Kony Fabric

Sync Services Framework

Getting Started 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.

Revision History

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

Table of Contents

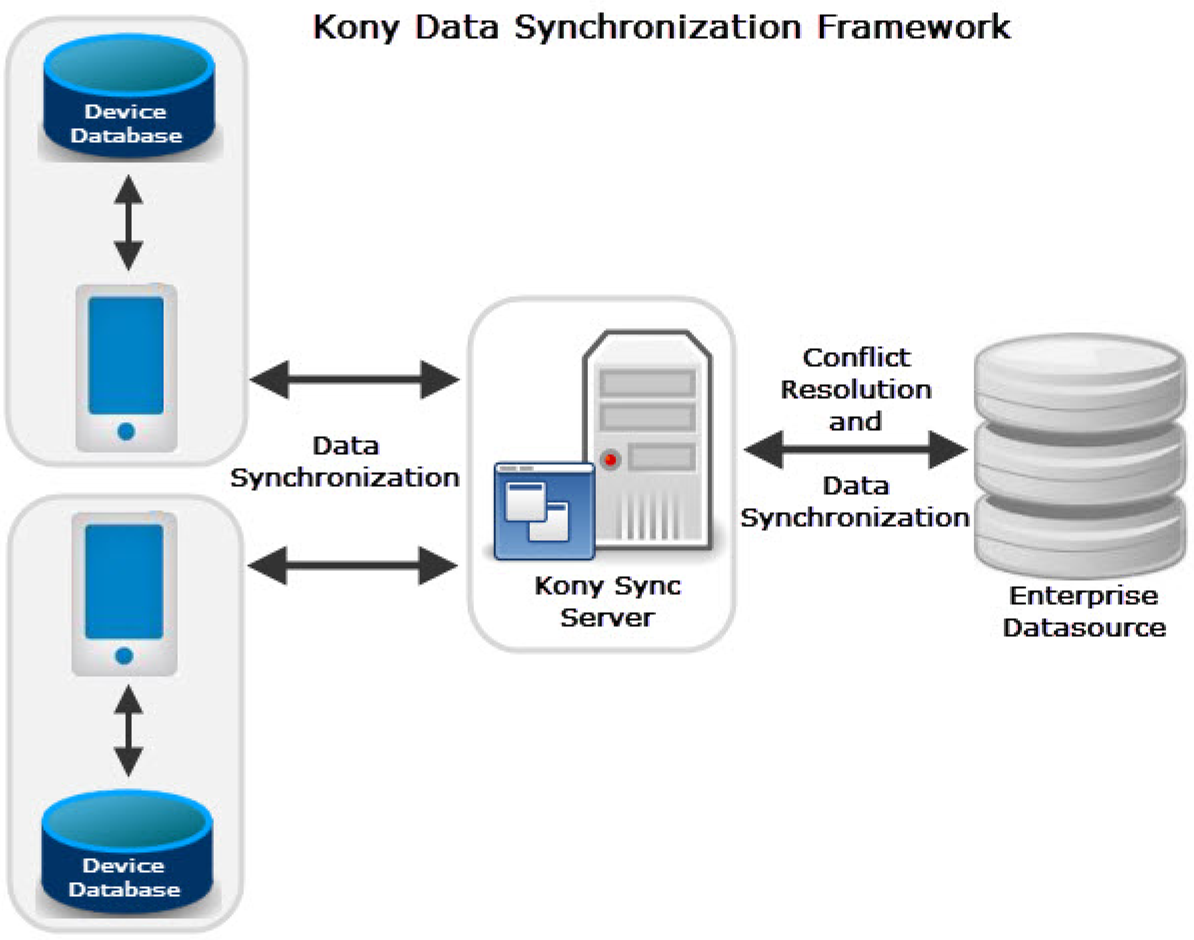
1. Preface

Kony Fabric Sync is a comprehensive synchronization platform that enables developers to add synchronization capabilities to mobile applications. Kony Fabric Sync solves the problem of synchronizing any type of data in any relational database management store using any protocol over any topology. Fundamental to Kony Fabric Sync, is the ability to support offline and collaboration of data between devices to server.

Some of the fundamental features of Kony Fabric Sync are:

- Bi-direction synchronization (changes can happen on both client and server)
- Incremental download (only the data changed on the server from the last time the device synced with the server should be sent to the client)
- Incremental upload (only the data changed on the client from the last time the device synced with the server should be sent to the server)
- Conflict resolution (same data set updated by the client and the server at the same time)
- Define a unit of synchronization similar to a Business Object / Data Object
- Access to the enterprise backend data using existing interfaces like Web Services
- Keep track of objects in a client device so that only you can download the difference between the client device and a backend system to the device (Delta synchronization)
- Allow to define rules for selecting object instances to be downloaded to a client device (Filtering)
- Provide a framework so that an application developer does not need to deal with the complexity of data synchronization
- Provide tools to monitor, process and log/archive synchronization messages

Data synchronization can periodically take information stored in the client database (such as SQLite) and synchronize changes with a server database (such as Oracle Database Server). The advantage of a synchronization-based solution is that users need not require to have a constant network connection to access their information. Since their data is stored locally they are given constant access to their data while offloading processing requirements from the central database. Furthermore, the user is no longer limited by the network speed and can now access the data at the speed of the device.



Note: The term Kony Sync / Kony Sync Server is changed to Kony Fabric Sync. This implies to all the images, text and also the cross-references in this document.

1.1 Purpose

This document helps you familiarize with Kony Fabric Sync and provide procedural information to install and configure the framework, and use the various features that sync offers.

1.2 Intended audience

This document is intended for developers and administrators who use Kony Fabric Sync.

1.3 Formatting Conventions

The following typographical 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.

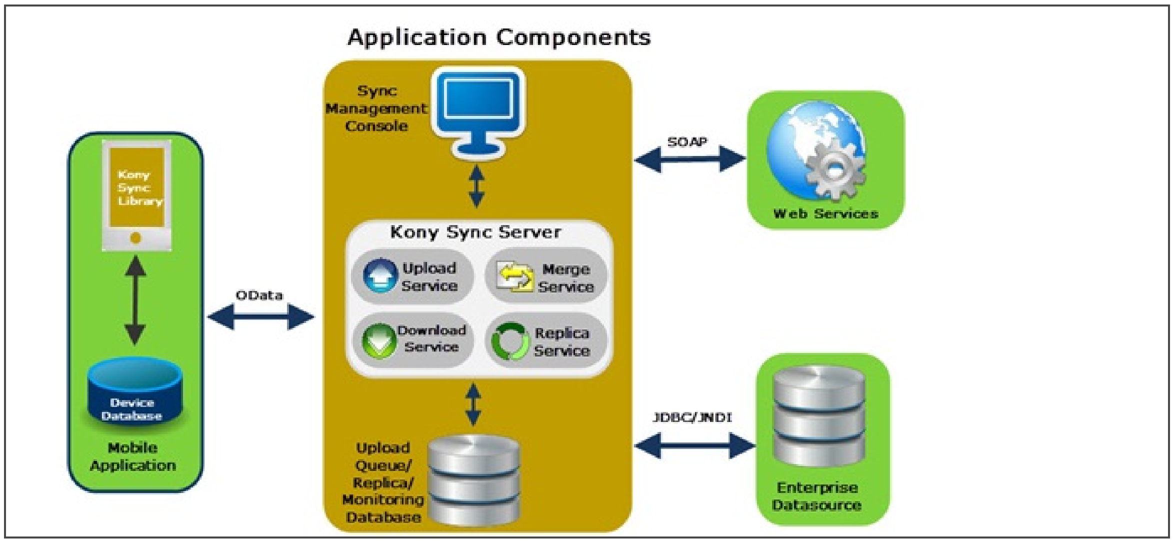
Convention	Explanation
Bold	▪ Windows ▪ Menus ▪ Buttons ▪ Icons ▪ Fields ▪ Tabs.
URL	Active link to a URL
Note	Provides helpful hints or additional information
Important	Highlights actions or information that might cause problems to systems or data

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's product line, contact support@kony.com.

2. Technical Overview

2.1 Architecture



2.2 Kony Fabric Sync

The server has following tasks:

- It is responsible for the queuing the client requests and applying the client changes to backend datasource.
- As part of data synchronization, it identifies data packages to be downloaded for individual mobile devices (data allocation), determines the delta data to be sent to the device (delta data determination), manages conflict situations and provides configuration and monitoring tools.
- It is responsible for loading the appropriate datasource provider and communicating with the backend.

2.3 Kony Fabric Sync Client Library

The client side library has the following tasks:

- Data persistence to the SQLite device database.
- Change tracking / delta determination on the client side.
- Performing data synchronization with the Kony Fabric Sync.
- Logging and tracing.

2.4 Sync Strategies

During offline operation, the mobile application on the device is completely disconnected from the sync. Instances of the same application that may run on other mobile devices at the same time, either connected to or disconnected from the sync. Data stored locally on the mobile device running in offline mode may be modified or deleted or new data may be created. Potentially, the same data entity may be modified by multiple mobile clients concurrently, running either in online or offline operation, with potential effects on the server data store. These modifications that may be in conflict with others, have to be synchronized with the data on the server when a mobile application switches back to online operation. During this process, conflicts need to be reconciled and updates have to be made available to other application clients connected.

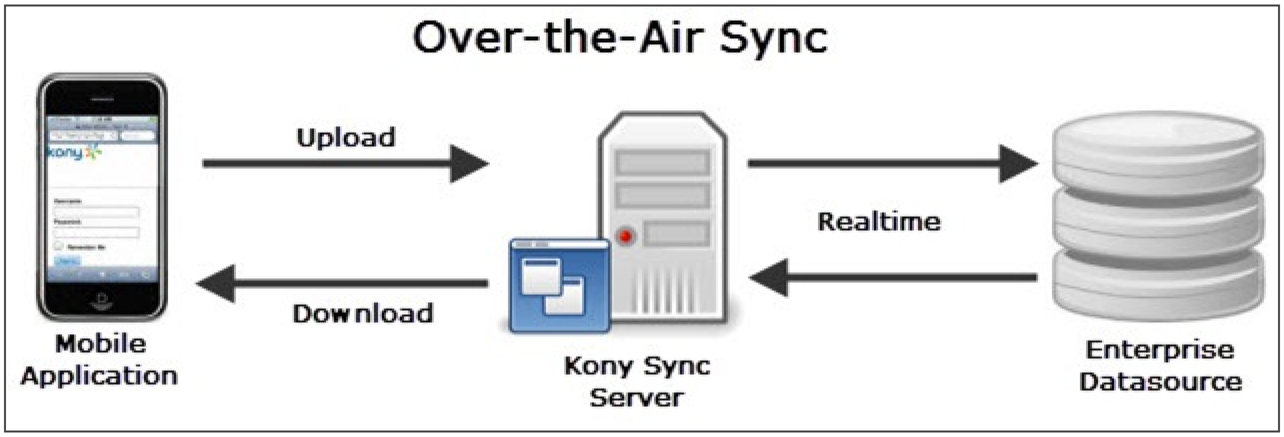
To enable the above synchronization needs the server may choose to store the data before merging it with the Enterprise Datasource. Though this is not a mandatory requirement, it is very desirable when the Enterprise Datasource has not been designed for handling additional mobile users or is only occasionally available.

Kony Fabric Sync provides two flexible sync strategies that help the application developer design the application that best suites the enterprise needs.

2.4.1 Over The Air Sync (OTAsync)

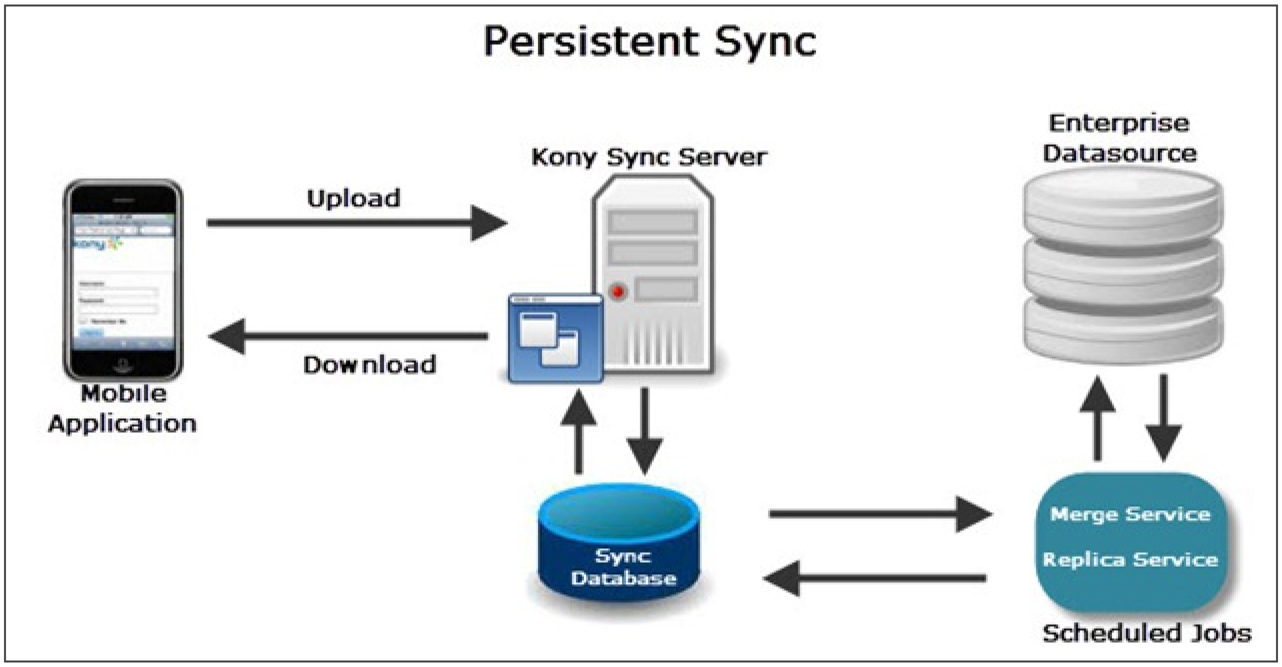
In this strategy whenever the mobile application invokes sync API, the Sync immediately merges upload packets with the enterprise server. Similarly the sync queries the enterprise server for "delta" changes real-time and sends the same to the mobile application.

In this strategy, Enterprise Datasource is assumed to be available when the device starts the sync session.



2.4.2 Persistent Sync

In this strategy the application data is first persisted on Kony Fabric Sync and later merged with the Enterprise Datasource as part of the offline process. Though this results in some data latency but ensures that the system is still functional even though the Enterprise Datasource may not be available.



In this strategy, the Kony Fabric Sync buffers / persists the backend data; during synchronization of a mobile client, the delta for that client is calculated based on the existing data in the sync. You can schedule replication jobs to occur at predetermined intervals to update the sync data. These jobs compare backend data with the data that is persisted in the sync and decide whether an object needs to be updated.

