



NEMF

Developer Guide

Version: 6.1

Disclaimer

This document contains information proprietary to Newgen Software Technologies Ltd. User may not disclose or use any proprietary information or use any part of this document without written permission from Newgen Software Technologies Ltd.

Newgen Software Technologies Ltd. makes no representations or warranties regarding any software or to the contents or use of this guide. It also specifically disclaims any express or implied warranties of merchantability, title, or fitness for any particular purpose. Even though Newgen Software Technologies Ltd. has tested the hardware and software and reviewed the documentation, it does not guarantee or imply that this document is error free or accurate regarding any particular specification. As a result, this product is sold as it is and user, the purchaser, is assuming the entire risk as to its quality and performance. Further, Newgen Software Technologies Ltd. reserves the right to revise this publication and make changes in its content without any obligation to notify any person, of such revisions or changes. Newgen Software Technologies Ltd. authorizes no Newgen agent, dealer or employee to make any modification, extension, or addition to the above statements.

Newgen Software Technologies Ltd. has attempted to supply trademark information about company names, products, and services mentioned in this document. Trademarks indicated below were derived from various sources.

Copyright © 2024 **Newgen Software Technologies Ltd.** All Rights Reserved.

No part of this publication may be reproduced and distributed without the prior permission of Newgen Software Technologies Ltd.

Newgen Software, Registered Office, New Delhi

E-44/13

Okhla Phase - II

New Delhi 110020

India

Phone: +91 1146 533 200

info@newgensoft.com

Contents

Preface	5
Revision history	5
About this guide.....	5
Intended audience	5
Related documents	5
Documentation feedback	6
Introduction	7
NEMF overview	9
NEMF server architecture.....	9
Server architecture diagram	9
Server binary distribution.....	10
NEMF client architecture	11
Client code distribution	11
Client architecture diagram	12
Framework components at high level	13
Basic entities.....	13
Framework interfaces	13
Mandatory configuration	16
Client server interaction.....	18
Interaction object	18
Interaction model.....	22
API endpoint URL format	22
Callback URL format	23
Interaction monitoring	24
Logging.....	25
Client side component	26
Writing solutions over NEMF	95
Server side	95
Defining server interface (abstractions).....	95
Writing concrete implementations	96
Pushing data to the client asynchronously	96
How the solution can choose to delay a response	97
Developing solution Plug-in	98
Implementing a SolutionInterface.....	98
Implementing and exposing factory.....	100
Implementing and exposing SolutionEventHandlers.....	103
Creating solutions.xml	103
Sample organization.xml.....	104
Sample cloud.xml	106

REST API web app	108
Web app structure	108
Folder structure	108
Web.xml	108
Third party libraries	109
Client side	112
NGOpenWebDesktopBase4.0 folder structure	112
NGOpenWebPhonegapBase4.0 folder structure	113
Client custom code overview	113
Design HTML templates	114
HTML view ['_login_view.js']	114
Preview	114
Implementing a Model ['ng-login-view-script.js']	116
Calling APIs from the Model	123
Form formats available in NEMF	124
Creating platform specific builds for mobile platforms	130

Preface

This chapter provides information about the purpose of this guide, details on the intended audience, revision history, and related documents for the NEMF Developer Guide.

Revision history

Revision date	Description
September 2024	Initial publication

About this guide

This guide provides information on the use cases of various NEMF components. It also explains how to create the application code using the NEMF framework.

Intended audience

This developer guide is intended for application developers responsible for creating and modifying the application code using the NEMF framework. The user must have a good understanding of the NEMF client and server side.

Related documents

The following documents are related to the NEMF Developer guide:

- NEMF Release Notes
- NEMF Deployment Guide
- NEMF Administration guide
- NEMF Installation and Configuration Guide

Documentation feedback

To provide feedback or any improvement suggestions on technical documentation, write an email to docs.feedback@newgensoft.com.

To help capture your feedback effectively, share the following information in your email:

- Document name
- Version
- Chapter, topic, or section
- Feedback or suggestions

Introduction

NEMF stands for Newgen Enterprise Mobility Framework. NEMF enables the rapid development of process-centric, context-aware, and device-independent mobile apps with rich imaging capabilities. Although it is written primarily to cater to the Enterprise Mobility requirements, it is a full-fledged development framework providing value across client and server layers.

Value delivered at the server end

- A highly secure, multitenant, flexible, robust, and geography-aware platform to deploy RESTful APIs.
- Allows “Writing to Abstractions”, to provide maximum shielding from variations in the Enterprise technology landscape.
- Supports serving multiple versions of the solution, to multiple customers, from a single hosting.
- Can handle multiple communication styles:
 - Synchronous
 - Asynchronous
- Can handle requests and responses of very large sizes, very efficiently.
 - Support for accepting very large requests, in multiple parts, with subsequent reconstruction at the server-end.
 - Support for sending large responses to clients, in multiple parts, in asynchronous mode.
 - Support for sending the paginated response for very large lists in synchronous mode.
 - Support for JSON transfer across the network layer. It has the clear advantage at the JavaScript-based client layer of being directly recognized as a JavaScript object without invoking the development overhead of invoking a parser. Also, it is a more compact and preferred form of data transfer format.
 - JSON parsers for non-JavaScript platforms are readily available.
- Supports server push technology to allow solutions to push changes to business data or client configuration to clients according to their roles or groups.

- A very simple development framework to write solutions with. Since “Writing to Abstractions” is at the soul, all components tend to remain in the maximum reusable state.
- Upcoming possible support for JAX-RS.
 - Most of the existing JAX-RS implementations do not allow the “Writing to Abstraction” concept which holds the key to winning over variability.
 - Also, the licensing for most of the existing JAX-RS implementations may not be permissive enough to be included in our products.

Value delivered at the client's end

- Develop once, run anywhere.
- Provides on-the-move operations, information capture, and accessibility.
- Covers all major mobile platforms as well as desktops (through HTML5, CSS3, and JS-Phonegap).
- Views can be arranged in a parent-child hierarchy, thus enabling the easy organization of application views.
- A very agile JavaScript-based MVC model that supports, besides model-based view generation, the asynchronous cascaded event processing across the parent-child hierarchy.
- Support for i18n/l10n.
- Out-of-the-box support for Responsive UI Frameworks such as Bootstrap.
- Image Processing capabilities at client's end.

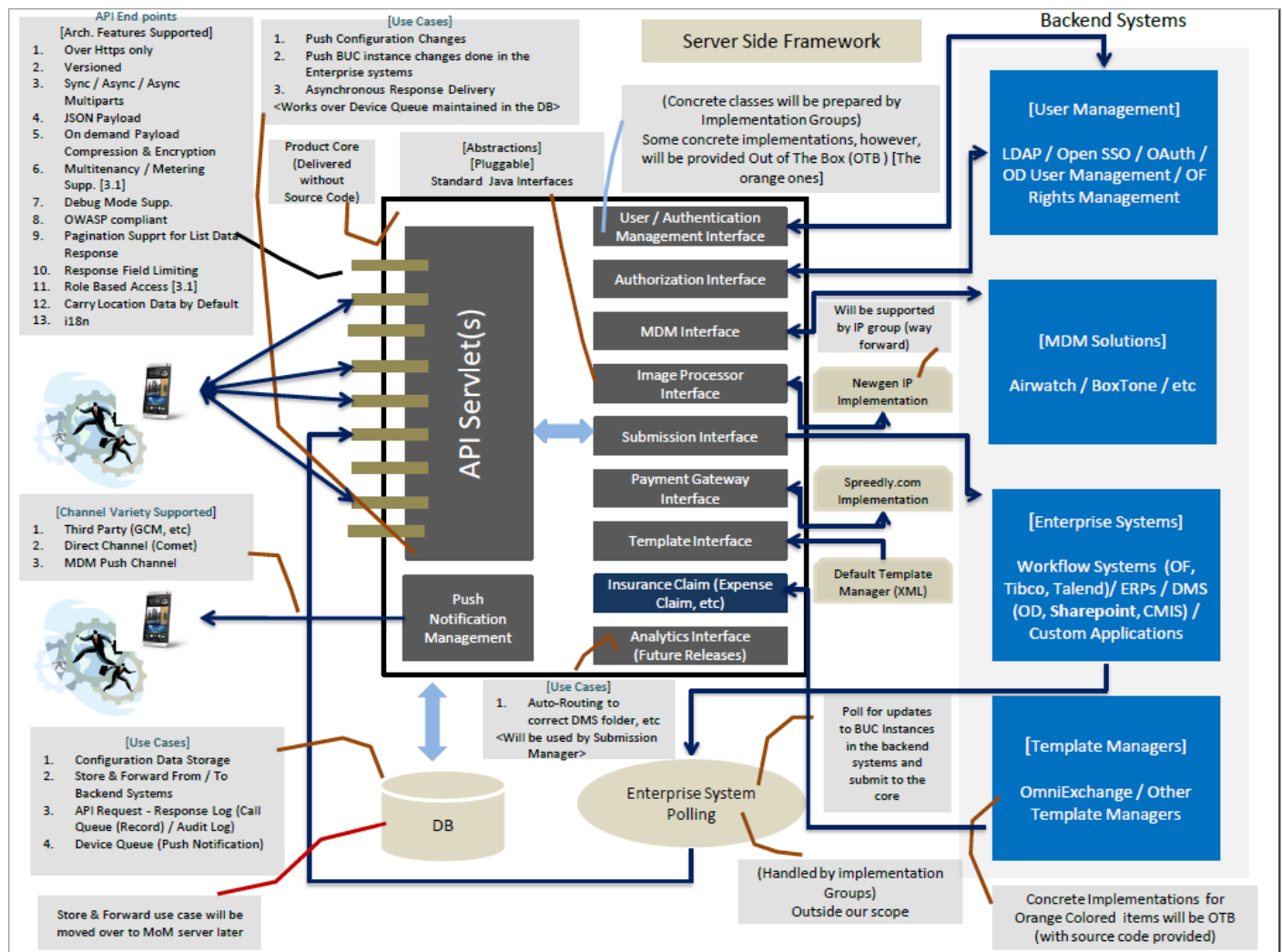
NEMF overview

NEMF overview describes the NEMF server architecture and client architecture.

NEMF server architecture

The following sections explain server architecture and server binary distribution.

Server architecture diagram



Server binary distribution

The following table describes the server binary distribution:

Artifact name	Artifact purpose	Artifact dependencies
NGApiCommons.4.0.jar (Closed Source)	This JAR is needed in Development as well as at Runtime. It contains all the needed classes that a solution writer wants to deal with at the time of development.	Not applicable
NGApiEngine.4.0.jar (Closed Source)	This JAR carries the implementation for API Engine (Product Core) that is the heart of the framework supporting all the required architectural contracts. This JAR is needed only at Runtime. It depends on NGAPICommons for the shared classes.	NGApiCommons.4.0.jar
NGApiWeb.4.0.jar (Closed Source)	This JAR contains the RestAPI servlet implementation that internally invokes API Engine for processing requests. JAR is needed only at the Runtime. It depends on NGApiInternal.	NGApiCommons.4.0.jar NGApiInternal.4.0.jar

Artifact name	Artifact purpose	Artifact dependencies
NGApiEssentials.4.0.jar (Open Source)	<p>This JAR contains concrete implementations for necessary abstractions that we feel are essential for all applications, however, the concrete implementations provided along with the Abstractions are by no means Essential. Solution writers can write their own Concrete implementations for these abstractions. They can even write their own Abstractions (or extend existing ones) and their concrete implementations.</p> <p>The provided source code can be used for learning solution writing.</p>	NGApiCommons.4.0.jar

NEMF client architecture

The following section explains client code distribution and architecture diagram.

Client code distribution

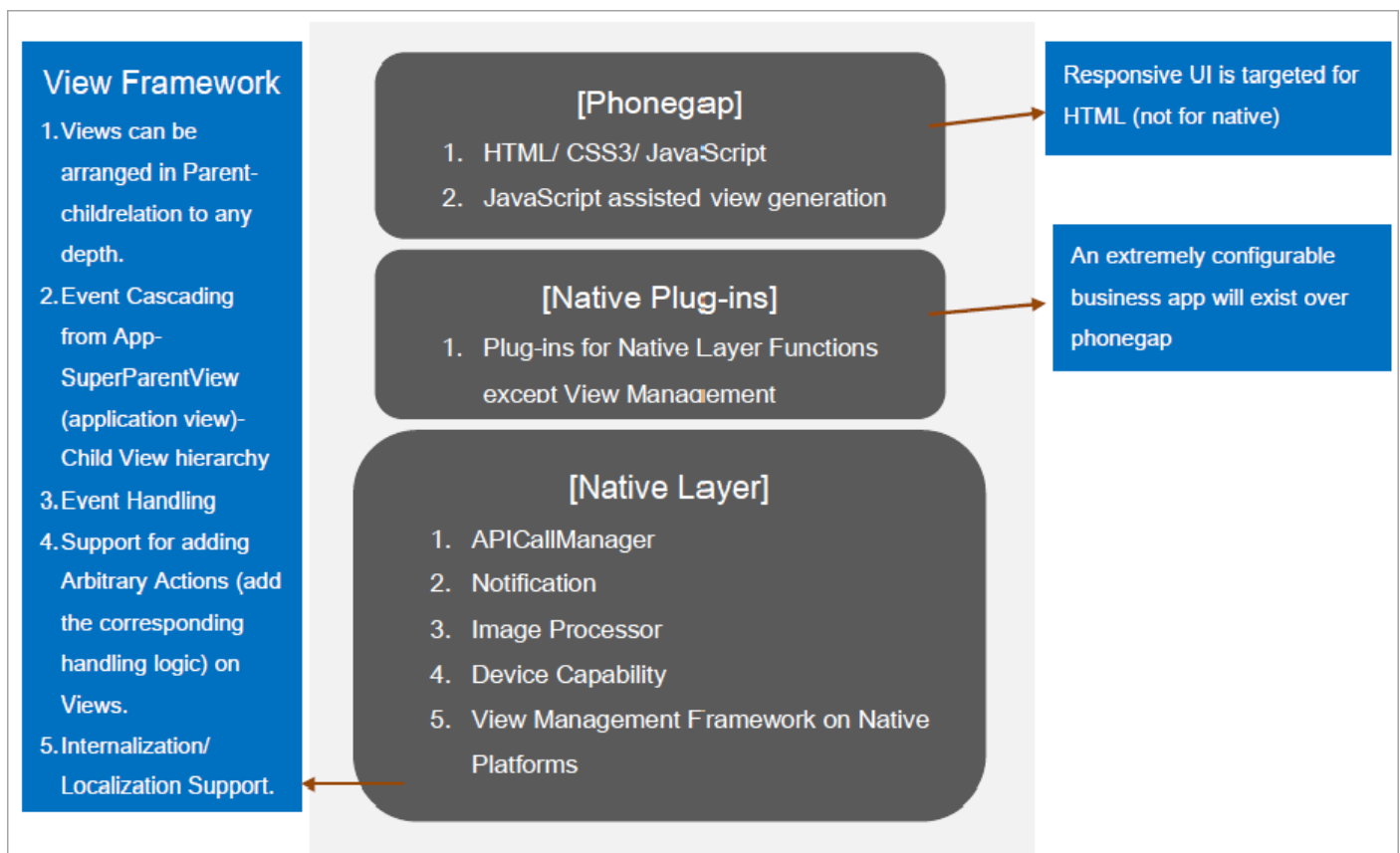
The following table describes the client code distribution:

Artifact name	Artifact purpose	Artifact dependencies
NGOpenWebDesktopBase.4.0.Zip (Open Source)	<p>This zip contains all the necessary components and scripts that a solution provider needs to write an application for Desktop-based clients.</p> <p>The artifact supports HTML5, CSS3, and JS-based development for the desktops.</p>	Not applicable

Artifact name	Artifact purpose	Artifact dependencies
NGOpenWebPhonegapBase.4.0.Zip	This supports HTML5, CSS3, and JS-based development for mobile devices (all major platforms). The solution written over 'DesktopBase' above can be easily ported over 'PhonegapBase' (and vice-versa) as the internal folder structures and the way a client-side app needs to be written is more or less the same.	Not applicable

There are two base projects that can be used to start the client-side development work. One is suitable for mobiles and the other for desktops. More about it is discussed later in the guide.

