIZE=32767
```

9. Handling Errors

Kony Fabric Sync Framework handles error handling through error codes. All the ORMs take errorCallback as a parameter that is called if any error occurs in that particular ORM. All ORMs return error parameter in errorCallback that is mapped and contains the following keys:

1. **errorCode[number]**: Each error is assigned a specific error code.
2. **errorMessage[String]**: Description of the error scenario.
3. **errorInfo[JSObject]**:Other information about error. It may also contain intermediate errors.

9.1 Kony Fabric Sync Client Error Codes

Following table describes various client error codes and corresponding error messages:

Error Code	Error Scenario	Error Message
7001	Invalid data type passed for some ORM	Invalid data type for the attribute *<attribute global name>* in *<Sync Object globalname>*. Expected: *<expected Type>* Actual: *<actual Type>*
7002	Mandatory Attribute Missing in the Sync Object	Mandatory Attribute *<attribute global name>* missing in the Sync Object *<Sync Object globalname>*
7003	Primary Key not specified in updating an item in Sync Object	Primary Key *<primaryKey>* not specified in updating an item in *<Sync Object globalname>*.
7004	Scopes loading failed	Scopes loading failed.
7005	Sync Reset failed	Sync Reset failed.
7006	Register device failed	Register device failed.

7007	Session breaks because of some scope failure	Session breaks since user scope failure.
7008	Session in progress	Session in progress.
7009	No data with specified primary key found in local database while doing performing some operation on a Sync Object	No data with specified primary key found in *<Sync Object globalname>*.
7010	Transaction failed to open	Transaction failed
7011	Error occurred while establishing a Database connection.	Error occurred while establishing a Database connection.
7012	If some record is created with markforupload=false, you should update it with markforupload=false only, else updates go to server for some non-existent server record and merge fails	Record does not exist on server, mark it for upload before updating/deleting it.
7013	Error during Deferred Upload Transaction	Error during Deferred Upload Transaction
7014	Referential Integrity Constraints Violation	Referential Integrity Constraints Violation: *<Target Attribute> = < Target Value>* does not exists in *<Target Object>*.
7015	Length exceeds specified limit while creating/updating a record	Length exceeds the limit for the attribute *<attribute global name>* in *<Sync Object globalname>*. Expected: *<Expected Length>* Actual: *<Actual Length>*

7016	If user tries to create a record with already existing primary key.	Primary Key *<attribute global name>* already exists in table *<Sync Object globalname>*. Please give different value of primary key.
7018	Some malicious values like positive/negative infinities, NaN is passed in some create/update operation	Malicious value *<value>* given for attribute *<attribute global name>*.
7019	Some error occurs while executing SQL statement	Some error occurred in executing SQL statement
7020	Error occurred while Uploading changes to Server	Error occurred in Uploading changes to Server
7021	Error occurs while Downloading changes from Sever	Error occurred in Downloading changes from Sever
7022	Error occurs while syncing one or more scopes	Error occurred while syncing one or more scopes
7028	Error occurs if `markforUpload` value is set to `true`.	`markforUpload` value can not be made true if the object is created with mark for upload as `false`. If the user tries to change an object with `markForUpload` value from `false` to `true` using `update` or `updateByPK` APIs, the following error is occurred.
7777	Some random error	The following error occurred while performing *<operation Name> : <error>*. Possible reasons can be sync.init may not have been invoked.[This comes for random error]
8888	Unknown error from server	Unknown error from server

1000		Unknown Error while connecting (If the platform cannot differentiate between the various kinds of network errors, the platform reports this error code by default)
1011		Device has no WIFI or mobile connectivity. Please try the operation after establishing connectivity.
1012		Request Failed
1013		Middleware returned invalid JSON string
1014		Request Timed out
1015		Cannot find host
1016		Cannot connect to host
1022		Service call is canceled. You can customize the behavior of the application if a service call is canceled.
1200		SSL - Certificate related error codes.

Note: **7022** is a generic error from client side which will be shown for different scenarios as described in Sync Server Error Codes section.

9.2 Kony Fabric Sync Server Error Codes

Following table describes various server error codes and corresponding error messages:

Error Code	Error Scenario	Error Message
1001	User credentials are incorrect	Invalid userID or password.

Error Code	Error Scenario	Error Message
1002	Not an authorized user for the application. A user is not assigned to the application.	Application not authorized to the user.
1003	A user's deviceId is invalid or does not exist in a registered device list at the Kony Fabric Sync server.	Invalid deviceID.
1004	`SyncConfig xml` corresponding to applicationID not found at the Kony Fabric Sync Server.	Unknown applicationID.
1005	A userID does not exist in a registered device list at the Kony Fabric Sync Server.	Unknown userID.
1006	A user is not registered to the Kony Fabric Sync Server with this deviceID	A device is not assigned to a user.
1008	Empty deviceID in the request.	Empty deviceID.
1010	Other authentication errors not already listed	Unknown error occurred, error message : *<errMsg>*.
SY3001E	The current client app is built on an old sync config.	An application with version *<app version>* for application ID *<applicationId>* is not the latest one. Please upgrade the application.