2.4.3 When to Use which Sync Strategy?

Deciding upon to use a Sync Strategy is one of the key architectural / designs decisions that you need to make when developing any enterprise grade offline application. This is not a simple decision but is based on number of input parameters and system constraints. Below are some of the recommendations that can help you determine to choose the appropriate strategy:

OTASync Strategy is recommended solution when:

- The Enterprise backend is highly available to the Sync. It can be assumed that the availability of connectivity between the Device and Sync is the same as the availability of connectivity between the Sync and the Enterprise backend. So whenever the device has network availability it can directly post the changes to the Enterprise backend.

- When there is a need to keep system and operational costs down by not having to replicate the enterprise backend on the sync. Data replication does added to the amount of disk space needed to maintain the data and also resources to monitor the data.
- The Enterprise backend is “Provisioned”. Provisioned is the term used to signify that typical database design patterns are followed like tracking deletes through a soft delete flag and tracking lastupdated timestamp for each data item (or a table row). These elements are essential for OTASync as they are needed for sending appropriate updates back to the client database and keep the processing as low as possible on the sync.
- The Enterprise backend can provide delta changes based on the last updated timestamp as that the device sends. Before we get into the details to understand the need for this, it is important to understand that the essence of OTASync strategy to is to avoid “persisting” a copy of the Enterprise backend data on the sync. During OTASync, the device waits for the client changes to merge with enterprise backend and at the same time, enterprise updates to been sent to the device. Now, if the enterprise backend does not provide the ability to query “deltas”, it means that you have to do the real time delta determination. This means that current state of device data is available on the sync to compare with the current state of the data on the Enterprise backend (so that you can determine the deltas between two sources). Since we don’t persist data on the sync (for OTASync strategy), this means sending a snapshot of the client data with every upload request may increase the pay load of the upload request to an unmanageable level. **SalesForce WebServices offers a very good design pattern on how you should design services to suite the OTASync strategy.**
- Every time the users perform a sync, they get access to the Latest updates on the Enterprise backend.

Persistent Sync is recommended solution when:

- The Enterprise backend is not highly available. For example: the Order Processing system is busy serving desktop users during the peak hours (say 9:00 AM to 5:00 PM) and cannot take additional load of the “new” mobile users (employees accessing the system using mobile devices). So, in order to make sure that the mobile users are still able to submit requests, you have to persist these requests on sync and merge with the Enterprise backend offline (during off peak hours).

- A scheduled system down time does not allow users to access the (typically due to time zone differences)
- The Enterprise backend is “UnProvisioned”. Provisioned is the term used to signify you have to follow typical database design patterns such as tracking deletes through a soft delete flag and tracking lastupdated timestamp for each data item (or a table row). An unprovisioned backend does not follow these characteristics and hence it means that sync has to do delta determination (which dataitems /rows are added new, deleted and updated). When we deal with row items in thousands, this can be a processor intensive and time consuming exercise. Hence this is usually done offline say few times during a day for the entire dataset (data for all users in the system)
- It is acceptable that users are productive even with information set that is “outdated” or “stale” by a few hours. Merging of the data is done usually at periodic intervals, it means that data in the Replica Database can be outdated with respect to the Enterprise backend.

What are the prerequisites for OTASync strategy ?

In order to select the OTASync strategy for a particular SyncScope, ensure that the follow prerequisites are met:

- Enterprise backend is Provisioned. This means that it provides a way to query updated records based on a timestamp and also query the deleted records based on a timestamp.
- It should provide a way to get data items (rows) in batches. This is important as the device has limited memory and can process limited data at any given point in time.

What are the prerequisites for PersistentSync strategy?

- The system provides access to data for all users with a single login / authentication. This is essential since the replica database stores the data for all users and should be able to query the same from the Enterprise backend. The replica service does not refresh data for a particular user. It does across users. For example: it invokes getAll operation on Account objects. This means that it gets all the accounts across users and then tags the one that is updated and that is deleted. The replica service does not have any context of the users that are defined in the systems. The replica service only job is to refresh data periodically from Enterprise backend.

- ◦ Additional storage capacity to be allocated on sync as it now also stores a copy of the Enterprise backend.

2.4.4 Change Tracking

Applications requiring data synchronization capabilities require that changes are tracked in the server database so that these changes can be delivered to clients during a subsequent synchronization session (and vice versa). Below, we discuss different strategies that you can apply / configure to ensure that you can share only incremental updates between client and server.

Kony Fabric Sync provides the ability to either utilize change tracking that the datasource provides or in absence of such capability at the datasource, the Kony Fabric Sync itself can change tracking in the server.

2.4.5 Conflict Resolution

During the synchronization two data stores (local device client and enterprise server), conflicts may occur if the same data object is modified independently in both participating stores. You can detect the conflicts by comparing state information of the two objects. The resolution of a conflict is called reconciliation and is always performed on the enterprise schema.

2.4.5.1 Conflict Detection

Conflict detection happens on "hashsum" generated for every row at sync server during downloads.

When a row is modified at the device, the device sends the same “hashsum” along with the content in the modified row during synchronization.

Sync server will fetch the row from backend with the PK (Primary Key) and regenerates the “hashsum”.

If both the “hashsum” values are different, then the backend data is modified by some other source. Sync server treats this as conflict and applies the appropriate conflict mechanism.

2.4.5.2 Conflict Policies

The existing conflict resolution policies are:

- **Client wins**: The changes on the client side take priority over the changes on the server-side.
- **Server wins**: The changes on the server side take priority over the changes on the client-side.
- **Custom**: The logic or policy defined in the user uploaded interceptor class decides the priority.

Note: The (hard) delete action on the server and client side is ignored (by not considering it a conflict) and no action is performed.

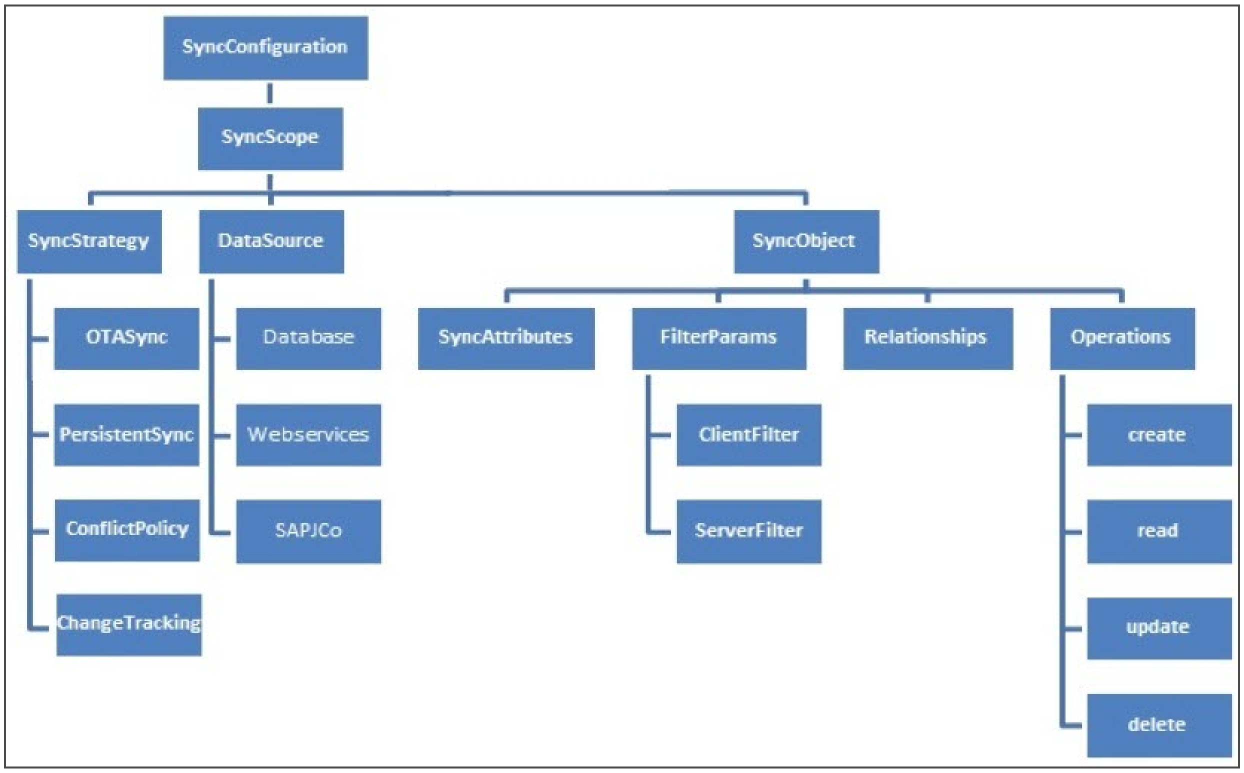
The following table lists all possible scenarios that can arise with the above supported conflict policies.

Client	Server	Conflict type detected	Policy	Action on Data source	Result returned to client-side
Insert (only for non-auto generated Primary key)	Record already exists	Client insert server exists	Client wins	Update with client-side record (Insert converted to update as there is a conflict)	Update event with client-side record
			Server wins	Dummy update	Update event with server-side record

Client	Server	Conflict type detected	Policy	Action on Data source	Result returned to client-side
Update	Delete (soft delete) Soft delete means the record is marked as delete but the record exists in the database.	Client update Server delete	Client wins	Update with client-side record	Update event with client-side record
	Update	Client update Server update	Client wins	Update from client-side	Update event with client-side record
			Server wins	Dummy update	Update event with server-side record
	Delete (soft delete)	Client update Server delete	Client wins	Update from client-side (Update converted to Insert)	Update event with client-side record
			Server wins	No action	Delete event with client-side record
Delete	Update	Client delete Server update	Client wins	Delete is performed	Delete event will be sent
			Server wins	Dummy update	Update event with server-side record

3. Sync Configuration

A Sync Configuration captures details of the data synchronization characteristics of an application. These details are captured in a file typically referred to `SyncConfig.xml` (the name really does not matter) adhering to the `SyncConfig.xsd` schema. A `SyncConfig.xml` represents the below structure.



The two most important elements of this schema are:

3.1 SyncObject

Conceptually, you can consider a SyncObject as a business object that has some public attributes and some methods. The public attributes correspond to the fields visible to client devices, and they are used for synchronization. The methods correspond to the CRUD operations that map to the backend services exposed for the object. The parameter values methods /operations based on both public attribute values.

A SyncObject is meta-data:

- Defining the business object model of an application.
- Defining the way data is exchanged between mobile devices and backend.

A SyncObject is data:

Sync Object data is a business object instance exchanged between client and server.

3.2 SyncScope

A SyncScope groups together the SyncObjects that share common synchronization characteristics like SyncStrategy, Datasource and so on.

A SyncConfiguration can have multiple SyncScopes. It is not possible to define relationships between SyncObjects belonging to different SyncScopes.

4. Enterprise Datasources

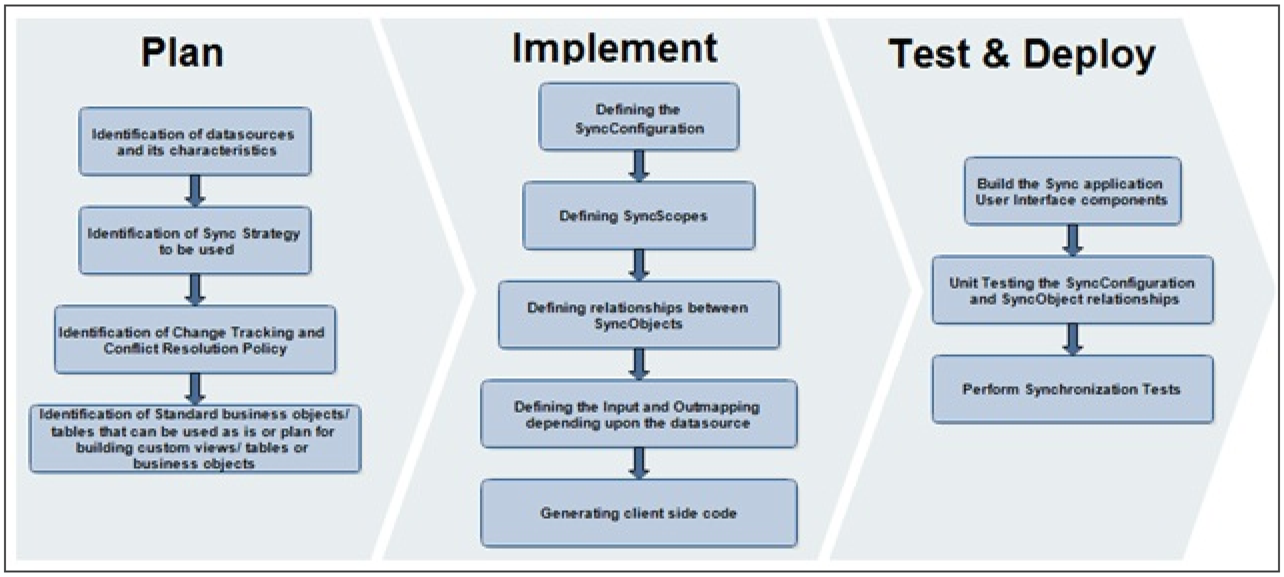
4.1 Database

This release of Kony Fabric Sync supports Database as Enterprise Datasource (For Example: MS SQL, Oracle, MySQL, Postgre SQL and DB2).

4.2 Web Services

This release of Kony Fabric Sync supports SOAP and REST XML or JSON based Web Services as Enterprise Datasource.

5. Application Development


Plan
Identification of datasources and its characteristics
Identification of Sync Strategy to be used
Identification of Change Tracking and Conflict Resolution Policy
Identification of Standard business objects/ tables that can be used as is or plan for building custom views/ tables or business objects
Implement
Defining the SyncConfiguration
Defining SyncScopes
Defining relationships between SyncObjects
Defining the Input and Outmapping depending upon the datasource
Generating client side code
Test & Deploy
Build the Sync application User Interface components
Unit Testing the SyncConfiguration and SyncObject relationships
Perform Synchronization Tests

Data Synchronization application development is not different from the normal software development. Therefore, the best practices for a successful software development also apply to Data Synchronization application development. However, it is also useful to be aware of the additional best practices for the effective and efficient Kony Fabric Sync application development. The key factors for a successful Kony Fabric Sync application development are as follows:

- Design the SyncObjects used in an application based not only on the client application requirements but also on the business objects available in the target backend system(s)
- Design the SyncObjects to only capture the data to show on the device or drive the UI logic on the client side.
- Be aware of the dependencies if developing client- and server-side components in parallel
- Plan short iterative development cycles to ensure that the application is always integrated and that risks are identified and handled as early as possible.