Client architecture diagram



Framework components at high level

The following sections explain the basic entities, framework interfaces, mandatory configuration, and more.

Basic entities


The framework provides a few basic entities. The solution writers are required to reuse these (or subclass these if they need more attributes) so that the core is more tightly integrated with the solution at run time and can support cases like multi-tenancy and push events (according to user roles).

All these entities are available in the following package:

com.newgen.mcap.core.external.basic.entities.concrete

Framework interfaces

Following is the list of basic Java interfaces. These interfaces are available in the *com.newgen.mcap.core.external.basic.interfaces* package.

Interface name	Purpose
Abstracted Functionality	<p>All abstractions that a solution writer defines the need to be extended for the Abstracted Functionality interface.</p> <p>The interface enables the score to inject APIContext objects within the concrete implementation at runtime. The context object enables solution writers to:</p> <ul style="list-style-type: none"> • Get access to APIScopes, which, in turn, enables solution writers to access or save attributes at the application, user session, or per-request level. But, since REST APIs are supposed to be stateless (more or less), use 'application' and 'session' scopes very sparingly. • APIContext also allows solution writers to call other APIs, hosted on the core, in a manner that bypasses HTTP. <p>The interface also allows solution writers to expose the solution Version, Abstraction Name, and Identifier Name to the core. The concrete implementation will, thus, get registered within the core with the following <i>identity</i> ("solutionversion/abstractionname/identifier"), and individual methods get registered as ("solutionversion/abstractionname/identifier/<methodname>").</p> <div data-bbox="630 1094 1498 1173">  JAX--RS annotations must be supported to map these methods in a way that is closer to pure REST flavor. </div>
Compressible	<p>This interface must be implemented for those solution domain objects that are required to be compressed and decompressed on their flight between client and server.</p> <p>The core will call the interface methods to get and set compressed and decompressed object images.</p>

Interface name	Purpose
Configurable	<p>This interface must be implemented for those solution domain objects that represent solution configuration items. The core will use the interface methods to satisfy the following use cases:</p> <ul style="list-style-type: none"> • Checks if the underlying configuration has changed recently. • If the configuration has changed, then the new configuration is queried for (through this interface) and the core's internal record of the configuration is updated. • Moreover, if the configuration is declared by the interface to be savable, then the new configuration gets saved in the RDBMS as well. <p>Also, if the configuration is required to be pushed to the device (as specified by an interface method itself) then the core will push the configuration to the client as well.</p>
ContextBuilder	<p>It provides an opportunity for the solution writer to register its context with the core. A solution context typically contains the following:</p> <ul style="list-style-type: none"> • Solution configurable. • Solution abstractions and their concrete implementations. • Solution event listeners. • Organization for which the solution is written. <p>An object implementing ContextBuilder and known to the core will be called upon by the core to inform the solution context to the core so that it can be internalized or registered.</p>
Encryptable	<p>This interface must be implemented for those solution domain objects that are required to be encrypted and decrypted on their flight between client and server. The core will call the interface methods to get and set encrypted and decrypted object image</p>
GeoFenceable	<p>This interface must be implemented for the solution domain objects that the solution providers want to get processed only when they are produced within a defined GeoFence. The core will throw exceptions if a GeoFenceable object is found breaching the fence and further processing will be stopped.</p> <p>GeoFence can be declared at the Organization account level.</p>

Interface name	Purpose
JSONable	<p>This interface must be implemented for those solution domain objects that need to transfer their state across the network in the JSON format.</p> <p>! Soon there will be support for XML-based transfer as well.</p>
Saveable	<p>This interface must be implemented for those solution domain objects that need to retain their states in data stores. The core will make sure that all such Objects are persisted or updated in the data store when received on the request.</p>
Searchable	<p>This provides methods for object graph searching based on string expressions (like XPath). It must be noted this is not for search into the data store.</p>
Streamable	<p>This interface needs to be implemented for those solution domain objects that need to stream their state across the network as binary. For example, images. The binary stream will be packed within JSON.</p>

Mandatory configuration

Solution writers need to provide the following configurable(s) in their XML forms (current support) for the core to function. The system must be designed in such a way that the configuration is required to be kept only at the server end; the client side is configured on the fly by the server using push notifications carrying necessary client configuration (which might be role-based).

Configurable name	Purpose
Cloud	<p>Provided as <i>Cloud.xml</i></p> <p>It contains the configuration that carries the Cloud provider-specific configuration as well as those parameters that are common across all solutions across all organization accounts. Since the core is multi-tenant and is supposed to be running on the cloud, the configurable is named <i>Cloud.xml</i>. However, the same configuration will apply for in-premise installations as well with a single or multiple organization accounts.</p> <p><i>Cloud.xml</i> needs to be provided by solution writers only for the in-premise mode.</p>

Configurable name	Purpose
Organization	<p>Provided as <i>Organization.xml</i> (or through Organization account creation screens planned for the future)</p> <p>This contains an organization account-specific configuration that will be common for all solutions running on the core for this organization.</p>
Solution	<p>Provided as <i>Solution.xml</i></p> <p>This contains the references to Solution Interface implementations, exposed by the solution writers. The main purpose is to be queried by the core to get the solution context, to be able to register the solution within itself and expose its corresponding REST API endpoints.</p>

Client server interaction

The server framework supports interactivity with clients of various natures in manners described in the below subsections.

Interaction object

The interaction model is available in the `com.newgen.mcap.core.external.apiengine.interaction.entities.concrete` package.

Object name	Purpose
APIRequest	<p>APIRequest models a typical request entity that a client can make to the NEMF server. A JSON corresponding to this entity is sent through Https post request for making the API call.</p> <p>It has the following attributes that a solution needs to deal with.</p> <ul style="list-style-type: none"> • <code>apiEndPointDetails</code> — It specifies the API endpoint method where this request needs to hit. It has the following format ("organizationId solutionVersion abstractionName identifier methodName"). • <code>apiResponse</code> — It is the corresponding response entity for the request. • <code>mode</code> — Request mode. A mode can be Sync, Async, or Async Multiparts. Async and Async MultiPart requests will receive a correlationID immediately. • <code>packedResources</code> — Resource objects that are carried by this request. These serve as the parameters to the API endpoint. • <code>responseFieldLimiting</code> — It is required to limit the object graph packed within the response. It contains the comma-separated list of object graph attributes that need to be packed within the response. Attributes that are not on this list, are not packed in the response. • <code>hitTimeStamp</code> — It represents the time stamp when the request was received.

Object name	Purpose
	<ul style="list-style-type: none"> • debugLevel — If logs are needed, then the debug level logs that should be packed can be specified on the URL as well. • requestState — It represents the state of a specific request. • device — It models the device that generated the request. It can be null. • user — The user account that generated the request. The user object must contain a validated authentication token. • group — Group for which request was generated. • role — Role for which request was generated. • organization — Organization for which request was generated. • geoLocation — The geo-location where the request was generated. • clientGivenID — It specifies the request ID at the client-end. • clientRequestType — It specifies the request type at the client-end. • logsNeededInResponse — It specifies whether the server-side logs need to be packed in the response. • compressOutput — It specifies whether to compress the response before sending it back to the client. • encryptOutput — It specifies whether to encrypt the response before sending back to the client. • compressInput — It specifies whether to compress the request before sending it to the server. • encryptInput — It specifies whether to encrypt the request before sending it to the server. • pagination (Boolean) — It is used when a giant list response is expected; Pagination can be turned on via this parameter. • recordsPerPage — It is used to specify the number of list items to pack per page. • requestedPageNumber — Which page number to fetch. The default is page number 1. • listSession — A unique ID against which the entire list is maintained on the server-side. Paginated chunk is created using this. • partNumber — It specifies the part number of the request when sent as Async multipart request.

Object name	Purpose
	<ul style="list-style-type: none"> • <code>totalNumberOfParts</code> — It specifies the total number of parts of the Async multipart request. • <code>totalBytesInTheParts</code> — It specifies the total bytes in a particular part of the request. • <code>byteArrayBeginIndex</code> — It specifies the 'begin index array' of the chunk sent in the particular part of a request sent as Async multipart request. • <code>byteArrayEndIndex</code> — It specifies the 'end index array' of the chunk sent in the particular part of a request sent as Async multipart request. • <code>totalNumberOfbytes</code> — It specifies the total number of bytes of the Async multipart request. • <code>correlationId</code> — It is used for synchronizing the asynchronous request-response at the client and server-end. • <code>solutionVersion</code> — It is used for defining the solution version. • <code>underProcess</code> — It is used to process the asynchronous request in the cluster mode. If the request has been picked by the job instance of one node, the value of this attribute must be "true".
APIResponse	<p>APIResponse models a typical response entity expected from the NEMF server. A JSON corresponding to this object is received by the client over HTTP or via one of the Async (Push) channels. It has the following attributes of relevance:</p> <ul style="list-style-type: none"> • <code>correspondingRequest</code> — The request that generated this response. It is not necessary that a response has a corresponding request. A solution can generate a response to notify of a server-side event as well. • <code>mode</code> — It specifies whether the response is received synchronously or asynchronously. • <code>packedResources</code> — The resource objects returned by the API end point. • <code>logs</code> — The logs generated at the server (if requested via <code>request.logsNeededInResponse</code>). • <code>responseTimeStamp</code> — It represents the time stamp when the response of request was sent. • <code>createdTimeStamp</code> — It represents the time stamp when the response was created on the server. • <code>responseState</code> — It represents the state of a specific response.

Object name	Purpose
	<ul style="list-style-type: none"> • targetDevice — The device to which response is targeted, required for Async responses • targetUser — If the target device is not set, the async response will be sent to all the devices that are associated with the target user. • targetGroup — If the target user is not set, async response will be sent to all the devices of all the users for a particular group. • targetRole — If the Target Group is not set, Async response will be sent to all the devices of all the users for a particular group. • organization — If all the target entities are not defined, the Async response will be sent to all the users on their associated device. It is mandatory for all Async responses. • clientGivenId — It is required for the client-end response ID. • serverRequestType — It specifies request type at server-end. • compressed — It specifies whether the response is compressed or not. • encrypted — It specifies whether the response is encrypted or not. • geoFenceBreachDetected — It is set to true if the request is outside the GeoFence and the request is not saved. • currentPage — If a paginated response is expected in the request, then this specifies the page number contained in the response. • listSessionHandler — It carries the listSession identifier against which the entire list is maintained at the server. • partNumber — It specifies the part number of a particular part if the bulk response is broken into multiple parts. • totalNumberOfParts — It specifies the total number of parts of the broken response. • totalBytesInThisPart — It specifies the total bytes in a particular part of the response when sent as a broken response. • byteArrayBeginIndex — It specifies the 'begin index array' of the chunk sent in the particular part of the response sent as a broken response.

Object name	Purpose
	<ul style="list-style-type: none"> • <code>byteArrayEndIndex</code> — It specifies the 'end index array' of the chunk sent in the particular part of the response sent as a broken response. • <code>totalNumberOfbytes</code> — It specifies the total number of bytes of the Async multipart request. • <code>correlationId</code> — It is used for synchronizing the asynchronous request-response at the client and server-end. • <code>solutionVersion</code> — It is used for defining the solution version. • <code>customMessage</code> — It is used to send a customized notification to the desired devices.

Interaction model

The interaction model explains the API endpoint and callback URL format.

API endpoint URL format

Interaction objects defined in the previous sections are sent and received through the API URL in the following format:

`https://<serverContext>/<debugLevel(Optional)>/api`



The pure REST flavor where domain entity model resources are exposed over URLs for CRUD operations is not yet supported. However, it is planned to do it with the JAX-RS compliant release.

Interaction with Rest API endpoint can take the following forms (managed internally within the *APICallManager* component):

- `APIRequest` is sent in synchronous mode and a corresponding synchronous `APIResponse` is received on the return path:
 - If while executing #1, `APIRequest` is found to be exceeding a configured multipart trigger value, then the request is broken (at client within `APICallManager`) into multiple smaller parts that are sent as asynchronous-multipart `APIRequests` one after the other. These parts get assembled back

at the server when all parts are received, and then the request proceeds as an async request (#2 – a correlation ID is received in this case).

- If while executing #1, `APIResponse`, on the return path, is found to be exceeding the trigger value, then it gets broken on the server into multiple smaller parts and is routed via async path (push channels) as multiple async-multipart responses. These parts get assembled at the client within `APICallManager` and delivered to the client layer as a single big response.
- `APIRequest` is sent in asynchronous mode. The request is received at the server and registered in the data store for processing later. A correlation ID, however, is generated and returned immediately (packed within a synchronous `APIResponse`). Later, one of the server-side events will process the async request and return the response asynchronously over push channels (with the same correlation ID).
- If while executing #2, `APIRequest` is found to be exceeding a configured multipart trigger value, then the request is broken (at client within `APICallManager`) into multiple smaller parts that are sent as asynchronous-multipart `APIRequests` one after the other. These parts get assembled back at the server when all parts are received and then the request proceeds as before (#2).
- If while executing #2, `APIResponse`, on the return path, is found to be exceeding the trigger value, then it gets broken on the server into multiple smaller parts and is routed via async path (push channels) as multiple async-multipart responses. These parts get assembled at the client within `APICallManager` and delivered to the client layer as a single big response (with a correlation ID).

Request-response pairs are maintained at the Client, as well as at the Server for the organization-defined configurable retention period (*Organization.xml*). This helps with production-level debugging. For further information, refer to the [Interaction Monitoring](#) section.

Callback URL format

Callback URLs are defined in the following format (*eventHandlerIdentifier* points to a *SolutionEventHandler* object exposed by the Solution Interface).

`https://<serverContext>/<debugLevel(Optional)>/api/organizationId/solutionVersion/eventHandlerIdentifier`

Interaction with the Callback URL is needed in the case API solution to be connected with other third parties such as Payment Gateways or Social Networks. These third parties need to pass the data to NEMF API solutions on one of these Callback URLs configured for the solution. A callback is handled by a `SolutionEventHandler` object implemented by the solution and a solution may define as many call-back handlers as are needed.

Interaction monitoring

JMX-based real-time monitoring is supported for the following monitoring and probing parameters:

- **Monitoring**

- Checks how many organization accounts are operational currently.
- Checks the services currently operational for a given organization.
- Checks how many requests are coming per second for a given organization (in real-time).
- Checks how many requests are coming per second across all the organizations (in real-time).
- Checks how many synchronous responses are sent back per second for a given organization (in real-time).
- Checks how many synchronous responses are sent back per second across all organizations (in real-time).
- Checks how many asynchronous responses are sent back per second for a given organization (includes single as well as multi-part responses, in real-time).
- Checks how many asynchronous responses are sent back per second across all organizations (includes single as well as multi-part responses in real-time).
- Average time (milliseconds) between a sync request and a sync response for a given organization (averages over the last X request-response pairs, where “X” is configurable).
- Average time (milliseconds) between sync request and its async response for a given organization (in case the response is too huge). This includes the time it takes for sending all the parts (averages over the last X request-response pairs, where “X” is configurable).