SY3002E	A sync config on which the client is built is not uploaded to the Kony Fabric Sync Server or overridden with the new version.	An application with version *<app version>* for application ID *<applicationId>* was not found in the application history.

Error Code	Error Scenario	Error Message
SY3003E	A client app version sent from the device is null or empty.	A client application version for applicationID *<applicationId>* is null or empty.
SY3004E	A server app version in sync config that is uploaded is null or empty.	A server application version for applicationID *<applicationId>* is null or Empty.
SY3005E	The sync config that is uploaded to Sync Server differs from the sync config on which app is built on.	The client application version *<Client App version>* for applicationID *<applicationId>* is not matching to server application version *<Server App version>*.
SY0000E	There are two scenarios, 1. After a successful sync, you delete the device using the sync console and repeat the sync operation. 2. Re-install the Sync Console DB, and upload `SyncConfig.xml` call `sync.startsession()` without `reset()`	DeviceID *<deviceID>* does not exist.
	Different strategies selected for client and server (app is built on the sync config that has a different strategy from the uploaded sync config).	Strategy mismatch server = *<strategy-defined in syncconfig>,* client = *<client strategy>*.
	During the sync download, chunk packets are corrupted due to network interruptions.	CheckSum not matched with the server calculated checksum for the payload *<payloadId>*

Error Code	Error Scenario	Error Message
	When the Kony Fabric Sync Server processes the request, the HTTP session times out.	Request failed due to session timeout
	It is a safe measure to protect the illegal access	Found invalid Application ID for this session
	It is a safe measure for the format issues in the dynamic schema upgrade.	Invalid selected sync object format
	It is a safe measure for the client context format coming from device.	Invalid client context
	Batch size passed must be greater than zero.	Invalid batch size : *<batch size>* must be greater than zero.
	Replica database version and the sync config uploaded to the Kony Fabric Sync Server are not the same. (Make sure that the persistent database is created with the latest sync config).	Replica version mismatch with server-side.
	Sync strategy is from client is not OTA and persistent.	Unknown sync strategy
	UserID and password combination is incorrect.	Authentication failed with given userID and password.
	It is a safe measure to protect the illegal access	Found invalid ApplicationID for this session
	It is a safe measure to protect the illegal access	Found invalid UserID for this session

Error Code	Error Scenario	Error Message
	It is a safe measure to protect the illegal access	Found invalid DeviceID for this session
	`sync.home` is not set properly or `syncservice.properties` not found in the `sync.home` location	Cannot find the properties file `syncservice.properties`. Please set a valid `sync.home` to continue.
	`sync.home` is not set properly. `syncservice.properties` not found in the `sync.home` location, or the file may be corrupted.	Failed to create instance for `ServicesConfigProperties` in context initialization.
	`sync.home` is not set to properly. `syncservice.properties` not found in the `sync.home` location, or the file may be corrupted.	Error occurs while reading `syncservice.properties` file.

9.3 Kony Fabric Sync Error Callbacks

Kony Fabric Sync Error Callbacks (both Upload and Download) provide exact messages of the error that occur at Kony Fabric Sync Server during the synchronization. You can use these error messages to display on the device application for the end user for further debugging. For the developer, the whole java stack trace is available in the response for debugging.

9.3.1 Example

Upload failed: Error occurred while upload on SyncObject 'Contact' : Unable to create/update fields: Name. Check the security settings of this field and verify that it is read/write for your profile or permission set.

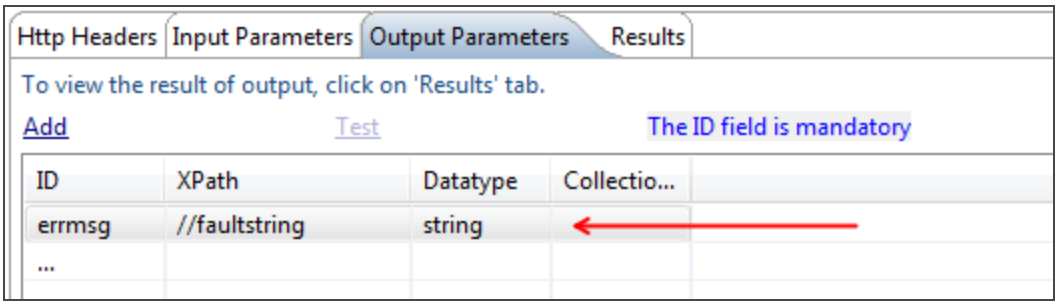
9.4 Handling SOAP Fault Error

Kony Fabric Sync Server will by default raise an error if it encounters a SOAP Fault element in the Web Service response. A typical SOAP Fault element looks like,

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
<SOAP-ENV:Fault>
<faultcode xsi:type="xsd:string">SOAP-ENV:Client</faultcode>
<faultstring xsi:type="xsd:string">
Failed to locate method ….
</faultstring>
</SOAP-ENV:Fault>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

If the Web Service returns error using standard SOA Body element, then the developer can still raise errors providing an XPATH (or relevant extraction expression) for the attribute 'errmsg'.

9.4.1 Example:



9.5 Handling Timeout Error

When backend services are taking more time to respond during download batch files, we can pass the following optional context variables from client side to handle the timeouts between the device and Kony Fabric Sync server:

1. **BATCH_TIMEOUT**: (The value in passed in milliseconds) indicates to Kony Fabric Sync server to forcefully respond back to device with the response data retrieved from backend datasource within this time. For example, if the http timeout between device and Kony Fabric Sync server is 60 seconds, then you may set this value as 50 seconds assuming 10 seconds is network delay between device and Kony Fabric Sync server.

   > ***Note:*** Network delay between device and Kony Fabric Sync server is not considered in this timeout value.

2. **BATCH_SERVICE_COUNT_LIMIT**: Numeric value indicating the maximum number of backend datasource calls allowed in a download batch. If this value is 'n' then Kony Fabric Sync server will make maximum of 'n' calls in a download batch.

9.5.1 Example