5.1 Planning Phase

Some of the most important artifacts in the planning phase are as follows:

- SyncObjects required in the application are identified and verified that you can map them to the business objects in a backend system as well as they are sufficient to realize the client-side requirements.
- Client GUI design and navigation model are determined and agreed by the stakeholders
- Use cases covered in each iterative cycle in the implementation phase are determined

One of the key decisions in the planning phase is to identify the business objects in the backend system and to decide how they are represented in the client application. Since one of the core functions of an Kony Fabric Sync application is that the business objects updated on a client device is successfully uploaded to a backend system and vice versa, it is very important to identify which business object in a backend system can be used and to identify the dependencies of the business objects on other objects. The identification of the business objects is typically followed by the identification of existing Database Tables, Web Service End Points or SAP BAPIs.

On the client side, the same business object identification process is required based on the requirements. It is especially important to identify whether the downsized or merged version of the business objects in a backend system can be used.

In most cases, the backend wrappers are responsible for absorbing the differences between the client-side business objects and those in a backend system. However, if they are greatly different, it should be carefully investigated whether backend wrappers can really absorb the differences.

Since the GUI requirements of client applications can differ greatly from an application to another depending on the target application users, it is advisable to conduct a preliminary usability test with potential users of the application and to agree on the GUI design and navigation model used in the application as early as possible.

Finally, before moving on to the first development cycle, it is worth planning which use cases will be covered in each development cycle (For more information, refer to Iterative / Use Case Driven Development).

5.2 Implementation Phase

In each development cycle in the implementation phase, the client- and server-side components can be developed in parallel. However, it is important to be aware of the following dependencies:

- To finalize the SyncObjects access logic in the client application, the definition of SyncObjects needs to be completed and the changes in SyncObject definition can affect the client SyncObject access logic.
- When SyncObjects are defined and backend wrappers are implemented, sample data from a backend system can be used for client application standalone testing; before that, test data must be created within the test code of the client application

For efficient parallel development, therefore, it is important to plan the activities accordingly taking these dependencies into account.

5.3 Test / Deployment Phase

At the end of each development cycle, it is important to conduct the integration / synchronization test, which should include the following:

- Synchronization performance benchmark with various data volume
- Performance benchmark of the client application on a target client device
- Application deployment test

Performing the integration / synchronization test at the end of each development cycle is beneficial because it makes it possible to identify issues and risks that should be addressed in the next development cycle.

After the completion of all the development cycles and testing, SyncObjects and backend wrappers are moved to the production system.

Below are the test case scenarios that need to be covered for each of the SyncObjects

- **Initial Data Download Test** (This test simulates the handling of a download service request sent from a client device)
- **Create Test** (After performing the initial download test, the create test should be performed to create a SyncObject instance / record on the client and ensure that the new record does appear in the backend after a successful sync operation)
- **Update Test** (The update test should be performed to modify a SyncObject instance / record on the client and ensure that the update record does appear in the backend after a successful sync operation)
- **Delete Test** (The delete test should be performed to delete a SyncObject instance / record on the client and ensure that the deleted record does get deleted in the backend after a successful sync operation)

6. Installation and Configuration

To start working with Kony Fabric Sync, you need the prerequisites mentioned in the table based on environment. The table provides information of binaries and artifacts required for installation and configuration.

S.no	Software	Reference Documents
1	Tomcat and MS SQL Express Edition (**Automatic**)	• Kony Fabric Sync Installer Linux Guide • Kony Fabric Sync Installer Windows Guide • For detailed information, refer to *Kony Fabric Sync Installation on Windows - Installer* guide in Kony On-Premises Documentation Library.
2	Kony Fabric Sync - Tomcat (**Manual**)	• Kony Fabric Sync Installation and Configuration Guide • For detailed information, refer to *Kony Fabric Sync Installation and Configuration* guide in Kony On-Premises Documentation Library.

3	Weblogic (**Manual**)	• KonyFabric Sync Deployment Steps for Weblogic • For detailed information, refer to *Kony Fabric Sync Installation - Manual - for Weblogic* guide in Kony On-Premises Documentation Library.
4	WebSphere (**Manual**)	• Kony Fabric Sync Installation - Manual - WebSphere • Fore detailed information, refer to *Kony Fabric Sync Installation - Manual - WebSphere* guide in Kony On-Premises Documentation Library.

6.1 Property Files

The following property files are available under **<sync.home>/conf**:

- **SyncConsole**

  The property file contains properties which include:

  - Hibernate Dialects.
  - LDAP.
  - ADS.
  - Salesforce.

    Note: Mention the following keys and provide their values if you want to externalize the Salesforce URL and namespace.

    - salesforce.url
    - salesforce.namespace

- **SyncServices**

  This is the property file which contains the below properties:

  - Database and Connection pool details.
  - Services JNDI details.
  - ReplicaDB and UploadQueueDB.
  - Schedule Jobs.
  - Proxy Settings.

- **Application -Securities**

  This is the property file which contains the below property:

  - Salesforce authentication manager details.

- **SyncLicense**

  This property file contains Sync License properties file for activating new license.

7. Developing an Application

1. Define `SyncConfig.xml` file using Kony Visualizer.
    a. You may refer to the `SampleSyncConfig.xml` file provided along with the Sync Tool (Folder - `/common/resources`).

2. Provide the `SyncConfig.xml` file as an input to SyncTool.
    a. Once defined, copy the `SyncConfig.xml` file to the folder `/common/resources` in the tool.

3. Change `SyncConfig` file name.
    a. Go to `/build` folder and open `build.properties` file to change the SyncConfig file name to the one which you have copied.

4. Run the SyncTool and generate code.
    a. Database as Datasource

       Refer to *Kony Fabric Sync Tool Configuration & Execution Guide* document, for more details available at http://developer.kony.com/Sync

    b. Web Services as Datasource

       Refer to *Kony Fabric Sync WS IDE SyncConfig Generation Guide* document for more details available at http://developer.kony.com/Sync

5. On Build Successful.
    a. Go to `/generated` folder find client & server scripts.
    b. In the *server* folder, you can find `.sql` files.

6. Find generated files

   a. In the `client`, you can find `.lua` and `.js` files.

   b. In the `server` folder, you can find `.sql` files

7. Using client and server scripts

   a. Copy the `.lua` files on to `/modules/lua` Or Copy the `.js` files on to **`modules/js`** folder in your project.

   b. Run the `.sql` scripts in the Sync database to create respective tables.

8. Create client application

   a. ORM API Usage

      Refer to `Kony Fabric Sync Client ORM API` document, for more details about ORM API usage available at http://developer.kony.com/Sync

   b. Auto Generate UI CRUD Forms Using IDE

      Refer to `Kony Fabric Sync Generate Forms`video for a demo available at http://developer.kony.com/Sync

9. Deploy SyncConfig

   a. To deploy the `SyncConfig` file and accomplish assignments.

      Refer to *Kony Fabric Sync Management Console User Guide* document, for more details about deploying `SyncConfig` file, available at http://developer.kony.com/Sync

   b. Open **Management Console** and go to **Applications** tab, and upload the `SyncConfig.xml` file.

   c. Accomplish assignments in the Management Console.
      i. Assign a Device to a User
      ii. Assign the User to a Group
      iii. Assign the Application to the Group

10. Authentication details
    a. Provide authentication details in your client project.
11. Build the Application, deploy and run.
12. Initialize Sync to create local device database.
13. Start Sync Session to download the initial data from Enterprise Datasource.
14. Client side CRUD Operations.
    a. Perform create, update, delete, read operations on the SyncObjects.
15. Upload client changes to Enterprise Datasource.
    a. Start Sync Session to upload the changes to the Enterprise Datasource.

8. Application Administration and Monitoring

8.1 Management Console

Kony Fabric Sync Management Console provides a single point of user interface for configuring and monitoring the Kony Fabric Sync services.

For a step-by-step instructions, refer *Console User Guide* document available at Kony Cloud Documentation Library.

9. Developing a Sample Application

Note: For detailed information, see *Generate Application Forms* section in *Kony Fabric Sync Developing Offline Applications* document and *Kony Fabric Sync ORM API Guide* available at Kony On-Premises Documentation Library.

9.1 Troubleshooting

- **Direct Access to the SQLite database**
  - iPhone(http://blog.chrismiles.info/2011/07/core-data-debugging-with-sqlite.html)
    - Open Terminal and run "sqlite3" pasting in the store path as argument.
    - $ sqlite3 "/Users/chris/Library/Application Support/iPhone Simulator/4.3.2/Applications/22CD429E-ADD2-4AAA-9C9E-5E57828A6FF8/Documents/OrganisingCoreData.sqlite"

      -- Loading resources from /Users/chris/.sqliterc
    - SQLite version 3.6.12
    - Enter ".help" for instructions
    - Enter SQL statements terminated with a ";"
    - sqlite> .tables
  - Android (http://developer.android.com/guide/developing/tools/adb.html#sqlite)
    - $ adb -s emulator-5554 shell
    - # sqlite3

      /data/data/com.example.google.rss.rssexample/databases/rssitems.db
    - SQLite version 3.3.12

- Enter ".help" for instructions
- .... enter commands, then quit...
- sqlite> .exit

9.2 References

For more information on developing a sample application, visit the following links at Kony Cloud Documentation Library.

- Design Guidelines
- Server Planning Guide
- Console User Guide.

10. Using Interceptors

The Interceptor interface provides callbacks capability from the Kony Fabric Sync to the application, allowing the application to inspect and / or manipulate properties of a sync object before it is inserted, updated, deleted, or loaded.

Interceptors can execute code before and after an Action (Upload, Download, and Merge) is invoked. Interceptor classes contain methods that are invoked in conjunction with the methods or life cycle events of the sync services.

10.1 Kony Fabric Sync Interceptors

10.1.1 Upload Interceptors

Using these interceptors custom implementation can manipulate the upload requests from the device and responses to the device.

Methods:

1. **processRequest**

   This method invokes by the sync services for each request from the client. Custom code can modify the properties of the incoming entities, or add/remove the new entities.

2. **processResponse**

   This method invokes by the sync services before sending the response to the client. Custom code can modify the response text by adding any information along with sync response and handle appropriately on client side.

10.1.2 Download Interceptors

Using these interceptors custom implementation can manipulate the download requests from the device and responses to the device.

Methods:

1. **processRequest**

   The sync services invokes this method for each request from the client. Custom code can modify the request coming from the client.

2. **processResponse**

   The sync services invokes this method before sending the response to the client. Custom code can modify the outgoing response string based on the business logic.

10.1.3 Merge Interceptors

Using these interceptors custom implementation can manipulate data in the merge process.

Methods:

processEvent

Merge service invokes this method at various phases while applying the changes to datasource.

10.1.4 Replicate Interceptors

Using these interceptors custom implementation can manipulate data in the replicate process.

Methods:

process Event

The replica service invokes this method at two places.

10.1.5 Notification Interceptors

Using these interceptors custom implementation can manipulate data in the notification events from the Enterprise Datasource.

Methods:

formatNotification

The sync services invokes this method when there a RCT notification from the Datasource.

10.1.6 Conflict Resolution Interceptor

Using these interceptors, you can implement custom conflict resolution at the sync scope level.

Methods:

Resolve Conflict

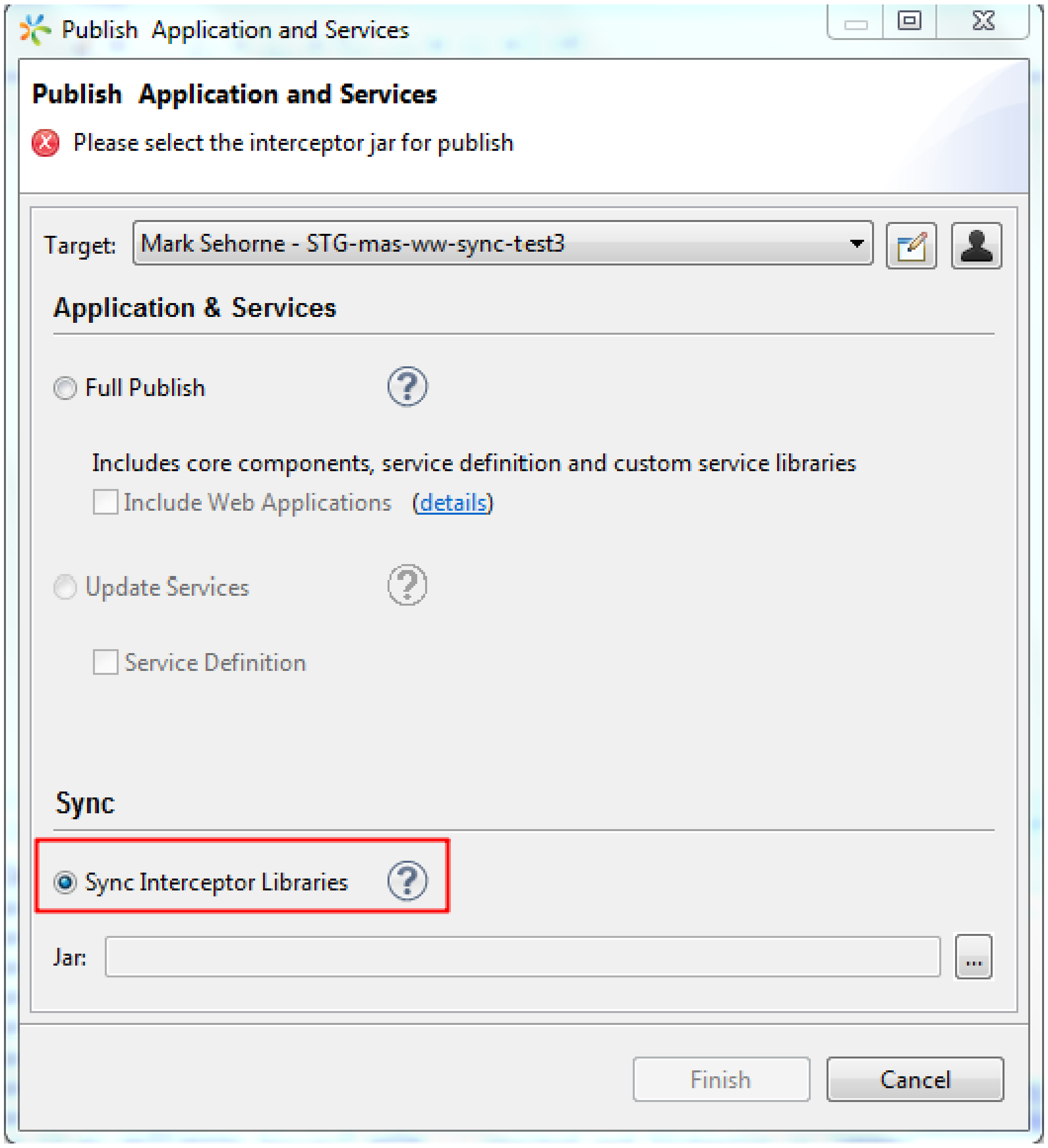
Method to resolve the conflict

10.1.7 Configuring Interceptors

After the custom logic is implemented, follow these steps to configure the interceptor with sync services:

- Make a jar file with all the interceptor implementation class files along with dependent files
- Place the jar file under the `<sync.home>\apache-tomcat-7.0.52\webapps\syncservice\WEB-INF\lib` folder.
- Restart the application server(tomcat/weblogic)
- For cloud, configure your cloud account in Kony Visualizer. While publishing the app, choose

the **Sync Interceptor Libraries** option to upload the jar file.