- Average time (milliseconds) between an async request and its async response for a given organization. In the case the response is too huge then this includes the time it takes for sending all the parts (averages over the last X request-response pairs, where “X” is configurable).
- Average time (milliseconds) between sync request and a sync response across all organizations (averages over last X request-response pairs, where “X” is configurable).
- Average time (milliseconds) between sync request and its async response across all organizations. In case the response is too huge, then this includes the time it takes for sending all the parts (averages over the last X request-response pairs, where “X” is configurable).
- Average time (milliseconds) between async request and its async response across all organizations. In the case the response is too huge, then this includes the time it takes for sending all the parts (averages over the last X request-response pairs, where “X” is configurable).
- **Probing (this helps with debugging production cases)**
 - Given a resource Name, retrieve all resource IDs that match the name.
 - Given a response ID, retrieve the response along with its full log and exception traces.
 - Given a correlation ID, retrieve all matching responses from connected clients as well as the server.

Logging

NEMF provides the logging functionality by exposing a common class and a defined method. The description and methods of the logging class are as follows:

- Class — LogMe
This class is used for providing the support of printing all kinds of logs.
- Method — logMe(int level, Throwable throwable)
 - int level — This represents the level of the log.
 - Throwable throwable — log string.

Sample Example

```
LogMe.logMe(LogMe.LOG_LEVEL_DEBUG, "callApiAsynchronousMultipartsApiResponse>>" +
toReturn.toJson());
```

Client side component

Client-side features or components are as follows:

- The client framework is written purely in JavaScript.
- NGView component:
 - Client views can be arranged hierarchically.
 - Async events are routed through the hierarchy from root to leaf so that views can update their states.
 - Supports MVC out of the box.

Method name	Description and Parameter	Returns
NGView.loadView(callback)	This method loads a view. Parameter: callback — callback method.	Void
NGView.showView (viewNameToBeShown, viewData)	This method renders a view. Parameters: <ul style="list-style-type: none"> • viewNameToBeShown — The name of the view to be shown. • viewData — The data to be shown on the view. 	Void
NGView.hideView()	This method hides a view.	Void
NGView.isHidden()	This method checks if the view is hidden or not.	<ul style="list-style-type: none"> • True – If the view is hidden. • False – if the view is visible.
NGView.addChildView (childView)	This method adds a child view to the parent view. Parameter: childView — The child view is to be added.	Void

Method name	Description and Parameter	Returns
NGView.removeChildView (childViewName)	This method removes a particular child view from the parent view. Parameter: childViewName — The name of the child view to be deleted.	Void
NGView.removeAllChildren()	This method removes all the child views from the parent view.	Void
NGView.searchViewFromRoot (viewName)	This method searches for a view starting from the root view. Parameter: viewName — The name of the view to be searched.	Searched view

- APICallManager component

Allow clients to interact with the REST API endpoints. It shields the complexity of REST interaction (as explained in previous sections) by exposing a very simple interface.

Method name	Description and Parameter	Returns
NGAPICallManager.callApi(request, callback, isBackgroundCall)	This method sends requests to the server and receives responses from it. Parameters: <ul style="list-style-type: none"> • request — This is the API Request object. • callback — optional callback method • isBackgroundCall — Boolean value to determine if the call is to be sent in the background. 	API Response object

- Interfaces

- **Storage** — In this interface, the framework provides the capability of storing user data in types of storages:
 - JSTORAGE — Storing the data into the file system.
 - FILESYSTEM — Storing the data into the file system.
 - SQLITE — Storing the data into the SQLite mobile database.
 - IndexedDB — Storing the data in an Indexed DB on the browser.
- **Submission Manager** — In this interface, the framework provides the capability of submitting the Business Use Case (BUC) to the defined targets. The framework provides the support of the below targets:
 - SERVER-OMNIFLOW — Submit the BUC Instance to OMNIFLOW at the server-side.
 - SERVER-OMNIDOCs — Submit the BUC Instance to OMNIDOCs at the server-side.
 - CLIENT-WIFIPRINTER — Submit the BUC Instance to the WiFi printer at the client-side.
 - CLIENT-EMAIL — Submit the BUC Instance to a specified email on the client-side.
 - SERVER-OMNIFLOW-APPROVAL — For handling getInstances and submit calls for Approval App.
- **Image Processing** — In this interface, the framework provides the capability of performing the image processing operation on the attachment, i.e., the captured image. Below are the supported image processing types provided by the framework:
 - NEWGEN_CLIENT — Performing the desired image processing operation at the client-side by using Newgen Image Processing capability.
 - NEWGEN_SERVER — Performing the desired image processing operation at the server-side by using Newgen image processing capability.
 - JAVACRIPT_IP_CLIENT — Performing the desired image processing operation at the client-side by using Newgen image processing capability in core JAVASCRIPT.
 - PARASCRIPT_CLIENT — Performing the desired image processing operation at the client-side by using Parascript image processing capability.
 - LEADTOOLS_CLIENT — Performing the desired image processing operation at the client-side by using LeadTool image processing capability.

- **Identity Provider** — This file contains an interface for authenticating with third-party APIs. Below are the supported third-party APIs:
 - Facebook — Authenticating the user with Facebook.
 - OmniDocs — Authenticating the user with OmniDocs.
 - Twitter — Authenticating the user with Twitter.
- **DB operations** — This file contains an interface for database operations.
 - taffyDB — This file contains an implementation of database operations for TaffyDB.
- **Social Media Provider** — This file contains an interface for different social media sharing implementations. Below are the supported social media:
 - Facebook — Media sharing with Facebook.
 - Twitter — Media sharing with Twitter.
- **Template Manager** — This file contains an interface for handling BUC Template operations.
 - Core — This file contains an implementation of the interface for handling BUC templates and their related operations.
- **BUC Async Utils** — This file contains an interface for handling asynchronous updates.
- **Core** — This file contains an implementation of the asyncUtils interface for handling asynchronous updates.

APIs name	Description and Parameter	Returns
NGBUCAsyncUtils. updateBUCInstance (bucInstance,trayArray, callback)	This method will update the business use case instance in the defined tray. Parameters: <ul style="list-style-type: none"> • bucInstance — An object of type NGBusinessUseCase • trayArray — array of trays • callback — callback method to return the result. 	Void
NGBUCAsyncUtils. updateWhitelabelingInfo (whiteLabelingConfi)	This method updates a new UI of the application. Parameter: whiteLabelingConfig — Boolean value for processing whiteLabelingConfiguration.	Void

APIs name	Description and Parameter	Returns
NGBUCAsyncUtils. updateBUCTemplate (bucTemplateconfiguration)	This method updates the application if any change is made in BucTemplate at the server side. Parameter: bucTemplateconfiguration — XML which contains new BucTemplate on the server-side.	Void

- **BUC Config Utils** — This file contains an interface for configuration-based utilities of the application.
- **Core** — This file contains an implementation of the config-utils interface.

APIs name	Description and Parameter	Returns
NGBucConfigUtils. setRestApiEndPoint (restAPIEndPoint)	Method will store RestAPI URL in the local cache. Parameter: restAPIEndPoint — Exact path of rest API in string format	Void
NGBucConfigUtils. setSortKeysForBuc (businessUseCase Template, keys)	Method will set the sort order in the businessUseCaseTemplate stored locally. Parameters: <ul style="list-style-type: none"> • businessUseCaseTemplate — An object of type NGBusinessUseCaseTemplate • keys — indexing field to sort the BUC 	Void
NGBucConfigUtils. getRestApiEndPoint	Retrieve rest API endpoints stored in local storage.	Returns rest API endpoints.
NGBucConfigUtils. setcertificate PinningInto (ngCertificate)	This will store the expected certificate info in the local store in an encrypted form. Parameter: ngCertificate: the object of NGCertificate type	Void

APIs name	Description and Parameter	Returns
NGBucConfigUtils. getGeofence Configuration FromServerResponse Handler (response)	Handles JSON NGApi response received corresponding to getGeofenceConfigurationFromServer. Parameter: response — Object of APIResponse type	Void
NGBucConfigUtils. getEncryptionKey FromServer	Deprecated Request server for the configured encryption key from the server side.	Void
NGBucConfigUtils. getEncryptionKeyFrom ServerResponseHandler (response)	Deprecated Handles JSON NGApi response received corresponding to getEncryptionKeyFromServer Parameter: response — Object of APIResponse type.	Void
NGBucConfigUtils. getGeoFenceConfiguration FromServer	This method fetches geo-fence configuration from the server side.	Void
NGBucConfigUtils. setUserPreference (ngUserPreference)	This method sets the user preferences. Parameter: ngUserPreference — an object of type NGUserPreference. UserPreferences will be stored locally, it will be saved for the logged-in user.	Void
NGBucConfigUtils. configureIndexField FormFieldMap (bucInstance,indexing Fields)	This method returns the value corresponding to the indexing fields received from the server. Parameters: <ul style="list-style-type: none"> • bucInstance — Business Use Case • indexingFields — Array fields whose value is to be mapped. 	Returns the mapped values or null incase the there is no value in <i>indexingFields</i> .
NGBucConfigUtils. getUserPreference	This method returns the user preferences stored locally.	Returns the user preferences stored locally.

APIs name	Description and Parameter	Returns
NGBucConfigUtils. getSortKeysForBuc	This method returns the sort keys for BUC.	Returns the sort keys for BUC.
NGBucConfigUtils. getCertificatePinningInfo	This method retrieves the certificate info from the local store.	Returns the certificate info retrieved from the local store.
NGBucConfigUtils. getConfiguration FromServer	This method gets the system configurations like retention frequency, retention trays, idle timer limit, data encryption algorithm, online mode and share mode from the server.	Returns true or false based on whether the configuration was received successfully or not.
NGBucConfigUtils. getConfigurationFrom ServerResponseHandler (response)	Handles JSON NGAPIResponse received corresponding to getConfigurationFromServer. Parameter: response — Object of APIResponse type.	Returns true or false based on whether the configuration was received successfully or not.
NGBucConfigUtils. getDeviceKey	This method gets the device key from the server if the key provider used is 'Configuration' and sets it in <i>jStorage</i> .	Returns true or false based on whether the device key was successfully received or not.

APIs name	Description and Parameter	Returns
NGBucConfigUtils. getActiveVersion(callback)	This method gets the current active version of the IPA or APK configured on the server. It is then matched with the current app's version and an update is provided accordingly. Parameter: callback — callback method to send the response back.	Void
NGBucConfigUtils. getActiveVersion ResponseHandler	Handles JSON NGAPIResponse received corresponding to getActiveVersion. Parameter: response — Object of APIResponse type.	Returns formatted JSON response in case of success, exception text if an exception is received from server and null if no response is received.

- **BUC Data Utils** — This file contains an interface for client-server interaction and BUC-related operations
- **Core** — This file contains the implementation of the data-utils interface for client-server interaction and BUC-related operations.

APIs name	Description and Parameter	Returns
NGBucDataUtils.getTray ForBucInstance (businessUseCase Instance)	Identify the tray(s) this Business Use Case instance is currently deposited into. Parameter: businessUseCaseInstance [in]: An object of type NGBusinessUseCase.	Returns the list of NGTray objects.

APIs name	Description and Parameter	Returns
NGBucDataUtils.getInstancesForTray (ngTray)	Function will get the instances (array) stored locally in the specified tray. Parameter: ngTray: An object of type NGTray.	Returns the list of NGBusinessUse Case objects.
NGBucDataUtils.pushInstanceInTray (businessUseCaseInstance, ngTray)	Push a specified instance in the specified tray. Parameters: <ul style="list-style-type: none"> businessUseCaseInstance — An object of type NGBusinessUseCase. ngTray — An object of type NGTray. 	Returns the updated NGBusinessUse Case object.
NGBucDataUtils.getBucInstanceFromUniqueld(buc, uniqueld)	Method will return the Business Use Case Instance from storage by Unique ID. Parameters: <ul style="list-style-type: none"> buc — Business Use Case uniqueld — Unique ID 	Return the instance of BUC
NGBucDataUtils.getListOfAttachments (businessUseCaseInstance)	Method will returns array of NGAattachment objects from defined Business Use Case Instance. Parameter: businessUseCaseInstance — An object of type NGBusinessUseCase.	Return the array of all attachments containing bucLevel Attachments ,form LevelAttachments, sectionLevel Attachments ,field LevelAttachments
NGBucDataUtils.listPendingWork (businessUseCaseInstance)	Lists all forms that are pending to be filled in a given BUC instance Parameter: businessUseCaseInstance — An object of type NGBusinessUseCase.	List all forms that are pending to be filled

APIs name	Description and Parameter	Returns
NGBucDataUtils.pop InstanceFrom Tray(business UseCaseInstance, ngTray)	Pop the specified instance in the specified tray. Parameters: <ul style="list-style-type: none"> • businessUseCaseInstance — An object of type NGBusinessUseCase. • ngTray — An object of type NGTray. 	Return object of NGBusiness UseCase.
NGBucDataUtils.put GeoTag (ngObject,callback)	Method will insert the current geo- location. Parameters: <ul style="list-style-type: none"> • ngObject — An object of type any JS-Object which is defined above. • callback — callback method for handling the return. 	Void
NGBucDataUtils.save BUC(bucInstance)	Method will save the Business Use Case instance into the defined storage. Parameter: Instance — Object of NGBusinessUse Case.	Void
NGBucDataUtils.search BUCInstances (instanceArray, pattern)	Method will return array of matching NGBusiness UseCase objects with specified search pattern. Parameters: <ul style="list-style-type: none"> • instanceArray — array of NGBusiness UseCase objects • pattern — Search pattern. 	Return array of matching NGBusinessUse Case objects.

APIs name	Description and Parameter	Returns
NGBucDataUtils.submit BUCInstance(destination, business UseCaseInstance)	Method will instantiate the correct NGSubmission Manager interface to submit the buc Instance Parameters: <ul style="list-style-type: none"> • destination — destination (server or client) where to submit the BUC object • businessUseCaseInstance — An object of type NGBusinessUseCase. 	Return Boolean (either submit or not).
NGBucDataUtils.handle Response(response)	It passes the response coming from the server and fetches the data provided by the server. Parameter: response — Object of APIResponse type	Return array of resources received from the server.
NGBucDataUtils.save BUC(bucInstance)	Method will save the Business Use Case instance into the defined storage. Parameter: Instance — Object of NGBusinessUse Case.	Void
NGBucDataUtils.search BUCInstances (instanceArray, pattern)	Method will return array of matching NGBusiness UseCase objects with specified search pattern. Parameters: <ul style="list-style-type: none"> • instanceArray — array of NGBusiness UseCase objects • pattern — Search pattern. 	Return array of matching NGBusinessUse Case objects.

APIs name	Description and Parameter	Returns
NGBucDataUtils.submit BUCInstance(destination, business UseCaseInstance)	Method will instantiate the correct NGSubmission Manager interface to submit the buc Instance Parameters: <ul style="list-style-type: none"> • destination — destination (server or client) where to submit the BUC object • businessUseCaseInstance — An object of type NGBusinessUseCase. 	Return Boolean (either submit or not).
NGBucDataUtils.handle Response(response)	It passes the response coming from the server and fetches the data provided by the server. Parameter: response — Object of APIResponse type	Return array of resources received from the server.
NGBucDataUtils.save BUC(bucInstance)	Method will save the Business Use Case instance into the defined storage. Parameter: Instance — Object of NGBusinessUse Case.	Void
NGBucDataUtils.search BUCInstances (instanceArray, pattern)	Method will return array of matching NGBusiness UseCase objects with specified search pattern. Parameters: <ul style="list-style-type: none"> • instanceArray — array of NGBusinessUseCase objects • pattern — Search pattern. 	Return array of matching NGBusinessUse Case objects.