```
function ondownloadstartCallback(outputparams){
showSyncLoadingScreen("Download Started")
var req = outputparams.downloadRequest;
if(req.clientcontext===undefined){
req.clientcontext = {};
}
req.clientcontext.BATCH_SERVICE_COUNT_LIMIT = 3;
req.clientcontext.BATCH_TIMEOUT = 50000;
return req;
}
```

10. Logging Framework

Logging mechanism is improved from **5.5.3 GA** onwards by providing below log levels and hierarchy follows as **TRACE**(6) > **DEBUG**(5) > **INFO**(4) > **WARN**(3) > **ERROR**(2) > **FATAL**(1) > **NONE**(0).

You can use the following API to set the log level:

10.1 Signature

*sync.log.setLogLevel(**level**, **logSuccessCallback**, **logFailureCallback**)*

where level can be one of the following:

1. kony.sync.log.TRACE: The level is the lowest log level. The level prints all Sync SDK logs.
2. kony.sync.log.DEBUG: The level prints Sync SDK debug info, warnings, errors, and fatal logs.
3. kony.sync.log.INFO: The level prints Sync SDK info, warnings, errors, and fatal logs.
4. kony.sync.log.WARN: The level prints Sync SDK warnings, errors, and fatal logs.
5. kony.sync.log.ERROR: The level prints Sync SDK errors, and fatal logs.
6. kony.sync.log.FATAL: The level prints only Sync SDK fatal logs.
7. kony.sync.log.NONE: The level does not print any Sync SDK logs.

10.2 Example:

sync.log.setLogLevel(kony.sync.log.ERROR, logSuccessCallback, logFailureCallback)

You can use the following APIs to check the current log level:

1. sync.log.isDebugEnabled()
2. sync.log.isTraceEnabled()

3. sync.log.isInfoEnabled()
4. sync.log.isWarnEnabled()
5. sync.log.isFatalEnabled()
6. sync.log.isNoneEnabled()

These return a Boolean value, *true* for Enabled and *false* for Disabled.

For example, if the level is set to DEBUG, sync.log.isTraceEnabled() returns *false* and the rest of the hierarchy functions return *true*.

Note:

1. The value is set to kony.sync.log.DEBUG, by default.
2. kony.sync.log.NONE disables all logs.

If log is set at TRACE level, then all the lower levels like DEBUG, INFO are also logged.

11. Passing Context Variable Values

From 5.5.1 GA onwards, you can pass the context variables from client side in upload or download calls, and can map them in Input mapping of the all the Operation. The mapping is not required in case of databases.

Find below the callback example to configure the context variables in upload:

11.1 Example