Publish Application and Services

Please select the interceptor jar for publish

Target: Mark Sehorne - STG-mas-ww-sync-test3

Application & Services

( ) Full Publish

Includes core components, service definition and custom service libraries

[ ] Include Web Applications (details)

( ) Update Services

[ ] Service Definition

Sync

[✓ Sync Interceptor Libraries]

Jar:

10.1.7.1 Example for Interceptor in Sync Configuration File

```
<SyncInterceptors>
        <UploadInterceptors>
                 <Interceptor
```

```
ClassName="com.kony.sync.interceptors.logging.LoggingUploadIntercept
or"/>
       </UploadInterceptors>
       <DownloadInterceptors>
               <Interceptor
ClassName="com.kony.sync.interceptors.logging.LoggingDownloadInterce
ptor"/>
       </DownloadInterceptors>
       <MergeInterceptors>
               <Interceptor
ClassName="com.kony.sync.interceptors.logging.LoggingMergeIntercepto
r"/>
       </MergeInterceptors>
       <ReplicaInterceptors>
               <Interceptor
ClassName="com.kony.sync.interceptors.logging.LoggingReplicaIntercep
tor"/>
       </ReplicaInterceptors>
       <NotificationInterceptors>
               <Interceptor
ClassName="com.kony.sync.interceptors.logging.LoggingNotificationInt
erceptor"/>
       </NotificationInterceptors>
</SyncInterceptors>
```

10.1.7.2 Example for Conflict Interceptor in Sync Configuration

```
<ConflictPolicy>
       <Type>custom</Type>
       <ProvisioningColumnsSupported>
               <UpdateUserID>false</UpdateUserID>
       </ProvisioningColumnsSupported>
       <ConflictInterceptor
ClassName="com.kony.sync.interceptors.logging.ConflictInterceptorImp
l"/>
</ConflictPolicy>
```

11. Filters

Kony Fabric Sync provides capability to apply filters to download specific data from Enterprise Datasource to the device.

There two ways a developer can apply filters over the data downloads. Filters need to be defined in the `SyncConfig` file on the SyncObject.

The below example illustrates, applying filters on SyncObject **Orders**, **for CustomerID (Client Filter)** and **ShipCountry (Server Filter)**SyncAttributes also applying filters on Parent SyncObject applies the respective filters on child SyncObject based on the relationship defined.

Client Filters are applied on the data that is downloaded based on the Server Filters.

```
<SyncObject SourceName=”Orders” GlobalName=”Orders”
SourceSchema=”dbo”>
        <SyncAttributes>
                <Key>
                        <Attribute GlobalName="OrderID"/>
                </Key>
                <SyncAttribute SourceName="OrderID" GlobalName="OrderID"
                Type="int" IsNullable="false" Autogenerated="true"/>
                <SyncAttribute SourceName="CustomerID" GlobalName="CustomerID"
                Type="string" IsNullable="true" Autogenerated="false"/>
                <SyncAttribute SourceName="EmployeeID" GlobalName="EmployeeID"
                Type="int" IsNullable="true" Autogenerated="false"/>
                <SyncAttribute SourceName="OrderDate" GlobalName="OrderDate"
                Type="date" IsNullable="true" Autogenerated="false"/>
                <SyncAttribute SourceName="ShipCountry"
                GlobalName="ShipCountry" Type="string" IsNullable="true"
                Autogenerated="false"/>
                <SyncAttribute SourceName="LastUpdateTime"
                GlobalName="LastUpdateTime"
                Type="string" IsNullable="false" Autogenerated="false"/>
                <SyncAttribute SourceName="UpdateUserID"
                GlobalName="UpdateUserID"
                Type="string" IsNullable="false"
                Autogenerated="false"/>
                <SyncAttribute SourceName="SoftDeleteFlag"
                GlobalName="SoftDeleteFlag"
                Type="boolean" IsNullable="false"
                Autogenerated="false"/>
        </SyncAttributes>
        <Relationships>
                <OneToMany TargetObject="OrderDetails"
                 TargetObjectAttribute="OrderID"
                 SourceObjectAttribute="OrderID" Cascade="true"/>
        </Relationships>
        <FilterParameters>
                <ClientFilterParameters>
                        <FilterParameter Name="CustomerID" Condition="EQ"
                               Value1="Customer1"/>
                </ClientFilterParameters>
                <ServerFilterParameters>
   <FilterParameter Name="ShipCountry" Value1="USA"
                               Condition="LIKE"/>
                </ServerFilterParameters>
        </FilterParameters>
        <UpdateTimeStampAttribute GlobalName="LastUpdateTime"/>
        <UpdateUserIDAttribute GlobalName="UpdateUserID"/>
        <SoftDeleteFlagAttribute GlobalName="SoftDeleteFlag"/>
        <Operations/>
</SyncObject>
<SyncObject  SourceName="OrderDetails" GlobalName="OrderDetails"
SourceSchema="dbo">
        <SyncAttributes>
                <Key>
                        <Attribute GlobalName="OrderID"/>
                        <Attribute GlobalName="ProductID"/>
                </Key>
                <SyncAttribute SourceName="OrderID" GlobalName="OrderID"
                Type="int" IsNullable="false"
                 Autogenerated="false"/>
                <SyncAttribute SourceName="ProductID" GlobalName="ProductID"
                Type="int" IsNullable="false"
                 Autogenerated="false"/>
                <SyncAttribute SourceName="UnitPrice" GlobalName="UnitPrice"
                Type="string" IsNullable="false" Autogenerated="false"/>
                <SyncAttribute SourceName="Quantity" GlobalName="Quantity"
                Type="short" IsNullable="false" Autogenerated="false"/>
                <SyncAttribute SourceName="Discount" GlobalName="Discount"
                Type="string" IsNullable="false" Autogenerated="false"/>
                <SyncAttribute SourceName="LastUpdateTime"
                GlobalName="LastUpdateTime"
                Type="string" IsNullable="false" Autogenerated="false"/>
                <SyncAttribute SourceName="UpdateUserID"
                GlobalName="UpdateUserID"
                Type="string"
                IsNullable="false" Autogenerated="false"/>
                <SyncAttribute SourceName="SoftDeleteFlag"
                GlobalName="SoftDeleteFlag"
                Type="boolean" IsNullable="false" Autogenerated="false"/>
        </SyncAttributes>
        <Relationships>
        </Relationships>
        <FilterParameters>
                <ClientFilterParameters>
                </ClientFilterParameters>
                <ServerFilterParameters>
                </ServerFilterParameters>
        </FilterParameters>
        <UpdateTimeStampAttribute GlobalName="LastUpdateTime"/>
        <UpdateUserIDAttribute GlobalName="UpdateUserID"/>
        <SoftDeleteFlagAttribute GlobalName="SoftDeleteFlag"/>
        <Operations/>
</SyncObject>
```

12.

13. Authentication Service

You can perform the following modes of authentication on sync from client side.

13.1 Authenticate the User

In this mode we authenticate the userid with the given password. Call the service at the following url.

http://<Host-Name>:<PORT>/syncservice/resources/authenticateUser

The Content-Type should be set to “application/json” and pass the following parameters to the service

- userid
- password

13.2 Authenticate the Application

In this mode we check whether the userid and password is valid and the user is authorized to access the specified application.

Call the service at the following url:

http://<Host-Name>:<PORT>/syncservice/resources/authenticateUser

The Content-Type should be set to “application/json” and pass the following parameters to the service

- userid
- password
- appid

13.3 Authenticate the Device

In this mode we check whether the userid and password is valid and the user is authorized to specified device and application.

Call the service at the following url:

http://<Host-Name>:<PORT>/syncservice/resources/authenticateUser

Content-Type should be set to "application/json" and pass the following parameters to the service

- userid
- password
- appid
- deviceid

13.3.1 JavaScript Example

The following sample in js calls the Authenticate Service based on the inputs provided.

```
function onLogIn(){
       var gblUserId = frmLogin.txtUserId.text;
       var gblPassword= frmLogin.txtPwd.text;
       var httpheaders = []
       httpheaders[ kony.decrement("Content-Type") ] = "application/json";

       if(gblUserId != "" && gblPassword != ""){
               var params = {userid:gblUserId, password:gblPassword, appId :
               gblAppId};
               params[ kony.decrement("httpheaders") ] = httpheaders;
               var res;
               var url =
               http://127.0.0.1:8081/syncservice/resources/authenticateUser";
               res = kony.net.invokeService(url, params, true);
               var basicConf =
               {message: res["msg"],alertType: constants.ALERT_TYPE_INFO
               ,alertTitle: "",yesLabel:"Ok",noLabel: "", alertHandler: null};
               var pspConf = {};
```

```
                kony.ui.Alert(basicConf,pspConf);
        }else{
                var basicConf = {message: "User Name and Password is
                manadatory",alertType:
                constants.ALERT_TYPE_ ERROR ,alertTitle: "",
                yesLabel:"Ok",noLabel: "", alertHandler:
                null};
                var pspConf = {};
                kony.ui.Alert(basicConf,pspConf);
        }
}
```

14. Supporting ADS in Kony Fabric Sync

Spring security 3.1 framework is used to connect to Microsoft ADS. In GA 5.0 we have the support for using Microsoft ADS.

To enable ADS Support, follow these steps:

1. In `conf\syncconsole.properties` you need to set the below properties of ADS server:

```
#ADS connection settings
ads.ldap.url=ldap://platformtest.kony.internal:389/

ads.ldap.url.doamin=
ldap
://platformtest.kony.internal:389/DC=
platformtest,DC=kony,DC=internal
ads.ldap.domain=platformtest.kony.internal
ads.ldap.userDn=administrator@platformtest.kony.internal
ads.ldap.password=C@t@10gUe
ads.ldap.referral=follow
```

Note: You need to change the above values as per deployment environment.

2. You need to add the user in ADS syncconsole DB as well. As password field is mandatory. A dummy password can be inserted through the management console while creating the user. Authentication is done using ADS and not syncconsole DB.

    a. Assign one of the roles (ROLE_ADMIN/ROLE_REPORT_VIEWER/ROLE_USER) to above user

    b. Assign Group

Note: Kony Fabric Sync uses ADS or open ldap to authenticate a user. But Authorization is performed using syncconsole DB.

3. Update the file: `WEB-INF\applicationContext-security.xml` located in `syncconsole.war`.

   Uncomment the below tag to enable ADS Support:

```
<! -- Uncomment to Enable LDAP
<security:authentication-provider ref="ldapAuthProvider" />
<security:authentication-provider ref="ldapActiveDirectoryAuthProvider"
/>
-->
```

Note:

- <security:authentication-provider ref="ldapAuthProvider" /> is for open ldap
- `<security:authentication-provider ref="ldapActiveDirectoryAuthProvider" /> is for ADS`Three

15. DataSources

15.1 Databases

15.1.1 Sample XML tags for MySQL, Oracle, MSSQL, PostgreSQL, and DB2 server

Below are the XML tag examples for various databases that are useful to define the database datasource in the Sync Configuration file.

MS SQL Server

```
<DataSource ID="KonySyncSample" Type="database"
Class="com.kony.sync.services.datasource.jdbc.JDBCDatasource2">
          <Database Name="KonySyncSampleConfig" Type="MSSQLSERVER"
IsJNDIDataSource="false"

JndiORJdbcURL="jdbc:sqlserver://localhost:1433;databaseName=KonySync
Sample;user=sa;password=kony123!"

DriverName="com.microsoft.sqlserver.jdbc.SQLServerDriver"/>
</DataSource>
```

Oracle

```
<DataSource ID="KonySyncSample" Type="database"
Class="com.kony.sync.services.datasource.jdbc.JDBCDatasource2">
       <Database Name="KonySyncSampleConfig" Type="ORACLE"
IsJNDIDataSource=
        "false"

JndiORJdbcURL="jdbc:oracle:thin:system/kony123!@localhost:1521:xe"
        DriverName="oracle.jdbc.driver.OracleDriver"/>
</DataSource>
```

MySQL

```
<DataSource ID="KonySyncSample" Type="database"
Class="com.kony.sync.services.datasource.jdbc.JDBCDatasource">
       <Database Name="KonySyncSampleConfig" Type="MYSQL"
IsJNDIDataSource=
        "false"

JndiORJdbcURL="jdbc:mysql://localhost:3306/konysyncsample?user=root&
amp;password=kony123!"
        DriverName="com.mysql.jdbc.Driver"/>
</DataSource>
```

PostgreSQL

```
<DataSource ID="KonySyncSample" Type="database"
Class="com.kony.sync.services.datasource.jdbc.JDBCDatasource">
        <Database Name="KonySyncSampleConfig" Type="POSTGRESQL"
IsJNDIDataSource="false"
JndiORJdbcURL="jdbc:postgresql://localhost:5432/konyunittestdatabase
?user=postgres&amp;password=kony123!"
DriverName="org.postgresql.Driver"/>
   </DataSource>
```

DB2

```
<DataSource
Class="com.kony.sync.services.datasource.jdbc.JDBCDatasource2"
ID="db2" Type="database">
     <Database DriverName="com.ibm.db2.jcc.DB2Driver"
IsJNDIDataSource="false"
```

```
JndiORJdbcURL="jdbc:db2://localhost:50000/SYNCDB:currentSchema=KONYS
YNCSAMPLE;"
     Name="dbclient" Password="kony123" Type="DB2"
UserName="db2admin">
               <ConnectionProps/>
           </Database>
</DataSource>
```

15.2 Other Datasources

15.2.1 Operations

Create: This operation is required when user creates new records from the device. When the change type from the device for a row is “insert”, sync engine invokes the service that is mapped for Create operation.

Update: This operation is required when user updates records from the device. When the change type from the device for a row is “update”, sync engine invokes the service that is mapped for Update operation.

Delete: This operation is required when user deletes records from the device. When the change type from the device for a row is “delete”, sync engine invokes the service that is mapped for Delete operation.

Get: Sync engine uses this operation to fetch the latest data from the datasource. Before invoking the Update / Delete on datasource, Get is invoked to check whether there are any conflicts. If Get operation is not defined, then conflicts are not supported. Sync engine will also use this operation during download to fetch the data from the datasource for each row when "getUpdated" operation does not return all the column data.

For example, if "getUpdated" operations returns only the list of primary keys, then sync engine will internally invoke the ‘get’ operation for each primary key and accumulate all the data and send it device.

GetUpdated: This operation is invoked to download the data from datasource to the device/replica. This operation has to accept the LAST_SYNC_TIMESTAMP from the context variables to get the incremental updates.