APIs name	Description and Parameter	Returns
NGBucDataUtils.submit BUCInstance(destination, business UseCaseInstance)	Method will instantiate the correct NGSubmission Manager interface to submit the buc Instance Parameters: <ul style="list-style-type: none"> • destination — destination (server or client) where to submit the BUC object • businessUseCaseInstance — An object of type NGBusinessUseCase. 	Return Boolean (either submit or not).
NGBucDataUtils.handle Response(response)	It passes the response coming from the server and fetches the data provided by the server. Parameter: response — Object of APIResponse type	Return array of resources received from the server.
NGBucDataUtils.get ConfiguredForms InBuc(buc TemplateName)	It fetches and returns the list of 'Forms' configured in the provided BUC template. Parameter: bucTemplateName — name of BUC Template	Returns list of Forms
NGBucDataUtils.get Configured AttachmentsInBuc (buc TemplateName)	It fetches and returns the list of 'Attachments' configured in the provided BUC template. Parameter: bucTemplateName — name of BUC Template	Returns list of Attachments
NGBucDataUtils. getConfigured TraysInBUC (business UseCaseTemplate)	Method will retrieve all tray configuration from Business Use Case Template Parameter: businessUseCaseTemplate — An object of type NGBusinessUseCaseTemplate.	Void

APIs name	Description and Parameter	Returns
NGBucDataUtils.set TrayNames ForBUC(businessUse CaseInstance, ngTray)	This methods push the given tray into the BUC instance object given tray. Parameters: <ul style="list-style-type: none"> • object — NGBusinessUseCase businessUseCaseInstance • object/string — NGTray ngTray 1. string — String ngTray 	Returns NGBusinessUse CasebusinessUse CaseInstance on success.
NGBucDataUtils.remove TrayNames FromBuc(bucInstance Tray)	It removes the name of the trays in tray attributes of BUC instance. Parameter: bucInstanceTray — name of tray	Returns updated BUC instance.
NGBucDataUtils.get FormProgress Percentage(form)	It calculates the percentage completion of corresponding form. Parameter: form — object of NGForm type.	Returns percentage completion
NGBucDataUtils. getProgress Percentage (business UseCaseInstance)	It calculates the percentage completion of corresponding BUC instance. Parameter: buc — object of NGBusinessUseCase type	Returns percentage completion
NGBucDataUtils. getListOf FormAttachments (form)	Fetches and returns the list of attachments attached with the respective form. Parameter: form — object of NGForm type	Array of attachments
NGBucDataUtils. getAttachments Statistics (bucInstance)	Creates an object that fills itself with array of all the attachments, BUC level attachments, Form level attachments, Selection level attachments, Field level attachments. Parameter: bucInstance — object of NGBusinessUseCase type	JSON objects containing array of attachments at various levels.

APIs name	Description and Parameter	Returns
NGBucDataUtils.encryptAppData(data)	Encrypts provided data using secret key fetched from the server. Parameter: data — a string or object to be encrypted	Encrypted data
NGBucDataUtils.decryptAppData(encryptedData)	Decrypts provided encrypted data using secret key received from the server. Parameter: encryptedData — Encrypted data to be decrypted	Void
NGBucDataUtils.encryptAppData_V2(data)	Encrypts provided data using secret key and data encryption algorithm fetched from the server. Parameter: data — a string or object to be encrypted	Encrypted data
NGBucDataUtils.decryptAppData_V2(encryptedData)	Decrypts provided data using secret key and data encryption algorithm fetched from the server. Parameter: encryptedData — a string or object to be decrypted	Decrypted data as a string
NGBucDataUtils.saveAttachment(key, attachments, storageType, buc)	It stores an attachment object in the required storage time with the name provided in key parameter. Parameters: <ul style="list-style-type: none"> • key — string indicating file name • attachments — object of NGAAttachment type • storageType — type of storage • buc: name of BUC 	Void

APIs name	Description and Parameter	Returns
NGBucDataUtils. fetchAttachment (key, storageType, buc, callback)	It fetches an attachment saved by the name provided in the key and returns the result as a parameter to the provided callback function. Parameters: <ul style="list-style-type: none"> • key — string indicating file name • storageType — type of storage • buc — name of BUC • callback — callback function for returning result. 	Returns attachment.
NGBucDataUtils. clearBucInstance FromTrays (bucInstance)	Deletes provided BUC instance from all the trays it is present. Parameter: bucInstance — object of NGBusinessUseCase type	Void
NGBucDataUtils. removeBUC (bucInstance)	Deletes provided BUC instance from the BUC instance list. Parameter: bucInstance — object of NGBusinessUseCase type	Void
NGBucDataUtils. isBucEncrypted (bucName)	It determines whether the instance of the provided BUC requires encryption or not. Parameter: bucName — name of BUC	Returns Boolean
NGBucDataUtils. getListOf AllInstances()	Creates and returns lists of all the BUC instances present in the application.	Returns array of BUC instances.
NGBucDataUtils. getListOfBuc Instances (bucName)	It creates and returns list of BUC instances present which belongs to the corresponding BUC. Parameter: bucName — name of BUC	Returns list of BUC instances.

APIs name	Description and Parameter	Returns
NGBucDataUtils. getIndexingFields ForBuc(bucName)	It fetches the indexing fields of list of a BUC. Parameter: bucName — name of BUC	Returns list of indexing fields.
NGBucDataUtils. sortBucInstances (instanceList, indexingFields)	Sorts the BUC instances of the corresponding instances on the basis of corresponding instances indexing fields. Parameters: <ul style="list-style-type: none"> • instanceList — list of instances to be sorted. • indexingFields — JSON object having indexing field information. 	Returns sorted instance list.
NGBucDataUtils.create SortedString (jsonObject,fieldNames, sortingString,check)	This method will return the string by concatenating the values of JSON corresponding to fieldNames. Parameters: <ul style="list-style-type: none"> • jsonObject — Output model of BUC • fieldnames — Array fieldNames • sortingString — String value created as a result of indexing fields. • check — internal check value for execution of the method. 	Returns String sorting String on success.
NGBucDataUtils. sendRequest (request,callback)	This method sends request to the server. Parameters: <ul style="list-style-type: none"> • request — API Request • callback — function object that returns • the response asynchronously 	Returns response received from the server.

APIs name	Description and Parameter	Returns
NGBucDataUtils deleteAttachment (key,storageType,buc)	This methods will delete the attachment on local storage (based on storage type) of a particular BUC name. Parameters: <ul style="list-style-type: none"> • Key — String key • storageType — String storageType • buc — String buc 	Void
NGBucDataUtils.getGeo Coordinates(callback)	It fetches latitude and longitude coordinates of the current location. Parameter: callback — callback function for returning result	Returns JSON containing latitude and longitude information.
NGBucDataUtils. getMaster ListFromServer ()	This method fetches the list of master names configured at server.	Returns Array of master names.
NGBucDataUtils. getMasterListFrom ServerResponse Handler (response)	This method is corresponding to handle the response with respect to getMasterListFromServer. Parameter: apiResponse — Object of APIResponse type	Returns Array of master names.
NGBucDataUtils. getMasterData FromServer (master)	This method fetches the master options list for a master configured at server. Parameter: object — Object of type NGMaster whose data is to be fetched	Returns Object of Master type filled with options data.
NGBucDataUtils.get MasterDataFrom ServerResponseHandler (response)	This method is corresponding to handle the response with respect to getMasterDataFromServer. Parameter: object — NGAPIResponse response NGAPI Response object	Returns Object of Master type filled with options data.

APIs name	Description and Parameter	Returns
NGBucDataUtils.checkForMasterUpdate(master)	This method checks for any updates in masters at the server side. Parameter: object — object of type NGMaster	Returns updated master object.
NGBucDataUtils.checkForMasterUpdateResponseHandler(response)	This method is corresponding to handle the response wrt to checkForMasterUpdate. Parameter: object: NGAPIResponse response NGAPI Response object	Returns updated master object.
NGBucDataUtils.getTrayFromTrayName(currentBucName, trayName)	This method gets the instance of Trays corresponding to tray name and BUC Name. Parameters: <ul style="list-style-type: none"> • string — String currentBUCName • string — String trayName 	Returns NGTray object on success.
NGBucDataUtils.sendAttachmentToServer(attachmentDetailsJSON)	This method sends attachment to server in background. Parameter: attachmentDetailsJSON — attachment object.	Void
NGBucDataUtils.checkForMastersUpdate(masterList)	This method checks for update on the present masters list all at once. Parameter: masterList — the list of masters to be updated.	Void

APIs name	Description and Parameter	Returns
NGBucDataUtils. getInstancesFor RetentionTray(buc, ngTray, callback)	This method sets the tray object according to the required user, and returns a tray object. Parameters: <ul style="list-style-type: none"> • buc — the buc in which the tray is configured • ngTray — the tray for which the instances have to be fetched • callback — callback method to send the response back. 	Void
NGBucDataUtils. getMasterData FromServer (master)	This method fetches the master options list for a master configured at server. Parameter: object — Object of type NGMaster whose data is to be fetched	Returns Object of Master type filled with options data.
NGBucDataUtils.get MasterDataFrom ServerResponseHandler (response)	This method is corresponding to handle the response with respect to getMasterDataFromServer. Parameter: object — NGAPIResponse response NGAPI Response object	Returns Object of Master type filled with options data.
NGBucDataUtils.check ForMasterUpdate (master)	This method checks for any updates in masters at the server side. Parameter: object — object of type NGMaster	Returns updated master object.
NGBucDataUtils. checkForMaster UpdateResponse Handler(response)	This method is corresponding to handle the response wrt to checkForMasterUpdate. Parameter: object: NGAPIResponse response NGAPI Response object	Returns updated master object.

APIs name	Description and Parameter	Returns
deleteBUCInstance OnLongPress	This method deletes a buc instance when it is longpressed on dashboard.	Void
addWaterMark(img, angle, watermarkText, fontSize, fontType, transparency, watermarkColor, positionX, positionY)	<p>This method adds water mark on the provided image.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • img — the image on which water mark is • to be applied • angle — the angle of the watermark. • WatermarkText — the text to be applied as the watermark. • FontSize — fontsize of the watermark text • transparency — the transparency of the watermark text. • Watermarkcolor — the color of the water mark text • positionX — the x position of the watermark. • PositionY — the y position of the watermark. 	Base64 data of the watermark applied image.
retentionPolicyJob (asyncResponse)	<p>This method runs the retention job on the configured trays.</p> <p>Parameter:</p> <p>asyncResponse — the event generated for retention job.</p>	Void
pushCrash LogsToServer()	This method fetches the crash logs from the device (if any) and pushes the file to server.	Void
logout	This method logs the user out of the current session.	Void
updateApk	This method updates the APK downloaded from the server or play store	Void

APIs name	Description and Parameter	Returns
downloadApk	This method downloads the APK from the url received in the API response.	Void
getAppCurrentVersion(callback)	This method gets the current version of the running app. Parameter: callback — callback method to return the version number.	Void

- **BUC Device Utils** — This file contains an interface that includes media capture using device utilities.
- **Core** — This file contains the implementation of uiUtils interface for UI utilities and media operations in the application.

APIs name	Description and Parameter	Returns
invokeCameraForImageCapture(parameters, callback)	Will invoke plug-ins to invoke camera for image capture. Parameters: <ul style="list-style-type: none"> • parameters — reserved for future use. • callback — callback method for handling the return. 	Returns an NGAttachment object.
invokeCameraForBarcodeCapture(parameters, callback)	Method will invoke a camera through plug-in for scanning barcode. Parameters: <ul style="list-style-type: none"> • parameters — reserved for future use • callback — callback method for handling the return. 	Returns a decoded bar-code string.
invokeCameraForZoneCapture(parameters, callback)	This will invoke a camera through plug-in for capturing the Zone drawn by the user on the captured image and returns the array or co-ordinates. Parameters: <ul style="list-style-type: none"> • parameters — reserved for future use. • callback — callback method for handling the return. 	Callback

APIs name	Description and Parameter	Returns
invokeCamera ForAdvancelImage Capture(parameters, callback)	Method will call plug-in to invoke camera for image capture. Parameters: parameters : reserved for future use callback: callback method for handling the return.	Will return an NGAttachment object
invokeCamera ForVideoCapture (parameters, callback)	This will invoke plug-in to invoke camera for video capturing. Parameters: <ul style="list-style-type: none"> parameters — reserved for future use. callback — callback method for handling the return. 	Returns an NGAttachment object.
invokeCamera ForMultipleImage Capture(callback)	Enables user to capture multiple images at a time. Parameter: callback — callback function for returning result	Required
invokeStart VoiceCapture (parameters,callback)	Method for voice capture. Parameters: <ul style="list-style-type: none"> parameters — reserved for future use. callback — callback method for handling the return. 	Returns an NGAttachment object.
invokeStop VoiceCapture()	Method for stop voice capture	Void
invokeGallery Selection (parameters, callback)	Method for image, video, and audio selection from existing gallery items. Parameters: <ul style="list-style-type: none"> parameters — reserved for future use. callback — callback method for handling the return. 	Returns an array of NGAttachment objects.

APIs name	Description and Parameter	Returns
getMediaList FromDevice (path,successCallback, errorCallback)	<p>This method will fetch all files from the folder specified in the path.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • path — path of the folder, for example gallery. • successCallback — success Callback method for handling the return. • errorCallback — error Callback method for handling the return. 	Returns array of files with absolute path.
invokeCustomCamera ForImagecapture (parameters,success Callback,errorCallback)	<p>This Method will call custom camera plug-in.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • parameters: It is JSON that contains: <ul style="list-style-type: none"> ◦ height — Targeted height of the image. ◦ width — Targeted width of the image. ◦ quality — Targeted quality of the image. ◦ autoFocus — Targeted focus of the image. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	Will return an NGAttachment object.

- **BUC UI Utils** — This file contains an interface for UI utilities and media operations.
- **Core** — This file contains the implementation of uiUtils interface for UI utilities and media operations in the application.

APIs name	Description and Parameter	Returns
invokeImageViewer(attachment)	This method will open attachment in HTML5 screen. Parameter: attachment — attachment is an NGAttachment object containing image.	Void
invokeVideoPlayer(attachment)	Method will start video play. Parameter: attachment — attachment is an NGAttachment object containing video.	Void
invokeAudioPlayer(attachment)	Method will start audio play. Parameter: attachment — attachment is an NGAttachment object containing audio	Void
lockAppUI()	Method will lock Application UI.	Void
unlockAppUI()	Method will unlock Application UI.	Void
showView(viewName)	This method displays the mentioned view. Parameter: viewName — view to be displayed.	Void
setOnlineOfflineConfig	This method handles switching of app from offline to online mode and vice versa.	Void

- **Master ManagerUtils** — In this interface the framework provides utility methods for the master data received from the server.
 - [Storing data in file system]
 - FILESYSTEM
 - SQLITE [Storing the data in SQLITE mobile database]
 - DB-FROM-SERVER [Fetching an entire DB of Master data from server and saving it on client]
 - DB-FROM-ASSETS [Use a Master DB file packed along with the ipa/apk].