```
function onuploadstartCallback(res){
var req = res.uploadRequest;
if (req.clientcontext === undefined) {
req.clientcontext = {};
}
req.clientcontext.value = "Sample_Client_Context_Value";
//kony.print("*********onuploadstartDemoTest :" + JSON.stringify
(res))
kony.print("*********onuploadstartDemoTest***************** :");
for (var i in res) {
kony.print("****" + i.toString() + " :" + JSON.stringify(res[i]));
}
//showSyncLoadingScreen("Upload Started:" + outputparams)
return req;
}
```

Find below the callback example to set the context variables in download:

11.2 Example

```
function ondownloadstartCallback(outputparams){
                var req = outputparams.downloadRequest;
```

```
                if(req.clientcontext===undefined){
                                req.clientcontext = {};
                }
                req.clientcontext.var1 = "123";
                req.clientcontext.var2 = "xyz";
                return req;
}
```

Note: You can pass the context variables technically from a device side. You cannot use the context variables in database datasource applications. Passing Context Variable Values is supported for all the database and non-database datasources.

12. Override Network Service Calls

To pass custom headers or write custom logic during network calls, follow these steps:

1. Add the following parameters in config while passing it to sync.startSession API. The netOverrideFunc is the function that will be used to make other network calls

   config.invokeservicefunction = netOverrideFunc;

```
function syncStartSession(){
var config = {};
config.userid  = "syncadmin";
config.password = "SyncAdmin123";
config.appid = "syncAppID";
config.batchsize = "500";
config.serverhost = "ip.add.ress";
config.serverport = "8989";
config.onsyncstart = onsyncstartCallback;
config.onscopestart = onscopestartCallback;
config.onscopeerror = onscopeerrorCallback;
config.onscopesuccess = onscopesuccessCallback;
config.onuploadstart = onuploadstartCallback;
config.onuploadsuccess = onuploadsuccessCallback;
config.ondownloadstart = ondownloadstartCallback;
config.ondownloadsuccess = ondownloadsuccessCallback;
config.onbatchstored = onbatchstoredCallback;
config.onbatchprocessingstart = onbatchprocessingstartCallback;
config.onbatchprocessingsuccess =
onbatchprocessingsuccessCallback;
config.onsyncsuccess = onsyncsuccessCallback;
config.onsyncerror = onsyncerrorCallback;
//set networktimeout if needed
config.networktimeout = 90;
```

```
//New Config parameter
config.invokeservicefunction = netOverrideFunc;
showSyncLoadingScreen("Starting Sync Session")
sync.startSession(config);
}
```

2. Implement netOverrideFunc as in the following code:

```
function netOverrideFunc(url, params, callback, context){
/write your own implementation here
//after getting response, call callback as below
function networkCallbackStatus(status, result, cntxt){
//status should be 400 when call is complete, please read kony
//implementation of kony.net.invokeServiceAsync
if(status === 400){
kony.print(JSON.stringify(result));
callback(status, result, cntxt);
}
};
kony.print("Hitting "+ url + "with params " + JSON.stringify
(params));
//here you can write your own implementation or make any
service call to //get tokens, etc. before making this service
call
kony.net.invokeServiceAsync(url, params, networkCallbackStatus,
context);
}
```