GetDeleted: This operation is invoked to download the data that is marked as deleted in the datasource to the device / replica. This is an optional operation, and it is not needed when datasource has no delete functionality. If SoftDeleteFlag mapped in “getUpdated” operation output mapping, then also it is not required to define the “getDeleted” operation.

GetAll: Sync engine uses this operation to get all the data for the sync object when the datasource is unprovisioned.

GetBatch: Sync engine invokes this operation to fetch the data from the datasource in batches.

BATCH_OFFSET: Sync engine sets this context parameter into the context for the input mapping. This value contains an integer value indicating the number of rows downloaded so far in the current sync object. This parameter can be mapped when backend services are using pagination based batching (for example, batching with LIMIT and OFFSET). In case of pagination batching you can map the BATCH_SIZE to LIMIT and BATCH_OFFSET to OFFSET.

15.2.2 Bulk Upload Operation

While uploading the data from device kony sync has capability to upload the rows in batches. But using normal create/update/delete operations, sync server will upload single row in each service call to back-end data source, which is causing delay (Sometimes leading to Time Out issue) in uploading the device data and increasing the overall sync time. So to overcome this limitation we have provided bulk upload capability from sync server to back-end data source.

Sync server can upload the multiple rows through single service to the back-end data source when the back-end service supports the multiple rows during uploads. This is applicable only for SOAP/XML/JSON services only.

To achieve this you need to map the ‘Bulk’ operations with the services which accepts multiple rows. Sync server will consolidate the rows which belongs to one sync object and same change type (create, update, delete) into a collection of records and invoke the configured bulk service with that collection.

Example: If the device upload request has an update to 'Account' row and 'BulkUpdate' operation is configured at 'Account' sync object, then sync server will look for other 'Account' rows in the device upload request which has same change type as 'Update', and create a collection of records and invoke the configured 'BulkUpdate' service.

As service need to accept multiple rows, its input parameter should be of type **collection** and you should have defined the request template with '#foreach' and used the sync attribute names as inner parameters with in '#foreach' block.

For more information on **Collection** datatype, refer Adding a collection to SOAP Service.

> ***Note:*** Configured service should expect the 'collection' primary keys and return the rows which matches the input primary keys.

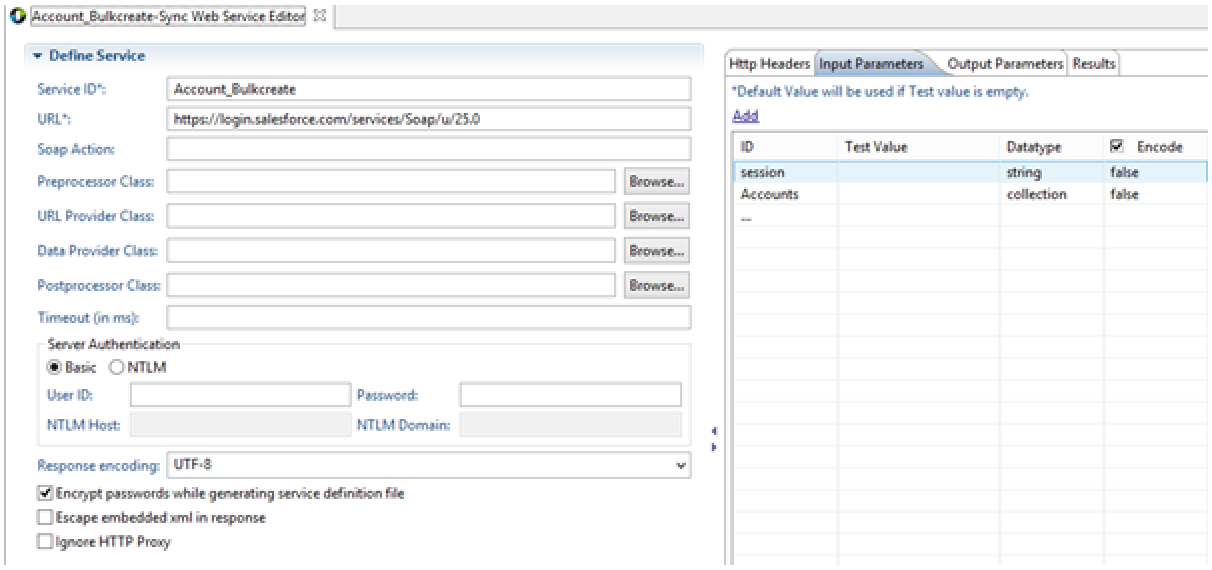
15.2.3 Bulk Create Operation

The **BulkCreate** operation is used when developer wants to upload all the records with change type as 'insert' from the device. The request should go to back-end datasource with a single service invocation.

The following example for all operations, we take Salesforce as back-end with OTA Sync Strategy for Account sync object.

Example:

- Service configuration (`Account_Bulkcreate`) for BulkCreate operation of Account sync object.



Account_Bulkcreate-Sync Web Service Editor

Define Service

Service ID*:	Account_Bulkcreate
URL*:	https://login.salesforce.com/services/Soap/u/25.0
Soap Action:	
Preprocessor Class:	
URL Provider Class:	
Data Provider Class:	
Postprocessor Class:	
Timeout (in ms):	

Server Authentication: [✓ Basic] NTLM

User ID:		Password:	
NTLM Host:		NTLM Domain:	

Response encoding: UTF-8

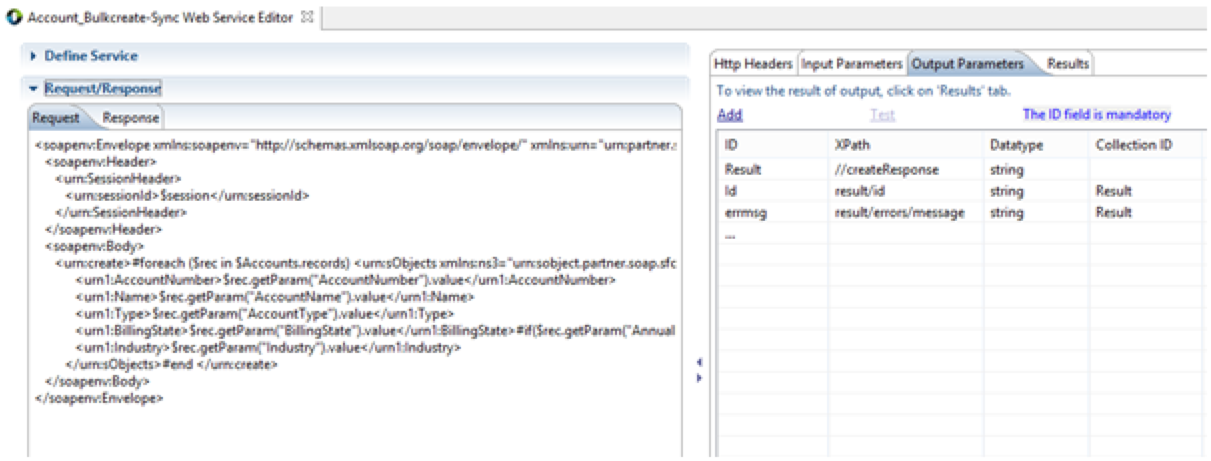
[✓] Encrypt passwords while generating service definition file
[ ] Escape embedded xml in response
[ ] Ignore HTTP Proxy

Http Headers | Input Parameters | Output Parameters | Results

*Default Value will be used if Test value is empty.

Add

ID	Test Value	Datatype	Encode
session		string	false
Accounts		collection	false
...			



Account_Bulkcreate-Sync Web Service Editor

Define Service

Request/Response

Request | Response

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:partner.
  <soapenv:Header>
    <urn:SessionHeader>
      <urn:sessionId>$session</urn:sessionId>
    </urn:SessionHeader>
  </soapenv:Header>
  <soapenv:Body>
    <urn:create>#foreach ($rec in $Accounts.records) <urn:sObjects xmlns:ns3="urn:sobject.partner.soap.sfc
        <urn1:AccountNumber>$rec.getParam("AccountNumber").value</urn1:AccountNumber>
        <urn1:Name>$rec.getParam("AccountName").value</urn1:Name>
        <urn1:Type>$rec.getParam("AccountType").value</urn1:Type>
        <urn1:BillingState>$rec.getParam("BillingState").value</urn1:BillingState>#if($rec.getParam("Annual
        <urn1:Industry>$rec.getParam("Industry").value</urn1:Industry>
      </urn:sObjects>#end </urn:create>
  </soapenv:Body>
</soapenv:Envelope>
```

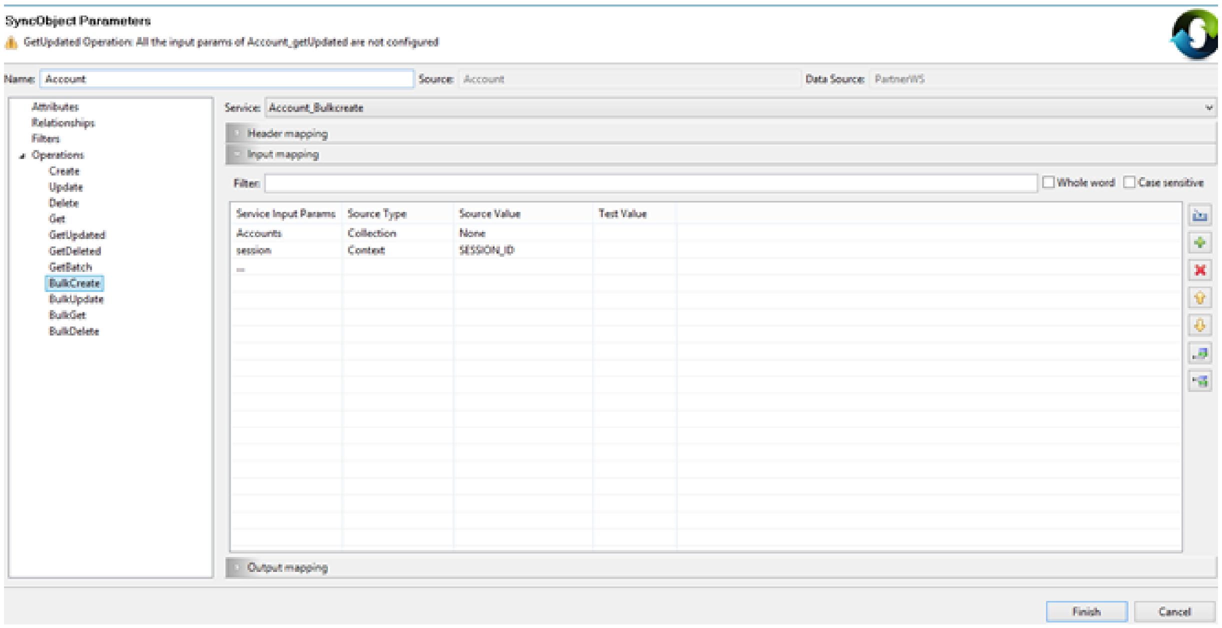
Http Headers | Input Parameters | Output Parameters | Results

To view the result of output, click on 'Results' tab.

Add Test The ID field is mandatory

ID	XPath	Datatype	Collection ID
Result	//createResponse	string	
Id	result/id	string	Result
errmsg	result/errors/message	string	Result
...			

- Mapping BulkCreate operation with `Account_Bulkcreate` service for Account Sync object.



SyncObject Parameters

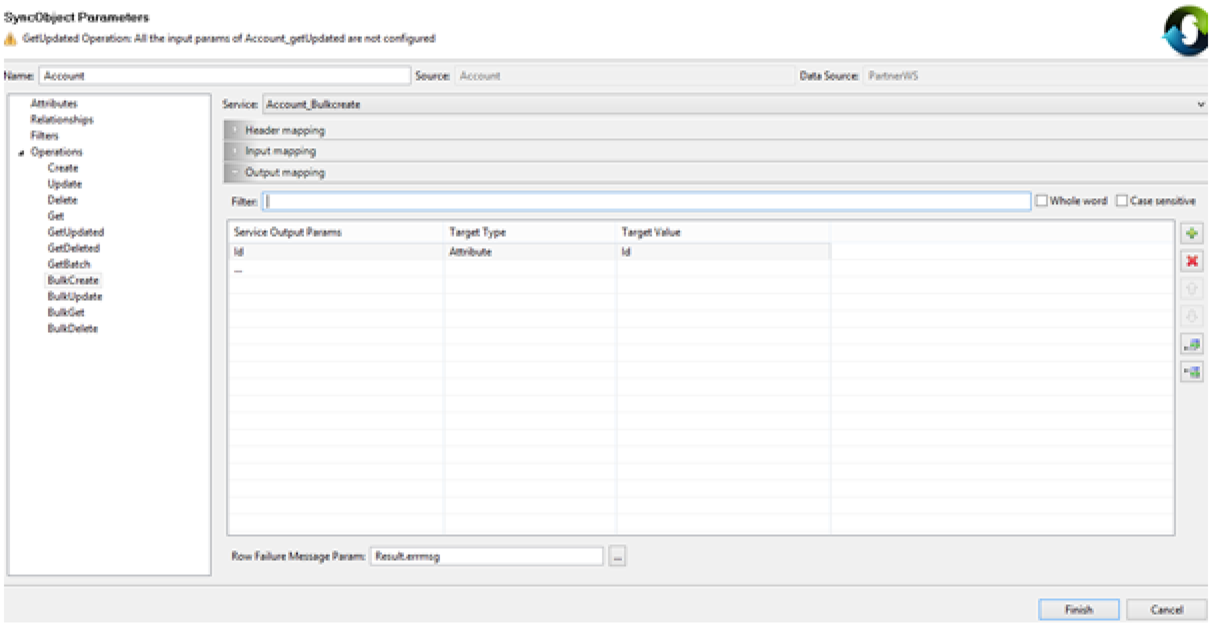
GetUpdated Operation: All the input params of Account_getUpdated are not configured

Name: Account Source: Account Data Source: PartnerWS

Service: Account_Bulkcreate

Filter:

Service Input Params	Source Type	Source Value	Test Value
Accounts	Collection	None	
session	Context	SESSION_ID	
...			



SyncObject Parameters

GetUpdated Operation: All the input params of Account_getUpdated are not configured

Name: Account Source: Account Data Source: PartnerWS

Service: Account_Bulkcreate

Filter:

Service Output Params	Target Type	Target Value
Id	Attribute	Id
...		

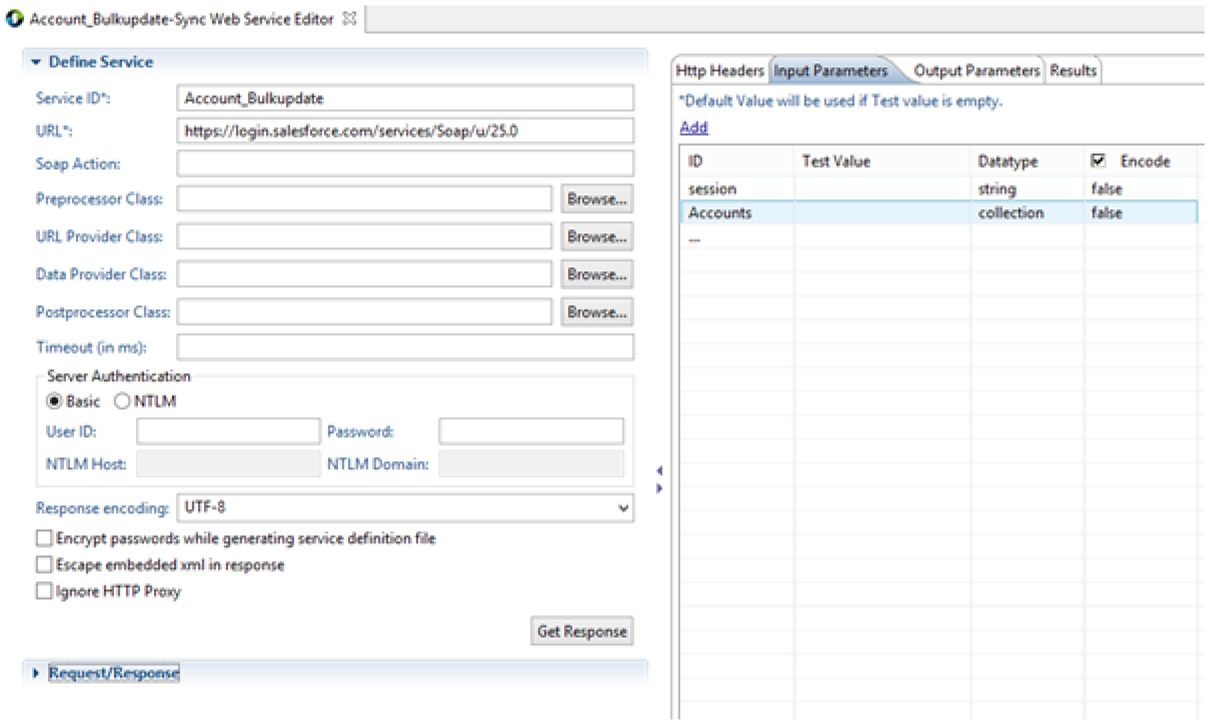
Row Failure Message Param: Result.errmsg

15.2.4 BulkUpdate Operation

This **BulkUpdate** operation is used when developer want to upload all the records with change type as 'update' from the device. The request should go to back-end data source with a single service invocation.

Example:

- Service configuration (`Account_Bulkupdate`) for BulkUpdate operation of Account sync object.



Account_Bulkupdate-Sync Web Service Editor

Define Service

Service ID*:	Account_Bulkupdate
URL*:	https://login.salesforce.com/services/Soap/u/25.0
Soap Action:	
Preprocessor Class:	
URL Provider Class:	
Data Provider Class:	
Postprocessor Class:	
Timeout (in ms):	

Server Authentication: [✓ Basic] NTLM

User ID:		Password:	
NTLM Host:		NTLM Domain:	

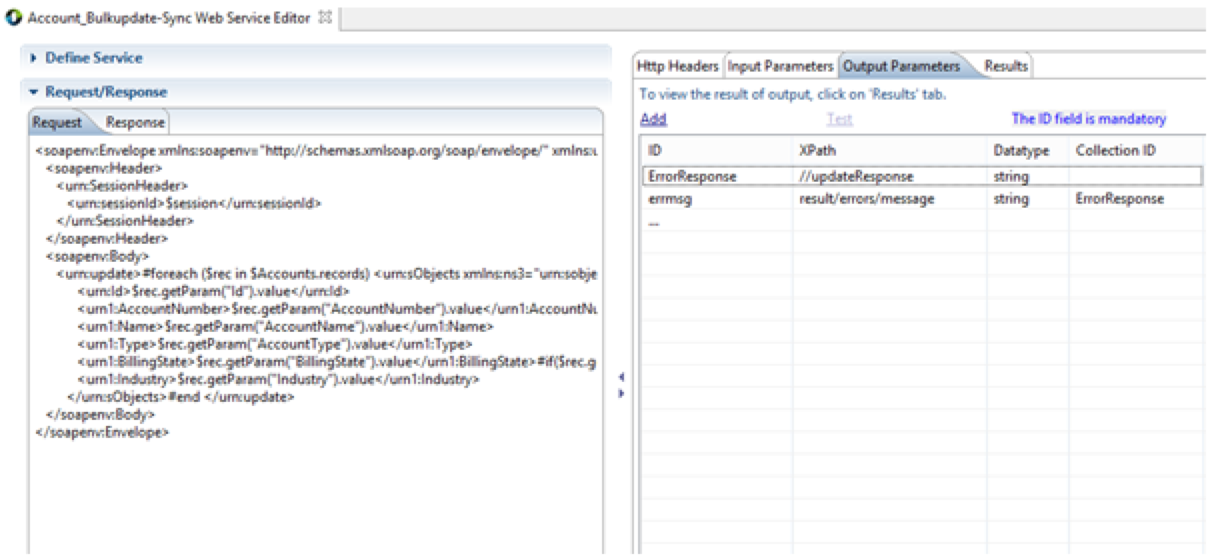
Response encoding: UTF-8

☐ Encrypt passwords while generating service definition file
☐ Escape embedded xml in response
☐ Ignore HTTP Proxy

Input Parameters

*Default Value will be used if Test value is empty.

ID	Test Value	Datatype	☑ Encode
session		string	false
Accounts		collection	false
...			



Account_Bulkupdate-Sync Web Service Editor

Request/Response — Request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:u
  <soapenv:Header>
    <urn:SessionHeader>
      <urn:sessionId>$session</urn:sessionId>
    </urn:SessionHeader>
  </soapenv:Header>
  <soapenv:Body>
    <urn:update>#foreach ($rec in $Accounts.records) <urn:sObjects xmlns:ns3="urn:sobje
        <urn:Id>$rec.getParam("Id").value</urn:Id>
        <urn1:AccountNumber>$rec.getParam("AccountNumber").value</urn1:AccountNu
        <urn1:Name>$rec.getParam("AccountName").value</urn1:Name>
        <urn1:Type>$rec.getParam("AccountType").value</urn1:Type>
        <urn1:BillingState>$rec.getParam("BillingState").value</urn1:BillingState>#if($rec.g
        <urn1:Industry>$rec.getParam("Industry").value</urn1:Industry>
      </urn:sObjects>#end </urn:update>
  </soapenv:Body>
</soapenv:Envelope>
```