APIs name	Description and Parameter	Returns
NGMasterManager. setMasterListJson (masterDataJson)	Method will show the history of masters coming from server. Parameter: masterDataJson — Array of masters	Void
NGMasterManager. getMasterListJson (callback)	Method will retrieve the list of masters from persistent storage. Parameter: Callback — It is a function that will be automatically called after retrieving result from the persistent storage.	Returns the array of master list
NGMasterManager. setMasterDataFor MasterKey (masterKey,masterData)	Method will restore the data of specific master. Parameters: <ul style="list-style-type: none"> • masterKey — Name of the master. • masterData — Data for the master. 	Void
NGMasterManager. getMasterData ForMaster (masterName,callback)	Method will retrieve data for specific master that is stored in persistent storage. Parameters: <ul style="list-style-type: none"> • masterName — Name of the master. • Callback — It is a function that will be automatically called after retrieving • result from the persistent storage. 	Returns the value of master data in the Json format

APIs name	Description and Parameter	Returns
NGMasterManager. getMatserData (masterKey,callback)	Method will retrieve data for specific master that is stored in persistent storage. Parameters: <ul style="list-style-type: none"> • masterKey — Name of the master. • Callback — It is a function that will be automatically called after retrieving the result from the persistent storage. 	Returns the value of master data in the Json format
NGMasterManager. getColumnSpecific MasterData (masterKey, columnName, callback)	Method will retrieve the values of specific column of specific master. Parameters: <ul style="list-style-type: none"> • masterKey — Name of the master • columnName — Input column name for which user needs to get data. • Callback — It is a function that will be automatically called after retrieving result from the persistent storage. 	Returns master value with respect to column in the form of array.
NGMasterManager. getColumnSpecific MasterData (masterKey, columnName, callback)	Method will retrieve the values of specific column of specific master. Parameters: <ul style="list-style-type: none"> • masterKey — Name of the master. • columnName — Input column name for which user needs to get data. • Callback — It is a function that will be automatically called after retrieving result from the persistent storage. 	Returns master value with respect to column in the form of array.

- **Key Provider** — In this interface, the framework provides functionality to get the secret key using either of the below two implementations:

- AWS (Amazon Web Service) — Secret key is different for each user and is received from AWS.
- Configuration — Secret key is same for each user as configured on the server.

Methods:

- `getEncryptionKeyFromServer`
- `getEncryptionKeyFromServerResponseHandler`
- **PDF Generator** — This interface provides functionality to generate PDF with either a static HTML string or some data and/or attachments from the form currently open.

Methods:

- `generatePdfWithHtmlString` — a PDF file can be generated using a static HTML string.
 - `printNemfPDF` — a PDF file can be generated using some data from the current form and/or attachments (images only).
 - To access these APIs the following methods have been exposed in *ng-buc-data-utils-core.js*:
- `$NGDatautilsImpl.generatePdfWithHtmlString(configuration, callback)`

```
var configuration = {
  "showInExternalPdfViewer": true,
  "attachPdfWithBuc": true,
  "margin": [1,0,1,0],
  "pageBreak": "",
  "attachmentName": "nemf-pdf-generate",
  "htmlTemplate": '<div class="form-section-body panel-body form-section yesOrNo" id="basicDetails"></div>'
}
```

Key name	Type	Default value	Remarks
margin	Number or array	0	PDF margin(in jsPDF units). Can be a single number too. [vMargin, hMargin] or [top, left, bottom, right].
pageBreak	Array or string	Contains .class as ".eg" or #id as "#eg"	Controls the page break behaviour in the pdf. Can also be an array [".eg", "#eg"].

Key name	Type	Default value	Remarks
htmlTemplate	String	Pass value in " single quotes	Restriction: <ul style="list-style-type: none"> • Image tag can contain only base 64 data. • Must not contain comments. • Must pass as a single line and in enclosing "div".
showInExternalPdf Viewer	Boolean	true	To generate a pdf and get its base 64 data or to show in an external viewer only mark it as "true".
attachPdfWithBuc	Bool	true	If you want to attach PDF with BUC instance you need to pass the "attachWithBuc" tag as "true" else pass it as "false".
AttachmentName	String	"	Pass name of the attachment where you want to attach your generated PDF. If you do not need to attach the PDF to the BUC, pass it as "" else provide the name.

```

var configuration = {
    "showInExternalPdfViewer": true,
    "attachPdfWithBuc": true,
    "margin": [1,0,1,0],
    "pageBreak": "",
    "attachmentName": "nemf-pdf-generate",
    "htmlTemplate": '<div class="form-section-body panel-body form-section yesOrNo" id="basicDetails"></div>'
}

NGBucDataUtils.generatePdfWithHtmlString(configuration, function(response){
    spinner.hideSpinner();

    var message = response["message"]
    if(message == "success"){
        var responseMessage = response["responseMessage"]
        customBootboxAlert($NGDataUtilsImpl.getTextValue(responseMessage, "pdfGenerator"));
    }else{
        customBootboxAlert($NGDataUtilsImpl.getTextValue(message, "pdfGenerator"));
    }
});

```

- \$NGDataUtilsImpl.printNemfPDF(configuration, attachmentNames, callback)

```

var configuration = {
    "showInExternalPdfViewer": true,
    "attachPdfWithBuc": true,
    "mainLogo": undefined,
    "attachmentName": "nemf-pdf"
}

```

Key name	Type	Default value	Remarks
mainLogo	String	"	base 64 data of the logo. If you don't pass logo base 64 data then it will take the default value and create a PDF.
showInExternalPdf Viewer	Bool	true	To generate a PDF and get its base 64 data or to show in an external viewer only mark it as "true".
attachPdfWithBuc	Bool	true	If you want to attach PDF with BUC instance you need to pass the "attachWithBuc" tag as "true". Else pass it as "false".

Key name	Type	Default value	Remarks
AttachmentName	String	"	Pass name of the attachment where you want to attach your generated PDF. If you do not need to attach the PDF to the BUC, pass it as "". Else provide the name.

To add a captured image attachment to the PDF generated using form data, follow the below steps:

1. Create an array with details of the attachments to be added.
2. Pass the array in the calling of the method as shown:

```

var configuration = {
  "showInExternalPdfViewer": true,
  "attachPdfWithBuc": true,
  "mainLogo": undefined,
  "attachmentName": "nemf-pdf"
}

var formFront = {"isFormLevel": false , "isSectionLevel": true, "sectionName": "applicantDetailsSection-1", "formName":
  "ClaimerForm", "attachmentName":"applicant-signature-1"}
var attachmentNames = [formFront]
NGBucDataUtils.printNemfPDF(configuration,attachmentNames,function(response){
  spinner.hideSpinner();

  var message = response["message"]
  if(message == "success"){
    customBootboxAlert("Pdf attached successfully");
  }else{
    customBootboxAlert(message);
  }
});

```

Here, *attachmentNames* is the array which contains the details of the form attachments which need to be added to the generated PDF file. The parameters are defined as below:

- isFormLevel — true if the attachment is form level, false otherwise
- isSectionLevel — true if the attachment is section level, false otherwise
- sectionName — the name of the section as present in the form, in which the attachment exists
- formName — name of the form in which the attachment present
- attachmentName — name of the attachment as present in the form.

To configure which elements need to be added to the generated PDF add an additional class 'pdf-group' and an attribute 'alias' to the immediate or nearest

div element of the corresponding form element. The alias(if provided) is used instead of the actual field ID in the PDF.

```
'<div class="form-group pdf-group"> <input id="account-number"
ismasked="true" maskType="partial" class="form-element fieldValue mt10"
type="number" onblur="NGFormValidation.checkAccountNo(this) "
name="account-number" required> <label class="fieldName" alias="My
Account No." for="account-number">Account Number</label> </div>'+
```

Common configurations for PDF generation:

If it is required to attach the generated PDF file to the form currently open, a new attachment element needs to be created in the view as follows:

```
'<li> <a href="#" focusValue="true" photoWidth = "" photoHeight=""
photoQuality="" class="attach pdf-attachment" attachmentLevel =
"form" attachmentFieldId ="ClaimerForm" attachmentName="nemf-pdf-generate"
id="nemf-pdf-generate" minAttachments="1" maxAttachments = "1"
```

Plug-ins

- **JailBreakDetection (iOS only)** — NGJailBreakDetection.isJailBroken (successCallback, errorCallback).
This method tells whether the device is jail broken or not.
Parameters:
 - successCallback — It is a method called on the success of plug-in.
 - errorCallback — It is a method called in case of error caused in the plug-in.
- **Aadhar Biometric** — This plug-in is used to capture thumb impression using a capturing device.

Method name	Description and Parameter	Returns
NGAadharBiometricPlugin.captureThumbImpression(successCallback, errorCallback)	<p>This method is used for capturing the thumb impression.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	Byte code of the thumb impression captured.

- **Advance Camera** — This plug-in is used to capture image and then perform multiple imaging operations on that particular captured image.

Method name	Description and Parameter	Returns
NGAdvanceCameraManager.captureImageUsingAdvanceCamera(successCallback, errorCallback, quality)	<p>This method will capture the image and allow performing a sequence of imaging operations on the captured image.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. • quality — This parameter defines quality of image. 	outImg: processed image

- **App Initialization** — This plug-in is used to store white labeling information (Foreground color, background color and Folder name) received from the server.

Method name	Description and Parameter	Returns
NGAppDataInitializationManager.setAppInitializationInfo(initData, successCallback, errorCallback)	<p>This method will provide the background and foreground color for the native plug-ins and the name of the folder in which the app data gets saved.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • initData — It is a JSON that has data to be used by the plug-in. <ul style="list-style-type: none"> ◦ backgroundColor — Background color for the native plug-in. ◦ foregroundColor — Foreground color for the native plug-in. ◦ appStorageFolderName — Name of the folder where app data is to be stored. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	outImg: processed image
NGAppDataInitializationManager.checkForFolderUpdation(successCallback, errorCallback)	<p>This method will detect whether the application on device is launched first time or not, also it determines presence of file system folders in device memory.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	JSON object with folder existence and app's primary data.

Method name	Description and Parameter	Returns
NGAppDataInitializationManager.deleteSdCardFolder(result, successCallback, errorCallback)	<p>This method will enable deletion of file system folders from the device memory.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • result — defines success of folder deletion. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	Void

- **Bar-Code Extraction** — This plug-in enables bar code extraction operation for extracting bar-code data from camera.

Method name	Description and Parameter	Returns
NGBarcodeExtractionManager.readBarcode(successCallback, errorCallback)	<p>This method will read the barcode.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	barcode data

- **Canvas** — This plug-in is used for manual drawing operation on canvas.

Method name	Description and Parameter	Returns
NGCanvasManager.drawOnCanvas (successCallback, errorCallback, quality)	<p>This method will open a canvas to draw and allow signing on the canvas.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. • quality — This parameter defines quality of image. 	outImg: snapshot of the signature.

- **Custom Camera** — This plug-in is used to capture image in a desired resolution parameters. It will launch a camera with the nearest target resolution.

Method name	Description and Parameter	Returns
NGCustomCameraManager.captureImageUsingCustomCamera(parameters, successCallback, errorCallback)	<p>This method is used for capturing the image with desired resolution.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • parameters — It is JSON that contains <ul style="list-style-type: none"> ◦ height — Targeted height of the image. ◦ width — Targeted width of the image. ◦ quality — Targeted quality of the image. ◦ autoFocus — Targeted focus of the image. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	File path of a captured image captured with nearest target resolution.

- **File Storage Manager** — This plug-in is used for file storage operations with the file system (set, get, delete, and more).

Method name	Description and Parameter	Returns
NGFileStorageManager.setDataInFileSystem(key, value)	<p>This method will set data in the File system.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • key — It is the key corresponding to which the data is to be set. • value — It is the data which is to be set. 	Void
NGFileStorageManager.getDataFromFileSystem(key, successCallback)	<p>This method will fetch data from the File system.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • key — It is the key corresponding to which the data is to be fetched. • successCallback — It is a method called on the success of plug-in. 	Value corresponding to key passed
NGFileStorageManager.setTTLForData(key, successCallback)	<p>This method will set the time for which the key shall remain in the File system.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • key — It is the key whose lifetime is set. • successCallback — It is a method called on the success of plug-in. 	Void

Method name	Description and Parameter	Returns
NGFileStorageManager.deleteDataFromFileSystem(key, successCallback)	<p>This method will delete key from the File system.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • key — It is the key which is to be deleted. • successCallback — It is a method called on the success of plug-in. 	Success or Error according to file found and deleted.
NGFileStorageManager.deleteFolderFromFileSystem(successCallback, errorCallback)	<p>This method will delete the specified folder from the File system.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • successCallback — It is a method called on the success of plug-in. • errorCallback — It is the method called on the failure of plug-in. 	Void

- **GPS** — This plug-in is used to fetch latitude, longitude and address of user's current location.

Method name	Description and Parameter	Returns
NGGPSManager.requestGPSEnabling (successCallback, errorCallback)	<p>This method requests the user to enable the GPS if it is disabled.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	GPS enabled or not.
NGGPSManager.checkGPSStatus (successCallback)	<p>This method checks the GPS status.</p> <p>Parameter:</p> <p>successCallback — It is a method called on the success of plug-in.</p>	True or false according to GPS enabled status.

Method name	Description and Parameter	Returns
NGGPSManager.getAddressFromGps(latitude, longitude, successCallback, errorCallback)	<p>This method fetches the address using the longitude and latitude of the location.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • latitude — The latitude of the location. • longitude — The longitude of the location. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	The address
NGGPSManager.getLocationMapFromGps(latitude, longitude, successCallback)	<p>This method gets the map of the location using the latitude and longitude.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • latitude — The latitude of the location. • longitude — The longitude of the location. • successCallback — It is a method called on the success of plug-in. 	Map of the location

Method name	Description and Parameter	Returns
NGGPSManager.checkGPSPermissionStatus(successCallback)	<p>This method checks the GPS permission status for the app.</p> <p>Parameter:</p> <p>successCallback — It is a method called on the success of plug-in.</p>	True or false according to GPS permission status.

- **Image Processing** — This plug-in contains all image processing operations, which are performed on a camera-captured image.

Method name	Description and Parameter	Returns
NGImageProcessingManager.isDefocusDetected(image, successCallback, errorCallback, quality)	<p>This method will automatically identify any defocus defect present in the mobile captured image.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • image — base64 data of mobile captured image. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. • quality — This parameter defines quality of the image. 	Returns defocus defect status.

Method name	Description and Parameter	Returns
NGImageProcessingManager. isGlare Detected(image, successCallback, errorCallback, quality)	<p>This method will automatically identify that is there any Glare defect present in the mobile captured image.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • image — base64 data of mobile captured image. • successCallback — It is a method called on the success of plug-in. • errorCallback— It is a method called in case of error caused in the plug-in. • quality — This parameter defines quality of the image. 	Returns glare Defect status.

Method name	Description and Parameter	Returns
NGImageProcessingManager. binarize (image, option, compressionType, xDPI, yDPI, TiffFileName, successCallback, errorCallback, quality)	<p>This method to perform the black and white conversion of mobile captured image and generate a G4 compressed Tiff file corresponding. All different methods of binarization can be chosen by providing the options</p> <p>Parameters:</p> <ul style="list-style-type: none"> • image — base64 data of mobile captured image which is converted into black and white. • option — options for choosing binarization methods. <ul style="list-style-type: none"> ◦ Adaptive Binarization ◦ Integral thresholding. • compressionType — applied compression (currently supported only G4) • xDPI — xDPI of tiff image • yDPI — yDPI of tiff image • TiffFileName — absolute path of tiff image. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. • quality — This parameter defines the quality of the image. 	Returns path of binarized image stored in device memory.