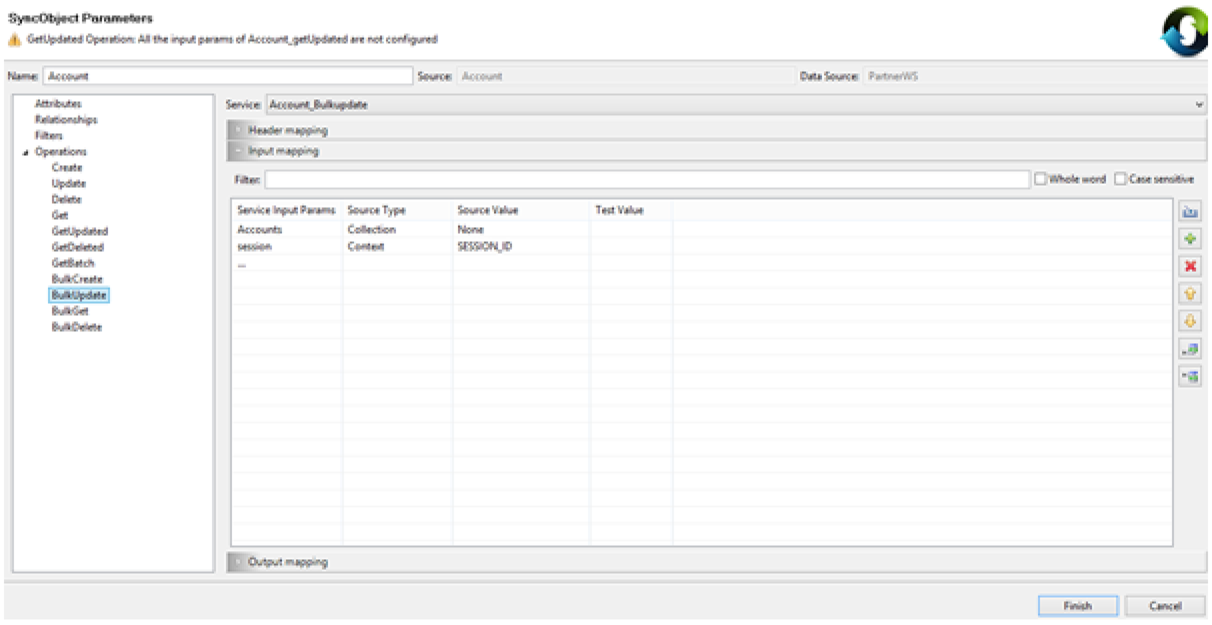
Output Parameters

To view the result of output, click on 'Results' tab.

The ID field is mandatory

ID	XPath	Datatype	Collection ID
ErrorResponse	//updateResponse	string	
errmsg	result/errors/message	string	ErrorResponse
...			

- Mapping BulkUpdate operation with `Account_Bulkupdate` service for Account Sync object.



SyncObject Parameters

GetUpdated Operation: All the input params of Account_getUpdated are not configured

Name: Account
Source: Account
Data Source: PartnerWS

Service: Account_Bulkupdate

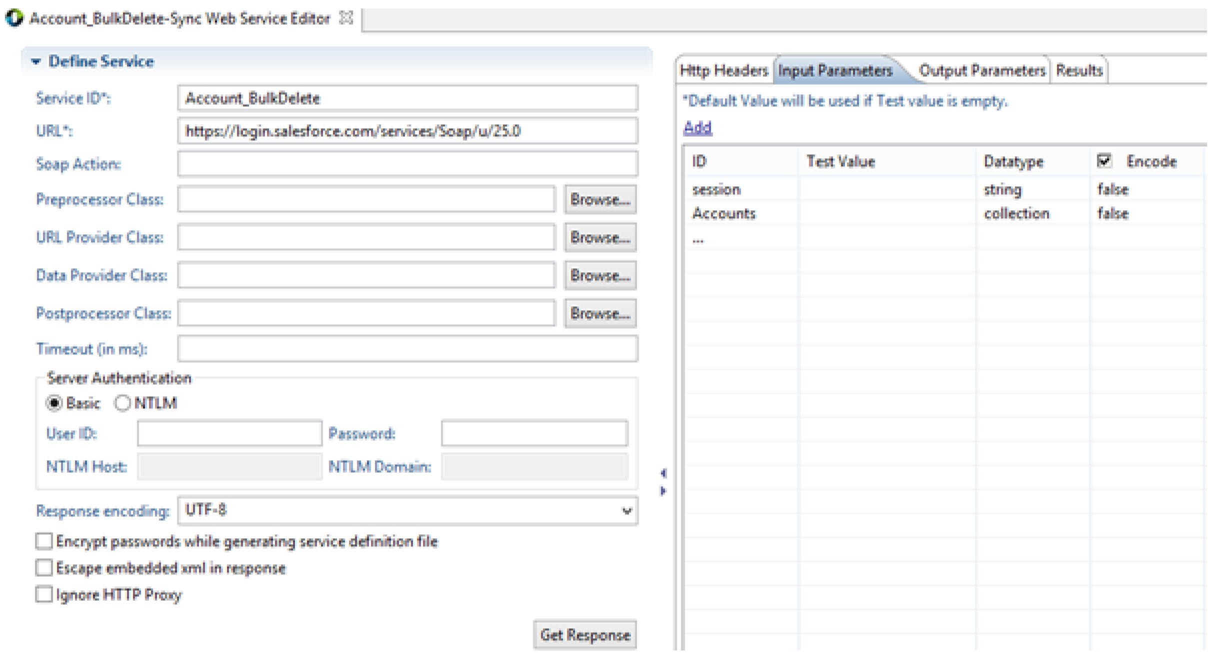
Service Input Params	Source Type	Source Value	Test Value
Accounts	Collection	None	
session	Context	SESSION_ID	
...			

15.2.5 BulkDelete Operation

This operation is required when developer wants to upload all the records with change type as 'delete' from the device. The request should go to back-end data source with a single service invocation.

Example:

- Service configuration (`Account_Bulkdelete`) for BulkDelete operation of Account Sync object.



Account_BulkDelete-Sync Web Service Editor

Define Service

Service ID*:	Account_BulkDelete
URL*:	https://login.salesforce.com/services/Soap/u/25.0
Soap Action:	
Preprocessor Class:	
URL Provider Class:	
Data Provider Class:	
Postprocessor Class:	
Timeout (in ms):	

Server Authentication: [✓ Basic] NTLM

User ID:		Password:	
NTLM Host:		NTLM Domain:	

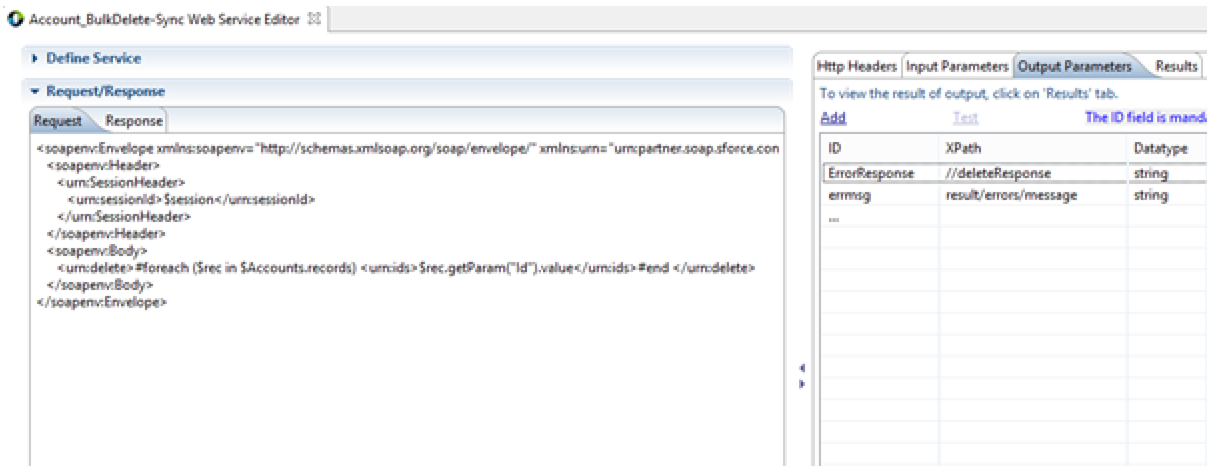
Response encoding: UTF-8

- [ ] Encrypt passwords while generating service definition file
- [ ] Escape embedded xml in response
- [ ] Ignore HTTP Proxy

Http Headers | Input Parameters | Output Parameters | Results

*Default Value will be used if Test value is empty.

ID	Test Value	Datatype	Encode
session		string	false
Accounts		collection	false
...			



Account_BulkDelete-Sync Web Service Editor

Define Service

Request/Response

Request | Response

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:partner.soap.sforce.con
  <soapenv:Header>
    <urn:SessionHeader>
      <urn:sessionId>$session</urn:sessionId>
    </urn:SessionHeader>
  </soapenv:Header>
  <soapenv:Body>
    <urn:delete>#foreach ($rec in $Accounts.records) <urn:ids>$rec.getParam("Id").value</urn:ids>#end </urn:delete>
  </soapenv:Body>
</soapenv:Envelope>
```

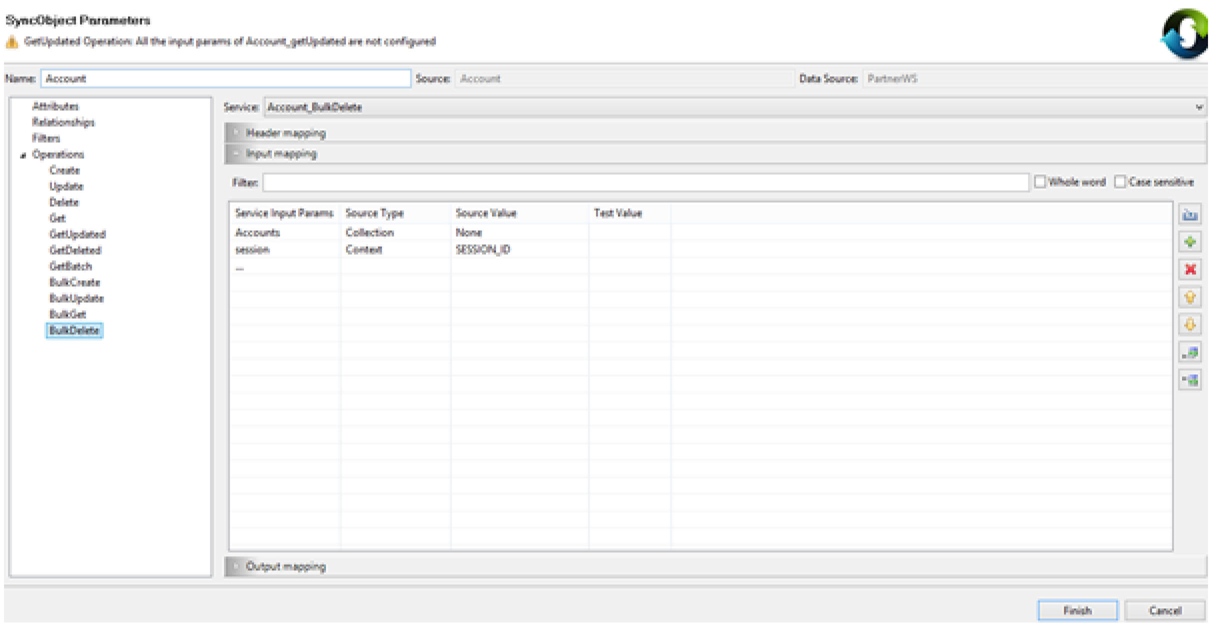
Http Headers | Input Parameters | Output Parameters | Results

To view the result of output, click on 'Results' tab.

The ID field is mand

ID	XPath	Datatype
ErrorResponse	//deleteResponse	string
errmsg	result/errors/message	string
...		

- Mapping BulkDelete operation with `Account_Bulkdelete` service for Account Sync object.



SyncObject Parameters

GetUpdated Operation: All the input params of Account_getUpdated are not configured

Name: Account Source: Account Data Source: PartnerWS

- Attributes
- Relationships
- Filters
- Operations
  - Create
  - Update
  - Delete
  - Get
  - GetUpdated
  - GetDeleted
  - GetBatch
  - BulkCreate
  - BulkUpdate
  - BulkGet
  - BulkDelete

Service: Account_BulkDelete

Header mapping

Input mapping

Filter: Whole word Case sensitive

Service Input Params	Source Type	Source Value	Test Value
Accounts	Collection	None	
session	Context	SESSION_ID	
...			

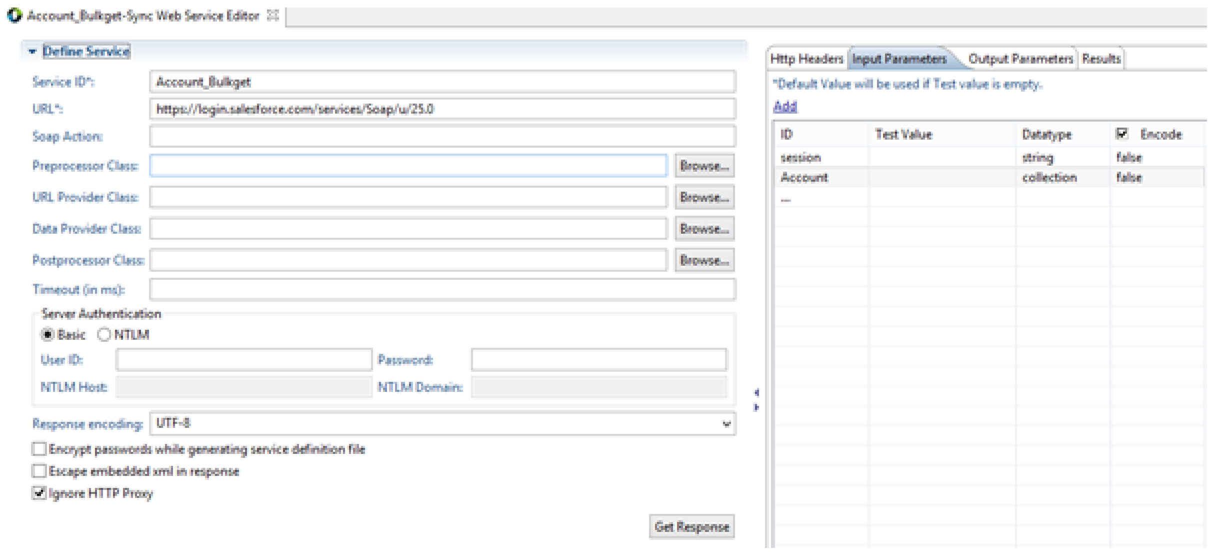
Output mapping

15.2.6 BulkGet Operation

Sync engine uses this operation to fetch the latest data from the datasource during the **BulkUpload** phase. Before invoking the BulkUpdate / BulkDelete on data source, BulkGet is invoked to check whether there are any conflicts. If BulkGet operation is not defined, then sync engine will fall back to ‘Get’ operation and invokes the service for each individual primary key.

Example:

- Service configuration (`Account_Bulkget`) for BulkGet operation of “Account” sync object.



Service ID*:	Account_Bulkget
URL*:	https://login.salesforce.com/services/Soap/u/25.0
Soap Action:	
Preprocessor Class:	
URL Provider Class:	
Data Provider Class:	
Postprocessor Class:	
Timeout (in ms):	

Server Authentication

[✓ Basic] NTLM

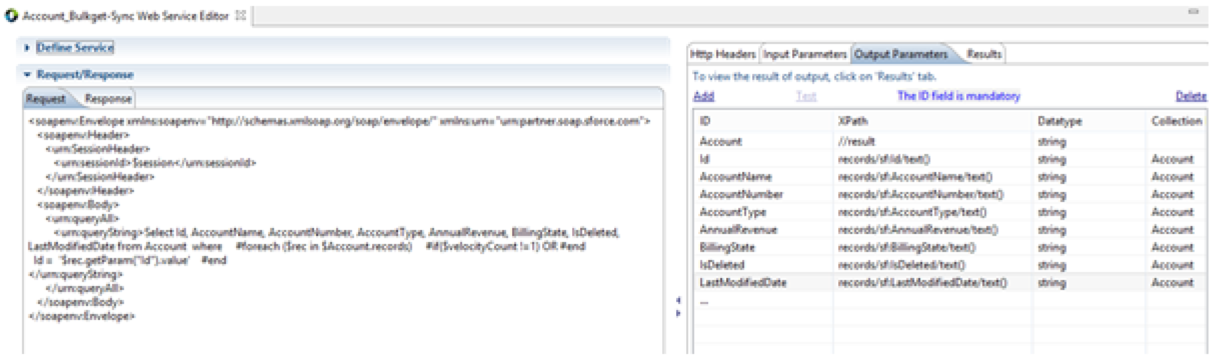
User ID:		Password:	
NTLM Host:		NTLM Domain:	

Response encoding: UTF-8

☐ Encrypt passwords while generating service definition file
☐ Escape embedded xml in response
☑ Ignore HTTP Proxy

*Default Value will be used if Test value is empty.

ID	Test Value	Datatype	☑ Encode
session		string	false
Account		collection	false
...			



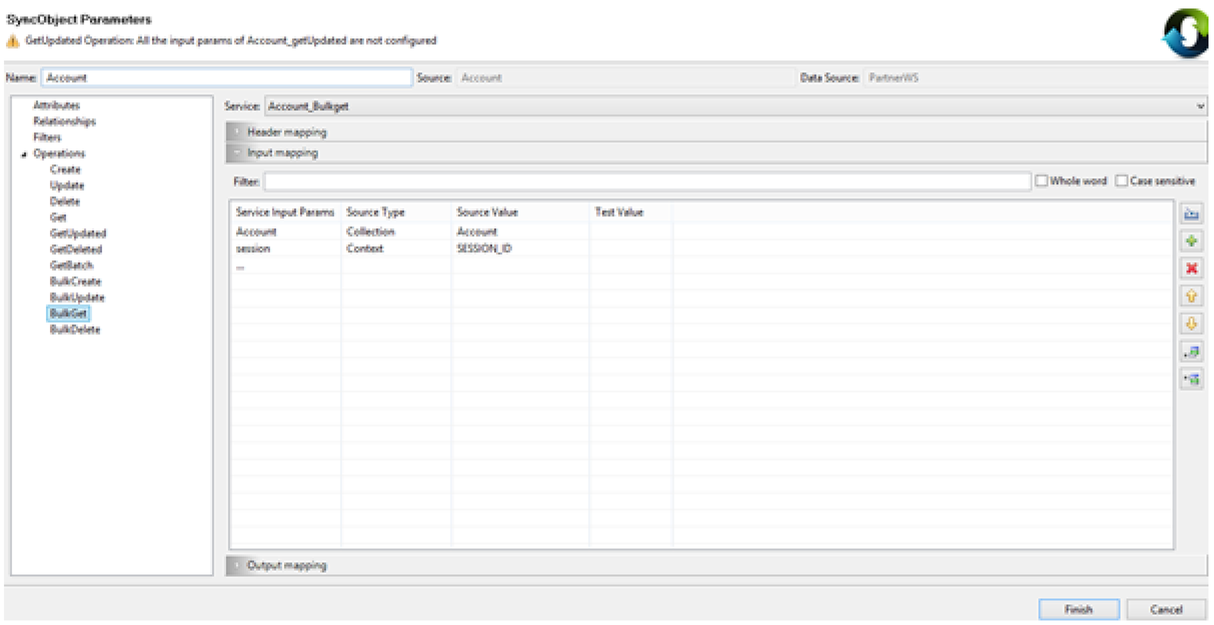
```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:partner.soap.sforce.com">
  <soapenv:Header>
    <urn:SessionHeader>
      <urn:sessionId>$session</urn:sessionId>
    </urn:SessionHeader>
  </soapenv:Header>
  <soapenv:Body>
    <urn:queryAll>
      <urn:queryString>Select Id, AccountName, AccountNumber, AccountType, AnnualRevenue, BillingState, IsDeleted,
LastModifiedDate from Account  where    #foreach ($rec in $Account.records)    #if($velocityCount != 1) OR #end
 Id = '$rec.getParam("Id").value'    #end
</urn:queryString>
    </urn:queryAll>
  </soapenv:Body>
</soapenv:Envelope>
```

To view the result of output, click on 'Results' tab.

The ID field is mandatory

ID	XPath	Datatype	Collection
Account	//result	string	
Id	records/sf:Id/text()	string	Account
AccountName	records/sf:AccountName/text()	string	Account
AccountNumber	records/sf:AccountNumber/text()	string	Account
AccountType	records/sf:AccountType/text()	string	Account
AnnualRevenue	records/sf:AnnualRevenue/text()	string	Account
BillingState	records/sf:BillingState/text()	string	Account
IsDeleted	records/sf:IsDeleted/text()	string	Account
LastModifiedDate	records/sf:LastModifiedDate/text()	string	Account
...			

- Mapping BulkGet operation with `Account_Bulkget` service for Account Sync object.



SyncObject Parameters

GetUpdated Operation: All the input params of Account_getUpdated are not configured

Name: Account Source: Account Data Source: PartnerWS

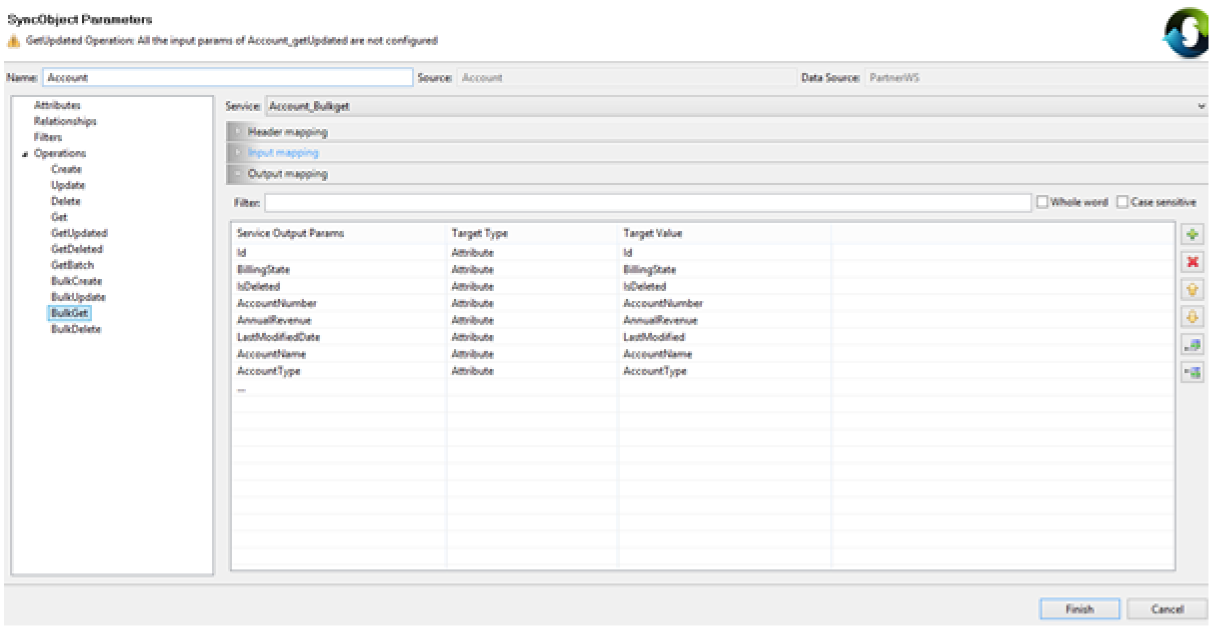
Service: Account_Bulkget

Header mapping

Input mapping

Service Input Params	Source Type	Source Value	Test Value
Account	Collection	Account	
session	Context	SESSION_ID	
...			

Output mapping



SyncObject Parameters

GetUpdated Operation: All the input params of Account_getUpdated are not configured

Name: Account Source: Account Data Source: PartnerWS

Service: Account_Bulkget

Header mapping

Input mapping

Output mapping

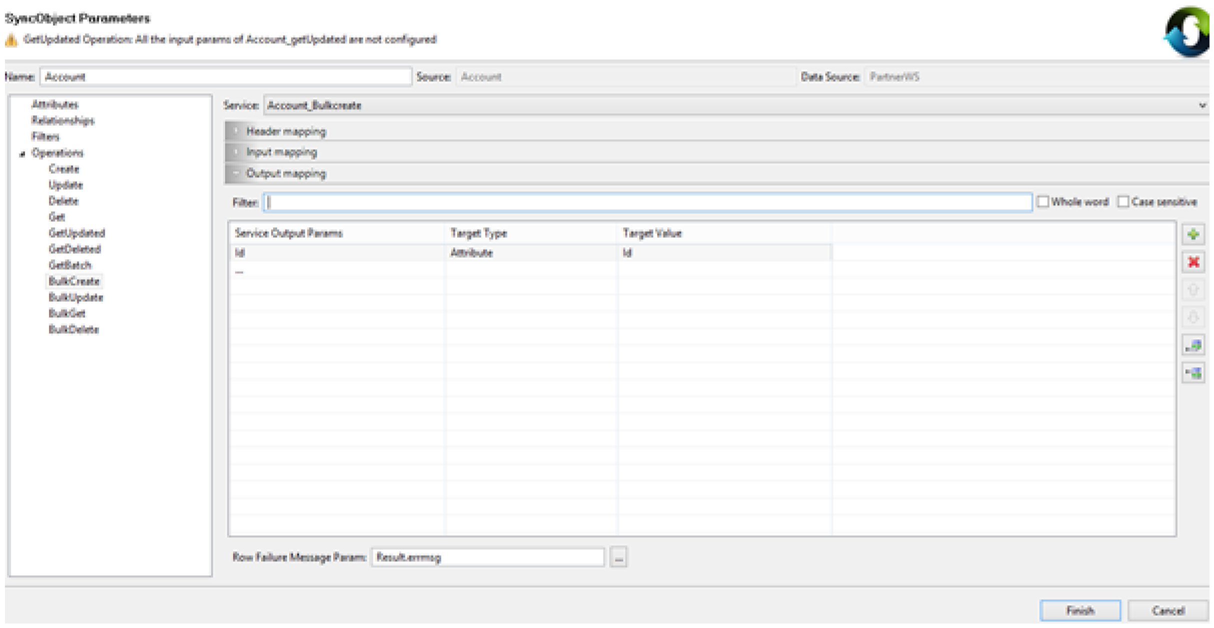
Service Output Params	Target Type	Target Value
Id	Attribute	Id
BillingState	Attribute	BillingState
IsDeleted	Attribute	IsDeleted
AccountNumber	Attribute	AccountNumber
AnnualRevenue	Attribute	AnnualRevenue
LastModifiedDate	Attribute	LastModified
AccountName	Attribute	AccountName
AccountType	Attribute	AccountType
...		

15.2.7 Use Cases for Bulk Operations

- **Parent-child rows**
    - All parent rows through one service and all child rows through one service.
    - Create a child row in existing parent row, create a new parent and child
    - Delete parent with cascade delete, delete another parent with cascade delete.
- **Self-referencing objects**
    - Person-Person, when we upload parent person & child person.

15.2.8 Error Handling During Bulk Uploads

During the bulk uploads, there can be a case where few records are failed at the back-end service logic due to business validations. To handle such failures in Kony Sync, user need to map the "Row Failure Message Param" in the output mapping in each of the bulk operation to the service output parameter which indicate the error.


SyncObject Parameters
GetUpdated Operation: All the input params of Account_getUpdated are not configured

Name: Account
Source: Account
Data Source: PartnerWS
Service: Account_Bulkcreate

Service Output Params	Target Type	Target Value
Id	Attribute	Id
...		

Row Failure Message Param: Result.errmsg

Kony Sync expects the error messages order to be same as input record order. For example, if we are uploading 4 rows and records at 2nd and 4th rows are failed, then the response should look similar to below, where "/error/message" is present only in 2nd and 4th rows and not in 1st and 3rd. Based on this mapping, Kony sync server will consider 2nd and 4th rows are failed and 1st and 3rd are successful.