Method name	Description and Parameter	Returns
NGImageProcessingManager. autoCrop (image, successCallback, errorCallback, quality)	<p>This method will automatically crop the document from the mobile captured image by using the corner of the present document.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • image — base64 data of mobile captured image. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. • quality — This parameter defines the quality of the image. 	outImage: byte data of cropped image;
NGImageProcessingManager. autoAdjust Brightness(image, successCallback, errorCallback, quality)	<p>This method will automatically adjust the brightness of mobile captured image and returned as enhanced image as output.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • image — base64 data of mobile captured image. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. • Quality — This parameter defines the quality of the image. 	Byte data of cropped image;

Method name	Description and Parameter	Returns
NGImageProcessingManager. autoDetect Corners(image, successCallback, errorCallback, quality)	This method will detect corners of the image. Parameters: <ul style="list-style-type: none"> • image — base64 data of mobile captured image. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. • Quality — This parameter defines the quality of the image. 	Corners of the image.
NGImageProcessingManager. rotate Image(image, angle, successCallback, errorCallback, quality)	This method will rotate the given image to the specified angle. Parameters: <ul style="list-style-type: none"> • image — base64 data of mobile captured image. • angle — angle with which the orientation of the image is to be changed. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. • Quality — This parameter defines the quality of the image. 	outImg: rotated image.

Method name	Description and Parameter	Returns
NGImageProcessingManager. convert ColorToGrey(image, successCallback, errorCallback, quality)	This method will convert colored image to grey. Parameters: <ul style="list-style-type: none"> • image — base64 data of mobile captured image. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. • Quality — This parameter defines the quality of the image. 	Grey image.
NGImageProcessingManager .convert GreyToColor(image, successCallback, errorCallback, quality)	This method will covert grey image to colored image. Parameters: <ul style="list-style-type: none"> • image — base64 data of mobile captured image. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. • Quality — This parameter defines the quality of the image. 	Colored image.

Method name	Description and Parameter	Returns
NGImageProcessingManager. linear Equalization(image, successCallback, errorCallback, quality)	<p>This method enhances the RGB intensity of the image.</p> <p>Parameters:</p> <ul style="list-style-type: none">• image — base64 data of mobile captured image.• successCallback — It is a method called on the success of plug-in.• errorCallback — It is a method called in case of error caused in the plug-in.• Quality — This parameter defines the quality of the image.	Processed image.

Method name	Description and Parameter	Returns
NGImageProcessingManager. median Filter(image, filterSize, iOption, successCallback, errorCallback, quality)	<p>This method will remove noise from the image.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • image — base64 data of mobile captured image. • filterSize — size of filter window. • iOption — option for pixel conversion for black and white image. <ul style="list-style-type: none"> ◦ iOption = 0 — This option will convert some filtered white pixel into black and vice-versa. ◦ iOption = 1 — This option will convert some filtered black pixels into white. ◦ iOption = 2 — This option will convert some filtered white pixels into black. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. • Quality — This parameter defines the quality of the image. 	Processed image.

Method name	Description and Parameter	Returns
UIImageProcessingManager.invertData(image, successCallback, errorCallback, quality)	<p>This method will change the image into its negative.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • image — base64 data of mobile captured image. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. • quality — This parameter defines the quality of the image. 	Negative of image.
UIImageProcessingManager.detectSkewByCorners(image, successCallback, errorCallback, quality)	<p>This method will identify the skew angle of the image on the basis of corner points of the document present with the image.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • image — base64 data of mobile captured image. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. • quality — This parameter defines the quality of the image. 	Angle: skew angle of the image.

Method name	Description and Parameter	Returns
NGImageProcessingManager. perspective Correction(image, iSkewOption, successCallback, errorCallback, quality)	This method will perspective correct the image according to the corner points of the document. Parameters: <ul style="list-style-type: none"> • image — base64 data of mobile captured image. • iSkewOption — If its value is 1 than skew correction will be performed otherwise not. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug- in. • quality — This parameter defines the quality of the image. 	outImg: perspective corrected and deskewed(if iSkewOption = 1) image.
NGImageProcessingManager. extractZone(image, coordinates, successCallback, errorCallback)	This method will extract a zone defined in image for OCR text. Parameters: <ul style="list-style-type: none"> • image — base64 data of mobile captured image. • Coordinates — coordinates of the zone to be extracted. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug- in. 	Angle: skew angle of the image.

• **Media Player:**

- Gallery Selection — This plug-in is used to select multiple images from the gallery.

- Media Player — This plug-in is used to play audio and video files.

Method name	Description and Parameter	Returns
NGMediaManager. invokeGallery (successCallback, errorCallback, quality)	This method invokes gallery for the selection of image. Parameters: <ul style="list-style-type: none"> • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. • quality — This parameter defines the quality of the images. 	The array of base64 data of the images selected.
NGMediaManager. playAudioFile (fileBase64, type)	This method plays the audio file. Parameters: <ul style="list-style-type: none"> • fileBase64 — It is the base64 data of the • audio to be played. • type — It is the format of fileBase64. 	Success or Error while running the audio.
NGMediaManager. playVideoFile (fileBase64, type)	This method plays the video file. Parameters: <ul style="list-style-type: none"> • fileBase64 — It is the base64 data of the • audio to be played. • type — It is the format of fileBase64. 	Success or Error while running the audio.

- **SMS** — This plug-in is used to send text messages.

Method name	Description and Parameter	Returns
NGSMSManager.sendSms (smsDetailsJson, successCallback, errorCallback)	This method will send message to the specified contact number. Parameters: <ul style="list-style-type: none"> • smsDetailsJson — JSON is used to retrieve data used in the plug-in. • receiverNumber — the contact number to which the message is to be sent.body- the 	Status

Method name	Description and Parameter	Returns
	<p>message content or the data to be sent.</p> <ul style="list-style-type: none"> • smsText — the message body which is to be sent. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	

- **Spinner** — This plug-in is used to show and hide spinner.

Method name	Description and Parameter	Returns
Spinner.showSpinner(title, message, isFixed)	<p>This method will show spinner dialog.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • title — title of the dialog. • message — message to be displayed in the dialog. • isFixed —If value is 'isFixed', the spinner does not appear as a pop up and hides only when hideSpinner is called. 	Void
Spinner.hideSpinner()	This method will dismiss and hide the spinner dialog.	Void

- **SQLITE Manager** — This plug-in is used for file storage operations with the SQLITE.

Method name	Description and Parameter	Returns
NGSqliteManager.createSQLTable (successCallback, errorCallback)	<p>This method will create the SQL table if it does not exist.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	True if table created successfully.
NGSqliteManager.setDataInSqlTable (dataKey, dataValue, successCallback, errorCallback)	<p>This method will set data in the SQL Table.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • dataKey — It is the key corresponding to which the data is to be set. • dataValue — It is the data which is to be set. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	Success if data set.

Method name	Description and Parameter	Returns
NGSqliteManager.getDataFromSqlTable(dataKey, successCallback, errorCallback)	<p>This method will fetch data from the SQL table.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • dataKey — It is the key corresponding to which the data is to be fetched. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	The data value
NGSqliteManager.deleteKeyFromTable(dataKey, successCallback, errorCallback)	<p>This method will delete key from SQL table.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • dataKey — It is the key which is to be deleted. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	Success if key deleted.

- **WIFI-Printer** — This plug-in is used to print data via wifi printer (only for android).

Method name	Description and Parameter	Returns
NGWifiPrinterManager.printDocument (data, printerType)	<p>This method will print the data sent.</p> <p>Parameters:</p>	status

Method name	Description and Parameter	Returns
	<ul style="list-style-type: none"> • data — data to be printed. • printerType — the type of printer used. 	

- **Zone Extraction** — This plug-in is used for extracting a particular part of the captured image.

Method name	Description and Parameter	Returns
NGZoneExtractionManager. drawZoneOn CapturedImage (successCallback, errorCallback, quality)	<p>This method will allow drawing a zone on the canvas.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. • quality — This parameter defines the quality of the image. 	Map of the coordinates.
NGZoneExtractionManager. extractZone CapturedImage (successCallback, errorCallback, imgData, coordinateZone, quality)	<p>This method will capture the image and allow performing a sequence of imaging operations on the captured image.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. • imgData — byte data of mobile captured image. • coordinateZone — map of coordinates to be extracted from image. • Quality — This parameter defines the quality of the image. 	outImg: cropped image with respect to the coordinate map.

Method name	Description and Parameter	Returns
NGZoneExtractionManager. extractZone FromImage(quality, successCallback, errorCallback)	<p>This method will capture the image and allow drawing zone on captured image.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • Quality — This parameter defines the quality of the image. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	Extracted base64 image data.
NGZoneExtractionManager. extractZone FromImageData(imageBase64, quality, successCallback, errorCallback)	<p>This method will take the base64 data of an image and allow drawing zone on the image.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • imageBase64 — Base64 data of the image on which zone is to be drawn. • Quality — This parameter defines the quality of the image. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	Extracted base64 image data.

• **FingerPrint Login:**

APIs name	Description and Parameter	Returns
NGBUCAsyncUtils. updateBUCInstance (bucInstance,trayArray, callback)	This method will update the business use case instance in the defined tray. Parameters: <ul style="list-style-type: none"> • bucInstance — An object of type NGBusinessUseCase • trayArray — array of trays • callback — callback method to return the result. 	Void
NGBUCAsyncUtils. updateWhitelabelingInfo (whiteLabelingConfi)	This method updates a new UI of the application. Parameter: whiteLabelingConfig — Boolean value for processing whiteLabelingConfiguration.	Void
NGBUCAsyncUtils. updateBUCTemplate (bucTemplateconfiguration)	This method updates the application if any change is made in BucTemplate at the server side. Parameter: bucTemplateconfiguration — XML which contains new BucTemplate on the server-side.	Void

Method name	Description and Parameter	Returns
NGFingerprintManager. checkAvailability: (successCallback, errorCallback)	This method is used to check whether fingerprint hardware is present in the device or not. Parameters: <ul style="list-style-type: none"> • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	Returns a string “Available” or “No hardware detected”.

Method name	Description and Parameter	Returns
NGFingerPrintManager. registerFingerprint: (successCallback, errorCallback)	This method is used to register the user fingerprint first time the after successful login. Parameters: <ul style="list-style-type: none"> • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	Returns a string “Fingerprint Registered” if successfully registered.
NGFingerPrintManager. verifyFingerprint: (successCallback, errorCallback)	This method is used to verify the user fingerprint. Parameters: <ul style="list-style-type: none"> • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	Returns a string “Fingerprint Verified” if successfully verified.
NGFingerPrintManager. saveCredentials: (savedUserName, savedPassword, successCallback, errorCallback)	This method is used to save user name and password after successfully registering the fingerprint. Parameters: <ul style="list-style-type: none"> • savedUserName — username to save • savedPassword — user password to save. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	Returns success if successfully saved.
NGFingerPrintManager. fetchCredentials: (successCallback, errorCallback)	This method is used to fetch username and password after successful verification. Parameters: <ul style="list-style-type: none"> • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	Returns a json object with saved UserName, saved Password.

• **DocScan Plugin:**

Method name	Description and Parameter	Returns
NGDocScan.scanDocs : (parameters, successCallback, errorCallback)	<p>This method is used to capture multiple images at once and performing image operations on them.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • parameters — It is a JSON to modify image attributes that contains <ul style="list-style-type: none"> ◦ height — Targeted height of the image. ◦ width — Targeted width of the image. ◦ quality — Targeted quality of the image. ◦ autoFocus — Targeted focus of the image. ◦ doPerspectiveCorrection — whether or not to do perspective correction after image is clicked. ◦ totalNoOfImages — total no of images to be clicked with camera at once. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	Returns array of paths of images clicked.

• **File Explorer:**

Method name	Description and Parameter	Returns
NGFileExplorer.importFile: (docType, successCallback, errorCallback)	<p>This method is used to import all type of files from the file system.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • docType — type of document to be imported. • successCallback — It is a method called on the success of plug-in. 	Returns the path of file selected from file system.

Method name	Description and Parameter	Returns
	<ul style="list-style-type: none"> • errorCallback — It is a method called in case of error caused in the plug-in. 	

• **Location Spoofing:**

Method name	Description and Parameter	Returns
CLLocationSpoofingChecker.is LocationSpoofed : (successCallback, errorCallback)	This method is used to check if the device location is being spoofed or not. Parameters: <ul style="list-style-type: none"> • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	Returns a JSON object with latitude and longitude if location is not spoofed else it will give that location is spoofed.

• **Push Notification:**

Method name	Description and Parameter	Returns
NGPushNotificationManager. registerApp ForPushNotifications: (successCallback, errorCallback)	This method is used to register a device on FCM server and gets a push token. Parameters: <ul style="list-style-type: none"> • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	Returns a json object with push token.

Method name	Description and Parameter	Returns
NGPushNotificationManager. onWindowsPushNotification: (successCallback, errorCallback)	<p>This is the event handler that executes whenever a windows push notification is received. This handler takes out the custom Message from the notification and displays it in the tile.</p> <p>Parameter: Evt — The javascript notification event object</p>	Returns a JSON object with push token.
NGPushNotificationManager. onMessage : (pluginMessage)	<p>This method is called from native side. It gets the message from push notification and sends it to Javascript.</p> <p>Parameter: pluginMessage — JSON returned from the server of a synchronous call.</p>	Return: NA

- **Media Splitter:**

Method name	Description and Parameter	Returns
NGMediaSplitterPlugin.getTotalParts: (filePath, fileName, fileTempPath, attachmentType, chunkSize, userId, isAttachmentEncrypted, successCallback, errorCallback)	<p>This method is used to get the total number of parts an attachment will be divided into according to chunk size for submission.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • filePath — The folder path where the attachment data is saved. • fileName — The name of the attachment file. • fileTempPath — The name of the temporary folder where the attachment file is saved (in case the BUC is created from the older approach) • attachmentType — The type of attachment (base64EncodedData/base64TiffData) • chunkSize — The size of each part that the file needs to be divided into. • userId • isAttachmentEncrypted — Whether the attachment is encrypted or not. • • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	Returns a json object with push token.

Method name	Description and Parameter	Returns
NGMediaSplitterPlugin .splitFilesIntoParts: (fileName, fileTempPath, partIndex, chunkSize, fileReferenceName, folderName, isAttachmentEncrypted, successCallback, errorCallback)	<p>This method is used to split a file into chunk size parts for transfer to server.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • fileName — The name of the file to split into parts • fileTempPath — The name of the temporary folder where the attachment file is saved (in case the BUC is created from the older approach) • partIndex — The index of the part to be received • chunkSize — The size of each part that the file needs to be divided into. • fileReferenceName — The fileReferenceName returned from getTotalParts() • folderName — The folder name where the attachment is saved • isAttachmentEncrypted — Whether the attachment is encrypted or not. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	Returns a JSON object with push token.

Method name	Description and Parameter	Returns
NGMediaSplitterPlugin .deleteFile : (fileTempPath, filename, successCallback, errorCallback)	<p>This method deletes the temporary file created for transfer after transfer is successful.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • fileTempPath — The name of the temporary folder where the attachment file is saved (in case the BUC is created from the older approach) • fileName — The name of the file to be deleted • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	Return: NA

Method name	Description and Parameter	Returns
NGMediaSplitterPlugin.mergeData (tempFilePath, fileName, startIndex, partData, successCallback, errorCallback)	<p>This method merges parts of a file to make one single file.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • tempFilePath — The name of the temporary folder where the attachment file is saved (in case the BUC is created from the older approach) • fileName — The name of the file whose data is being merged • startIndex — The starting index of the part which is to be written • partData — The data to be written. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	
NGMediaSplitterPlugin.getBase64Data (tempFilePath, fileName, successCallback, errorCallback)	<p>This method returns the base64 data of a file.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • tempFilePath — The name of the temporary folder where the attachment file is saved (in case the BUC is created from the older approach) • fileName — The name of the file. • successCallback — It is a method called on the success of plug-in. • errorCallback — It is a method called in case of error caused in the plug-in. 	