```
<createResponse>
    <result>
       <id>0039000001c3VgeAAE</id>
       <success>true</success>
     </result>
     <result>
       <errors>
          <fields>LastName</fields>
          <message>Required fields are missing: [LastName]</message>
          <statusCode>REQUIRED_FIELD_MISSING</statusCode>
        </errors>
        <id xsi:nil="true"/>
        <success>false</success>
      </result>
      <result>
        <id>0039000001c3VgfAAE</id>
        <success>true</success>
      </result>
      <result>
         <errors>
               <fields>LastName</fields>
               <message>Required fields are missing: [LastName]
</message>
               <statusCode>REQUIRED_FIELD_MISSING</statusCode>
          </errors>
            <id xsi:nil="true"/>
            <success>false</success>
       </result>
</createResponse>
```

15.2.9 Context Variables

USER_ID: Sync engine sets this variable into the context. This variable contains the user id of the user who initiated the sync.

LAST_SYNC_TIMESTAMP: Sync engine sets this variable into the context. Last update time stamp is sent from the device for each sync. Server sends the updated last sync time in the response. You have to map this variable in 'getUpdated' operation to get incremental updates.

CURRENT_TIMESTAMP: The current time of the server. If 'getServerTimne' is defined in the sync configuration, then this value contains the datasource current time as returned by the 'getServerTimne' service, otherwise value contains the Sync local current time.

MORE_CHANGES_AVAILABLE: You need to map this context parameter in the 'getUpdated' operation output mapping for batch processing. If the value is evaluated to true, then 'getBatch' operation is invoked for the next batch. If this context variable is not mapped, then more changes available are considered as false.

PENDING_BATCHES: You need to map this context parameter to indicate the pending number of batches. This is optional, if not mapped then pending batches are '0'.

BATCH_SIZE: Sync engine sets this context parameter into the context for the input mapping. Operations like 'getUpdated' and 'getBatch' can map this in the input mapping.

There is no guarantee that the requested batch size is the actual batch size. Actual batch size is closer to the specified batch size.

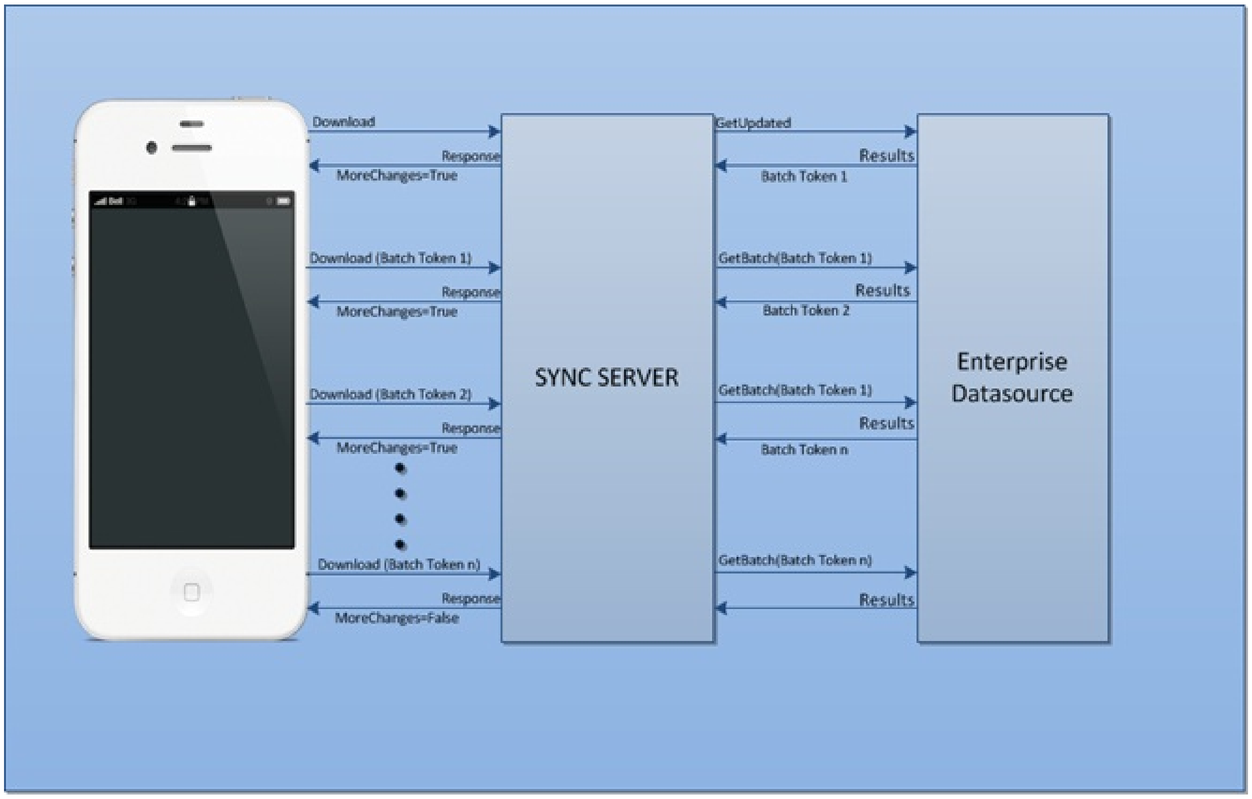
Note: While specifying the Batch size from the device, please make sure that it is a valid value for enterprise datasource. For example: Salesforce throws an error for batch size more than 2000.

SESSION_ID: You can set this context variable in the custom authentication manager class for session based batching. If SESSION_ID is set in the context, only for the first time, the user is authenticated in the batch processing, in the subsequent calls, you just verify if the SESSION_ID is valid or not. Once batch processing is over, SESSION_ID is cleared.

You can map this context variable in operation input mapping when service needs session id.

15.3 Batching in Webservices

The below diagram depicts the high level representation of the download service call with batching from the device to datasource in OTASync.



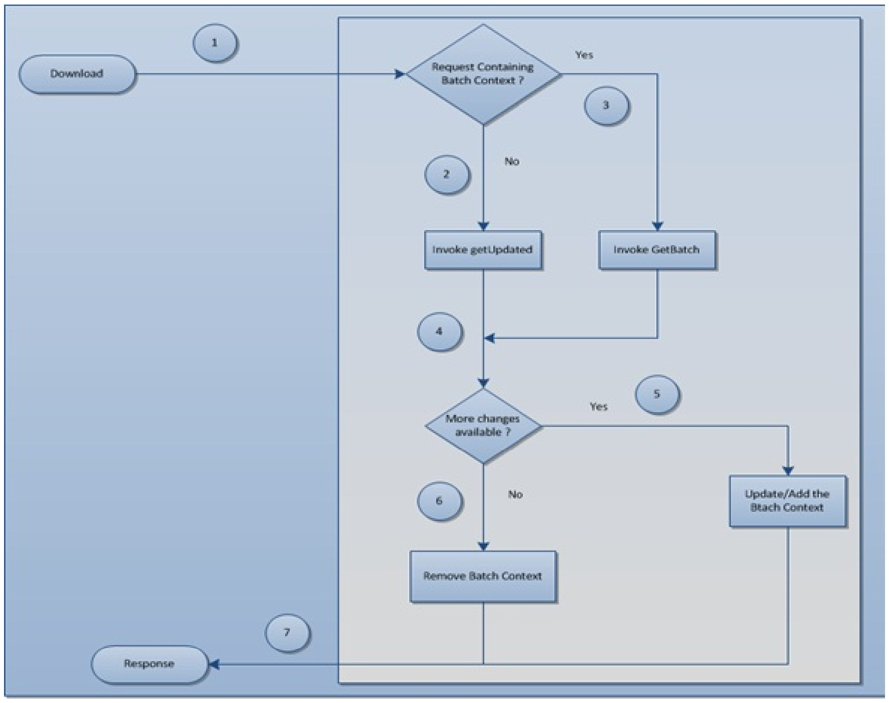
1. When user does a sync from device, “upload” and “download” service calls are invoked from device to Sync .
2. In “download” service call, Sync invokes the service mapped for the “Getupdated” operation, enterprise datasource returns the first batch data along with token that is needed to get the next batch.
3. Sync adds the batch token along with other context parameters into the response, so that Sync knows next time it has to get the next batch instead starting the download from the beginning.

4. Device keeps on invoking the “download” until “more change available” flag is *false*. Device sends the batch context received in the response from the Sync in the next request.

5. If the batch context is available in the download request, then Sync invokes the “GetBatch” operation to get the next batch.

15.3.1 Flow Diagram of Batching logic

The below diagram shows the batching logic applied when datasource supports the batching.



If download request from the device specifies the batch size, then device will receive the data in multiple batches depending on the batch size. There is no guarantee that the requested batch size will be the actual batch size. Actual batch size will be closer to the specified batch size.

For tracking the download status, sync engine will maintain the state in the batch context, which will be exchanged during request/response from the device to Sync .

15.3.1.1 Flow of Download Service

1. After receiving the download request from device, sync engine verifies whether batch context is present in the request.

2. If batch context is not available, then sync engine will invoke “GetUpdated” operation to start batching.

3. If batch context is available, sync engine will continue to fetch the next batch using batch context by invoking the “GetBatch” operation.

4. After fetching the results from GetBatch/GetUpdated, sync engine will check for the value of MORE_CHANGES_AVAILABLE context variable to know whether there are any more batches.

5. If the value of context variable is “true”, then batch context will be updated and included in the response.

6. If there are no more updates, then batch context will not be included in the response.

The above flow repeats from the device till the more changes available flag is *false*.

Note: In case of Webservice as Datasource, when errors are reported in SOAP response instead of SOAP Faults, then add the below param in the `Synconfig` file under each service output param to catch the exact error.

For example: To use salesforce webservice, add as shown below:

```
<Param Name="errmsg" Expression="//errors/message/text()"
Datatype="string"/>
```