- **MRZ Scanner:** User can scan passport MRZ lines to get the passport details filled in the form. There are three ways by which passport can be scanned:
 - Scan MRZ with Camera — In this passport can be scanned by placing in front of camera so that MRZ data could be read by its library.
 - Capture with Camera — In this image of passport needs to be taken by camera so that MRZ data could be read by its library.
 - Import with File Explorer — In this image of passport needs to be uploaded from file explorer so that MRZ data could be read by its library.

Following public methods are available for use in MRZ Scanner:

Method name	Description and Parameter	Returns
readText (String imagePath, String language, BOOL isCropRequired, BOOL isExtractedTextPreviewRequired)	This method is used to initialize the library and read the MRZ data from MRZ line when image of passport is either clicked by camera or taken from file explorer. Parameters: <ul style="list-style-type: none"> • imagePath(String) — path from where image is to be picked to read data. • language(String) — language with which OCR is to be performed on passport. • isCropRequired(BOOL) — Bool value which tells whether the image that is captured for performing OCR needs to be cropped for selection of OCR zone or not. • isExtractedTextPreviewRequired(BOOL) — Bool value which tells whether user wants to see the extracted data from the image on which OCR is to be performed or not. 	It returns a String value which is the recognized text got by performing OCR on the image of passport.
scanMRZ	This method is used to initialize the library and read the MRZ data from MRZ line when user want to scan the passport.	It returns a string value which is the recognized text got by performing OCR on the image of passport.

When scanning the passport with scanner, text received from scanning a particular frame gets validated with the Regex Pattern available in the MRZRegex.json then scanner stops and string gets parsed to fill data in form else the scanner keeps on scanning to get the validated data.

MRZRegex.json can be customised but it has to be in a certain format which is as below:

```

1  {
2      "mrzNoOfLines": 2,
3      "regex": [{
4          "countOfPredefinedCharacters": 5,
5          "separator": "##",
6          "lineRegex": "(P[A-Z0-9<]{1})([A-Z]{3})([A-Z0-9<]{##})"
7      },
8      {
9          "countOfPredefinedCharacters": 28,
10         "separator": "##",
11         "lineRegex": "([A-Z0-9<]{9})([0-9]{1})([A-Z]{3})([0-9]{6})([0-9]{1})([M|F|X|<]{1})([0-9]{6})([0-9]{1})([A-Z0-9<]{##})"
12     }
13 ]
14 }
15

```

Below is the details for each values in the JSON used above:

- **rzNoOfLines** — Number of MRZ data lines in passport.
- **regex** — Array of rules for each line
- **countOfPredefinedCharacters** — Each passport has some predefined characters and some characters that can vary from each passport to other passport. It is basically the user data that can vary. So while defining rules we need to define what is the number of predefined characters.
- **separator** — It is the symbol which would be replaced with a number in the line regex.
- **lineRegex** — It is the regex of the particular line in passport with ## replaced with number of characters that can vary.

External JS Loading Support

This file contains a method to load scripts dynamically. This takes a parameter as a string variable that contains the path of the script to be loaded. The path must start from scripts folder and the extension must also be added to the parameter. (*ng-custom-script-loader.js*)

Method name	Description and Parameter	Returns
NGCustomScriptLoader.loadScript(path)	This method loads the script.	

Method name	Description and Parameter	Returns
	Parameter: path — path of the script to be loaded.	

Writing solutions over NEMF

The following sections explain how to write the solution on the server and client-side over NEMF.

Server side

This section provides information on writing solutions on the server side.

Defining server interface (abstractions)

Specify the business requirements that the solution needs to meet at a high level and identify all the transactions in it. Next, identify all the entities (Objects) that are seen to be taking part in each of those transactions. [Entities are usually identified within Nouns used in the requirements text].

Identify entity relationships to create the ER model. All newly designed entities should extend from the Resource class.

Convert the identified high-level transactions into service methods. The method signature must perform I/O in terms of the entities that have been observed to participate in those transactions.

The set of method signatures, thus identified, serve as good abstractions around the business requirement of that particular nature.

All such abstractions need to be extended 'AbstractedFunctionality' interface.

```
Public interface IdentityProvider extends AbstractedFunctionality{
    Public User authenticate (User user) throws Exception;
    public User logout()throws Exception;
    ...
    ...
    ...
}
```

Writing concrete implementations

Concrete implementations need to implement the abstractions defined above.

```
public class MyLogin implements IdentityProvider {

@Override
Public User authenticate (User user)throws Exception{
    //TODO Aut0-generated method stub
    LogMe.logMe(LogMe.LOG_LEVEL_DEBUG, "Inside authenticate");
    Person toReturn = new Person();
    if (user == null) {
        throw new Exception("user cannot be null");
    }
    ...
    ...
    ...
    Return toReturn;//The user will be returned to the client as
    JSON, synchronously
}

@Override
Public void saveOrUpdateUser(User user)throws Exception{
    //Throw exceptions if it does not make sense to implement
    //an abstracted method. Exceptions will be returned to client
    //as JSON.
    throw new Exception("api not supported");
}
...
...
}
```

Pushing data to the client asynchronously

If solution writers need to push data to clients asynchronously, then in the concrete implementations, they need to set targets and then add to the device queue. Potential targets can be Users, Devices, and Role and Groups.

```
//constructing new response and route them asynchronously.
APIResponse response = new APIResponse();
response.setMode(APIResponse.MODE_ASYNCHRONOUS);
response.setResponseState(APIResponse.RESPONSE_CREATED);
```



```

response.setOrganization(response.getOrganization()); // mandatory to be set

//creating target user
User user = new User();//Or, search for an existing user
user.setUserName("lal.chandra");

//setting the targets for response
response.setTargetDevice(user);

//adding the response to deviceQueue
DeviceQueue.addToQueue(response);

```

How the solution can choose to delay a response

Sometimes, a solution is not in a position to return a response immediately (synchronously or asynchronously) because enough information to construct a response is not at hand (For example: maybe the required information is expected over a call back from a third party before a suitable response can be constructed). In this case, the solution may want to indicate to the core to inform the client that the response will be delivered later, asynchronously. Below are the steps to do it:

```

@Override
Public PaymentDetails getPaymentDetails(Order order) throws Exception{
...
..

//Indicate to the core that the response cannot be returned immediately.
Throw new DelayedResponseNotificationException("response will be delayed");
}

```

When *DelayedResponseNotificationException* is received by the core, it immediately puts the original *APIRequest* to the 'REQUEST_RESPONSE_DEFERRED' state but a response is returned to the client informing the correlation ID against which it can expect the actual response to arrive at a later time.

Later on, when the solution gets enough information (through a call back from third party), it can search for the deferred request, and check the correlation ID.

```
//Construct the response
APIResponse response=...;

// identify the user for whom the request was deferred earlier
User user=...; // search for this user in solution specific way

// find deferred requests
List
<APIRequest> matchingRequests =
GenericUtils.findResourceCarrierDeferredRequests
(user);

if (matchingRequests == null ||
    matchingRequests.size() == 0) {
    throw new Exception("no matching requests found..cant set correlationID");
}

// select the first one
APIRequest request = (APIRequest)matchingRequests.toArray()[0];
// set correlation ID and replace the old response with this new one
APIResponse earlierResponse = request.getApiResponse();
if (earlierResponse == null) {
    throw new Exception("no earlier response found matching request.. can't set
correlation ID");
}

response.setCorrelationID(earlierResponse.getCorrelationID());
request.setApiResponse(response); // discard the old response
request.saveOrUpdate();

// add the response to deviceQueue
DeviceQueue.addToQueue(response);
```

Developing solution Plug-in

This section describes how to develop a solution plug-in.

Implementing a SolutionInterface

Below is an example that illustrates how to implement the methods of the solution interface.

```

public class IdentityProviderPlugin implements SolutionInterface{
    /*
     * Core will call this method to get the list of all configurables
     * in a solution
     * Some of these Configurables may be ContextBuilders as well.
     * These will be recorded in the APIContext.
     * Whenever a configuration is loaded, it is checked whether
     * it is a ContextBuilder. If yes, it's buildContext method will
     * be called.
     * if it is savable, it is saved.
     */
    @Override
    public List
<Configurable> getConfigurables() throws Exception {
        // TODO Auto-generated method stub
        Configurable factory = new IdentityProviderFactory();
        ArrayList
        <Configurable> configurables = new ArrayList
        <Configurable>();
        configurables.add(factory);
        return configurables;
    }

    /*
     * Core will record the solution event handlers in the api context
     * and will
     * use them to deliver events generated in the core back to
     * the solutions.
     * Method will be called from APIEngine.initialize()
     */
    @Override
    public List
        <SolutionEventHandler> getSolutionEventHandlers()
        throws Exception {
        // TODO Auto-generated method stub
        ArrayList toReturn = new ArrayList();

        toReturn.add(Class.forName("com.newgen.mcap.social.facebook.concrete.Face
        bookCallBackHandler").newInstance());

        //add some more social event handlers if there are any
        return toReturn;
    }
    @Override
    public String getSolutionVersion() throws Exception {
        // TODO Auto-generated method stub
        return MobileCaptureConstants.SOLUTION_VERSION;
    }
}

```

Implementing and exposing factory

```

public class IdentityProviderFactory extends Factory {

    /*
    * Milliseconds after which the configuration should be reloaded by core.
    * If -1 is returned then the configuration will not be attempted
    * to be reloaded by core. Usually core will reload only when
    * configuration files have changed, but if this is specified
    * then this will take precedence.
    */
    // This method is not used by core at the moment
    @Override
    public int configurationRefreshInterval() throws Exception {
        // TODO Auto-generated method stub
        return 0;
    }

    /*
    * A configuration may not be intended to be saved to Datastore, if so
    * return 'false'
    */

    @Override
    public boolean isConfigurationSaveable() throws Exception {
        // TODO Auto-generated method stub
        return false;
    }

    /*
    * Whether propagation to devices required
    * If yes, then core will first try to send the Configurable
    * returned by buildConfigurableJSON method. If it returns
    * null then this object itself will be sent to the device.
    * This gives opportunity to the solution to send only the
    * required pieces of the configuration to the device.
    * If it is then core will propagate it first time as well
    * as after every change or reload of configuration.
    */
    @Override
    public boolean isPropogationToDevicesRequired() throws Exception {
        // TODO Auto-generated method stub
        return false;
    }

    /*
    * Directly checks the classpath for availability of xml file
    * and builds the Configuration object.
    */
    @Override

```

```

public Configuration loadConfigurationFromXML() throws Exception {
    // TODO Auto-generated method stub
    InputStream inputStream = null;
    Configuration toReturn = null;
    try {
        inputStream =
this.getClass().getClassLoader().getResourceAsStream(
MobileCaptureConstants.ESSENTIAL_IDENTITY_PROVIDER_CONFIG_FILE);

configuration = GenericUtils.getConfigurationFromInputStream(inputStream);
    } catch (Exception e) {
        // TODO: handle exception
        e.printStackTrace();
    } finally {
        if (inputStream != null) {
            inputStream.close();
            inputStream = null;
        }
    }

    return configuration;
}

/*
 * XML is passed as parameter
 */
@Override
public Configuration loadConfigurationFromXML(String xmlString)
    throws Exception {
    // TODO Auto-generated method stub
    InputStream inputStream = null;
    Configuration toReturn = null;
    .....
    .....
    return toReturn;
}

/*
 * The list will return a list containing User(s),
 * Devices(s) or Role(s) or Group(s). The
 * Configuration will be pushed to all
 * the listed Destinations (if
 * isPropagationToDevicesRequired() returns
 * true).
 *
 * If null or empty arrayList is returned then
 * Configuration will be sent to all.
 */
@Override
public ArrayList targetDestinations() throws Exception {

```

```

        // TODO Auto-generated method stub
        return null;
    }

    /**
     * Get the list of Saveables built from this Configuration
     * If 'isConfigurationSaveable' returns true then
     * all the Saveables that are returned from this list will be
     * Saved in data store.
     *
     * Also, the core will check if the Configurable itself is a
     * Saveable and if yes, save it too.
     */
    @Override
    public ArrayList
    <Saveable> getSaveablesBuiltFromConfiguration()
        throws Exception {
        // TODO Auto-generated method stub
        return null;
    }

    /**
     * generates the check-sum of xml file so that it can be checked
     * for changes and can be re-loaded.
     */
    @Override
    public String generateChecksumForXML() throws Exception {
        String toReturn = null;
        InputStream inputStream = null;
        try {
            inputStream = this
                .getClass()
                .getClassLoader()
                .getResourceAsStream(

MobileCaptureConstants.ESSENTIAL_IDENTITY_PROVIDER_CONFIG_FILE);

            toReturn = GenericUtils.generateSHA1Checksum(inputStream);
        } catch (Exception e) {
            e.printStackTrace();
            // Logs to be added
        } finally {
            if (inputStream != null) {
                inputStream.close();
                inputStream = null;
            }
        }
        return toReturn;
    }
}

```

Implementing and exposing SolutionEventHandlers

```
public class FacebookCallBackHandler implements SolutionEventHandler {

@Override
public String getIdentifier() throws Exception {
    // TODO Auto-generated method stub
    return "facebook";
}

@Override
public String getSolutionVersion() throws Exception {
    // TODO Auto-generated method stub
    return "1_0";
}

@Override
public void deliverEvent(CoreEvent coreEvent) throws Exception {
    LogMe.logMe(LogMe.LOG_LEVEL_DEBUG, "Inside deliverEvent");
    // TODO Auto-generated method stub
    HashMap requestParameters = (HashMap)coreEvent.getEventObjects().get(1);
    String authorizationCode = (String)requestParameters.get("code");
    .....
    .....
    // your code.....
}

}
```

Creating solutions.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Solutions organizationName="Newgen">
    <Category name="essentials">
        <SolutionInterface
implementation="com.newgen.essentials.usermanagement.plugins.IdentityProviderPlug
in"/>
        <SolutionInterface
implementation="com.newgen.essentials.templates.plugins.TemplateManagerPlugin"/>
    </Category>
</Solutions>
```

Sample organization.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Organization>
  <Name>Newgen</Name>
  <LogLevel>1</LogLevel>
  <DataMode>direct_classic</DataMode>
  <!--direct_classic/direct_chunksOnFileSystem -->
  <FileSystemPathDirectory>C:
\Users\gaurav.rohira\Desktop\gaurav_classes\testing</FileSystemPathDirectory>
  <!-- Optional element to create GeoFence at Organization level -->
  <!-- Can be overridden at BUC level (this element can be present at BUC level
config as well) -->
  <!-- Geofence breach event is fired from server and delivered to client via
DeviceSyncManager (push notifications) -->
  <!-- If MDM is configured and if it supports GeoFence setting then this data
will be sent to MDM as well -->
  <!-- If GeoFence is configured at Org level then the client mandatory needs to
enable geolocation on his device. -->
  <!-- Otherwise, app won't run --><GeoFence><Center latitude="24.67"
longitude="45.78"/><Radius value="5" units="meters"/>
  <!-- units can be meters, kilometers, miles, etc --></GeoFence>
  <!-- Data Encryption settings.. Secret key used for data enc/dec at client
device as well as server end -->
  <!-- Use Java Key Tool (or java cryptography APIs (KeyStore)) to generate a
password protected key. Point to the generated key file below -->
  <EncryptionManagement>
    <EncryptionAlgorithm>DES</EncryptionAlgorithm>
    <EncryptionKey>securityKey.file</EncryptionKey>
  </EncryptionManagement>
  <RetentionPolicy>
    <ServerSideRetentionPeriod unit="days" value="2" />
    <ClientSideRetentionPeriod unit="seconds" value="100" />
    <RetentionPolicyJobFrequency unit="seconds" value="20" />
  </RetentionPolicy>
  <!-- <CallbackURL>https://zapinserverIP/CallbackEndPoint</CallbackURL> -->
  <!-- Keep checking if configuration has changed since last check. -->
  <!-- If yes, devices may need to be synced for configured Resources unless
prevented -->
  <!-- A resource may be prevented from being synchronized to devices, if
'preventSynchronization' attribute is set to true -->
  <!-- This would primarily be used to synchronize Whitelabeling, BUC & Form
configurations to devices -->
  <!-- <ConfigurationUpdateCheck
units="minute">5</ConfigurationUpdateCheck> --><DeviceSyncConfig>
  <!--
pushMethod can be set as 'thirdParty' (default) or
'directChannel' -->
  <!--
If 'thirdParty' is set then ... -->
  <!--
if MDM configuration supports push notification (i.e. if
```



```

MobileDeviceManager.pushNotificationToSelectedDevices() does not throw Exception),
it will be used -->
    <!-- otherwise GSM (and Apple and Windows equivalent of it)-->
    <!-- notification services are used -->
    <!-- If 'directChannel' is configured then server will keep an always
on connection -->
    <!-- with each connected device through which it can push data to it. --
>
    <!-- If directChannel is configured but not possible to be created
(device offline mode) then -->
    <!-- Fall back to thirdParty mode --><Key
name="pushMethod">directChannel</Key><Key name="healthSignalDuration">5000</Key>
    <!-- milliseconds.. (heartbeat) to be used for directChannel mode only --><Key
name="connectionRefreshIntervalDuration">30</Key>
    <!-- minutes.. to be used for directChannel mode only --><Key
name="pushByPartsTriggerSize">60005</Key>
    <!-- push will begin by parts, if the trigger size is breached. --><Key
name="pushByPartsTriggerSizeUnits">kilobytes</Key>
    <!-- bytes, kilobytes, megabytes. --><Key name="pushByPartsSize">60000</Key>
    <!-- the size of JSON carried in push notification for each part push. --><Key
name="pushByPartsSizeUnits">kilobytes</Key>
    <!-- bytes, kilobytes, megabytes. --></
DeviceSyncConfig><SchedulerConfiguration>
    <!-- minimum time required on the basis of lowestHitTimestamp of all
corresponding multipart requests -->
    <Key name="MinimumWaitingTimeForMultiParts-TimeInterval">50000</Key>
    <!-- milliseconds -->
</SchedulerConfiguration>
<!-- Cloud level API config can be overridden at Organization level unless
override is prevented by organizationOverrideDisabled attribute --
><APIConfiguration><Pagination enabled="true">
<!-- False by default --><Key name="recordPerPage">10</Key><Key
name="listSessionAge">120000000</Key>
<!-- milliseconds --></Pagination><CallRestrictions><Call name="IdentityProvider|
default|authenticateDevice"><Permissions><Permission name="canMakeCall" ><Union>
<!-- --><AllowedUsers><UserReference
search=""/>
<!-- All users matching search criteria will be allowed. Wildcards allowed in
values // mandatory. Multiple are allowed.--></
AllowedUsers><AllowedGroups><GroupReference
search=""/>
<!-- All groups matching search criteria will be allowed. Wildcards allowed in
values // mandatory. Multiple are allowed.--></AllowedGroups></
Union><Intersection><AllowedRoles><RoleReference
search=""/>
<!-- All roles matching search criteria will be allowed. Wildcards allowed in
values // mandatory. Multiple are allowed.--></
AllowedRoles><AllowedDevices><DeviceReference
search=""/>
<!--// Since wildcards are allowed, you may configure a BUC to be loaded by any
random device as well with 'deviceId=' (a requirement in the specs)--></

```

```
AllowedDevices></Intersection>
<!--If none of AllowedUsers, AllowedGroups, AllowedDevices, and AllowedRoles are
present then all are allowed --></Permission></Permissions></Call></
CallRestrictions></APIConfiguration><Organization>
```

Sample cloud.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!--      i.      ???Union??? will mean that if any of the AllowedUsers OR
AllowedRoles OR ??? is met (from within Unioned portion),
            then the BUC will be allowed.
            ii.     ???Intersection??? will mean that if only when all of AllowedUsers
AND AllowedRoles AND ??? is met (from within
            Intersectioned portion), then the BUC will be allowed.
            iii. f none of ???Union??? or ???Intersection??? is specified then Union
will be assumed. Please change the configuration.
-->
<CloudConfig>
    <LogLevel>2</LogLevel>
<ProxyConfiguration proxy="true">
    <key name="proxyHost">192.168.55.118</key>
    <key name="proxyPort">8080</key>
    <key name="proxyUser">sonia.wadhwa</key>
    <key name="proxyPassword">xyz</key>
    <key name="basicAuthenticationUser">sonia.wadhwa</key>
    <key name="basicAuthenticationPassword">xyz</key>
</ProxyConfiguration>
<SchedulerConfiguration>
    <Key name="ConfigurationFilesHaveChanged-TimeDelay">12000</Key><!--
milliseconds -->
    <Key name="RespondToAsynchronousRequests-TimeDelay">10000</Key><!--
milliseconds -->
    <Key name="CompleteAsynchronousRequests-TimeDelay">5000</Key><!-- milliseconds
-->
    <Key name="MissingPartNotificationJobRequests-TimeDelay">200000</Key><!--
milliseconds -->
</SchedulerConfiguration>
<!-- API config can be overridden at Organization level unless
organizationOverrideDisabled attribute is set to true at Cloud level (See
configuration for 'saveOrUpdateOrganization' call) -->
    <APIConfiguration>
        <Pagination enabled="true"><!-- False by default -->
        <Key name="recordPerPage">10</Key>
        <Key name="listSessionAge">120000000</Key><!-- milliseconds -->
    </Pagination>

    <!-- Async multi parts request mode uses the configured part size -->
    <!-- triggerPoint is the size of JSON that triggers the client to begin async
```

```

request in parts -->
  <!-- attributes can be overridden at the individual API level -->
  <!-- If not overridden, this will be used. -->

  <!-- 'units' attribute can be bytes or kilobytes or megabytes-->

  <AsyncMultiPartRequest triggerPoint="1" triggerPartUnits="megabytes"
asyncPartSize="2000" asyncPartSizeUnits="bytes"/>

  <CallRestrictions>
  <Call name="authenticateDevice">
  <Permissions>
    <Permission name="canMakeCall"><!-- If 'name' attribute is missing then all
(canView, canUpdate...) are assumed wrt contained configuration -->
      <Union><!-- -->
        <AllowedUsers>
          <UserReference
search="field1:value1;field2:value2"/><!--All users matching search criteria will
be allowed. Wildcards allowed in values // mandatory. Multiple are allowed.-->
          </AllowedUsers>
          <AllowedGroups>
            <GroupReference
search="field1:value1;field2:value2"/><!--All groups matching search criteria will
be allowed. Wildcards allowed in values // mandatory. Multiple are allowed. -->
            </AllowedGroups>
          </Union>
          <Intersection>
            <AllowedRoles>
              <RoleReference
search="field1:value1;field2:value2"/><!--All roles matching search criteria will
be allowed. Wildcards allowed in values // mandatory. Multiple are allowed.-->
              </AllowedRoles>
              <AllowedDevices>
                <DeviceReference
search="field1;value1;field2:value2"/><!--// Since wildcards are allowed, you may
configure a BUC to be loaded by any random device as well with 'deviceId=*' (a
requirement in the specs)-->
                </AllowedDevices>
              </Intersection>
            <!--If none of AllowedUsers, AllowedGroups, AllowedDevices
and AllowedRoles are present then all are allowed -->
          </Permission>
        </Permissions>
      </Call>
      <Call name="saveOrUpdateOrganization" organizationOverrideDisabled="true">
      <Permissions/>
    </Call>
    <Call name="saveOrUpdateBusinessUseCase" organizationOverrideDisabled="true">
    <AsyncMultiPartRequest triggerPoint="3" triggerPartUnits="megabytes"
asyncPartSize="2" asyncPartSizeUnits="kilobytes"/>
    <Permissions/>

```

```

</Call>
</CallRestrictions>
</APIConfiguration>
</CloudConfig>

```

REST API web app

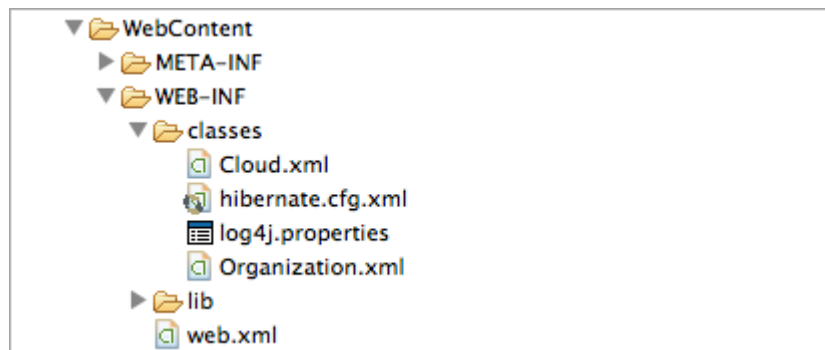
This section provides information on the web app structure and third-party libraries.

Web app structure

This section provides information on the folder structure and *Web.xml*.

Folder structure

The Lib folder contains all the third-party libraries and NEMF libraries.



Web.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://
java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="WebApp_ID" version="4.0">
    <display-name>ApiEngine</display-name>
    <welcome-file-list>

```

```

    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
</welcome-file-list>
<servlet>
    <description>APIServlet</description>
    <display-name>ApiServlet</display-name>
    <servlet-name>ApiServlet</servlet-name>
<servlet-
class>com.newgen.mcap.core.web.apiengine.concrete.ApiServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>ApiServlet</servlet-name>
    <url-pattern>/api/*</url-pattern>
</servlet-mapping>
<servlet>
    <description>CallbackServlet</description>
    <display-name>CallbackServlet</display-name>
    <servlet-name>CallbackServlet</servlet-name>
    <servlet-class>com.newgen.mcap.core.web.apiengine.concrete.CallbackServlet</
servlet-
class>
</servlet>
<servlet-mapping>
    <servlet-name>CallbackServlet</servlet-name>
    <url-pattern>/callback/*</url-pattern>
</servlet-mapping>
<servlet>
    <description>DirectChannel2</description>
    <servlet-name>DirectChannel2</servlet-name>
    <servlet-
lass>com.newgen.mcap.core.internal.longpolling.ApiDirectChannel2</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name> DirectChannel2</servlet-name>
    <url-pattern>/directchannel/*</url-pattern>
</servlet-mapping>
</web-app>

```

Third party libraries

Following is the list of third-party libraries:

Artifact name	Artifact type	License name
antlr-2.7.6.jar	Jar	-
Apnspush.jar	Jar	-
atmosphere-runtime-2.1.3.jar	Jar	Apache License, Version 2.0
bouncycastle.jar	Jar	MIT license
c3p0-0.9.5-pre8.jar	jar	-
c3p0-oracle-thin-extras-0.9.5-pre8.jar	jar	-
commons-beanutils-1.9.1.jar	Jar	Apache License, Version 2.0
commons-collections-4.0.jar	Jar	Apache License, Version 2.0
commons-jxpath-1.3.jar	Jar	Apache License, Version 2.0
commons-lang-2.4.jar	-	-
commons-logging-1.1.jar	Jar	Apache License, Version 2.0
dom4j-1.6.1.jar	Jar	-
Ejbclient.jar	-	-
gcm-server.jar	Jar	Apache License, Version 2.0
guava-16.0.1.jar	Jar	Apache License, Version 2.0
hibernate3.jar	Jar	LGPL 2.1
hibernate-jpa-2.0-api-1.0.0.Final.jar	Jar	LGPL 2.1
hsqldb.jar	Jar	BSD License based
lspack.jar	-	-
jackson-annotations-2.1.2.jar	-	-
Jackson-core-2.2.3.jar	Jar	Apache License, Version 2.0
jackson-databind-2.1.4.jar	-	-
javassist-3.12.0.GA.jar	Jar	LGPL 2.1
Jdts.jar	-	-
json-simple-1.1.1.jar	-	-
jsonSanitizer.jar	-	-

Artifact name	Artifact type	License name
Jta-1.1.jar	Jar	-
kryo.jar	Jar	BSD 3-Clause License
Log4j-1.2.14.jar	Jar	Apache License, Version 2.0
mchange-commons-java-0.2.7.jar	-	-
Minlog-1.2.jar	Jar	BSD 3-Clause License
Ngejbcallbackbroker.jar	-	-
Niplj.jar	-	-
Nsms.jar	-	-
Objenesis-1.2.jar	Jar	Apache License, Version 2.0
odweb.jar	-	-
ojdbc14.jar	Jar	-
omnishared.jar	-	-
protobuf-java-2.4.1.jar	-	-
quartz-all-1.6.6.jar	Jar	Apache License, Version 2.0
reflectasm-1.09-shaded.jar	Jar	BSD 3-Clause License
reflections-0.9.9-RC1.jar	Jar	Other Open Source
slf4j-api-1.6.1.jar	Jar	MIT license
bouncycastle.jar	Jar	MIT license
Sqljdbc4.jar	-	-
uidai-auth-client-1.6.jar	Jar	AUA License, ASA License
uidai-auth-proto-model-1.6.jar	Jar	AUA License, ASA License
uidai-auth-xsd-model-1.6.jar	Jar	AUA License, ASA License
uidai-biometric-integration-api-1.6.jar	Jar	AUA License, ASA License
uidai-sample-gui-app-1.6.jar	Jar	AUA License, ASA License

Client side

Client-side can be done for both Desktops as well as Mobile devices.

! The desktop and mobile client side will be browser-based.

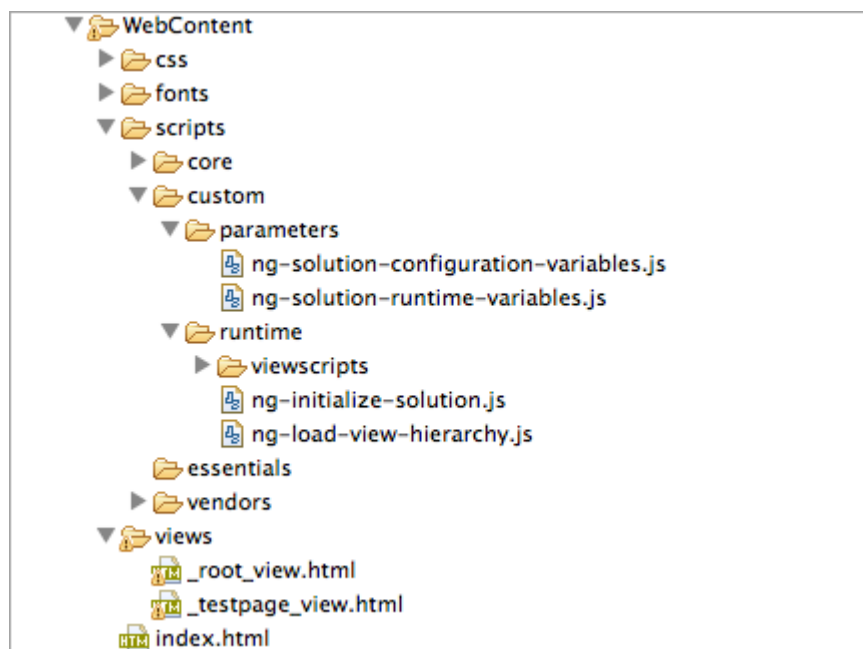
The development folder structures are slightly different for both differing only at the top levels. The part where developers need to code is essentially the same. Developers who wish to develop desktop browser-based apps will use *NGOpenWebDesktopBase4.0* as the template project while mobile-based development will require them to use *NGOpenWebPhonegapBase4.0* as a template.

Developer will typically code within */scripts/custom/* folders

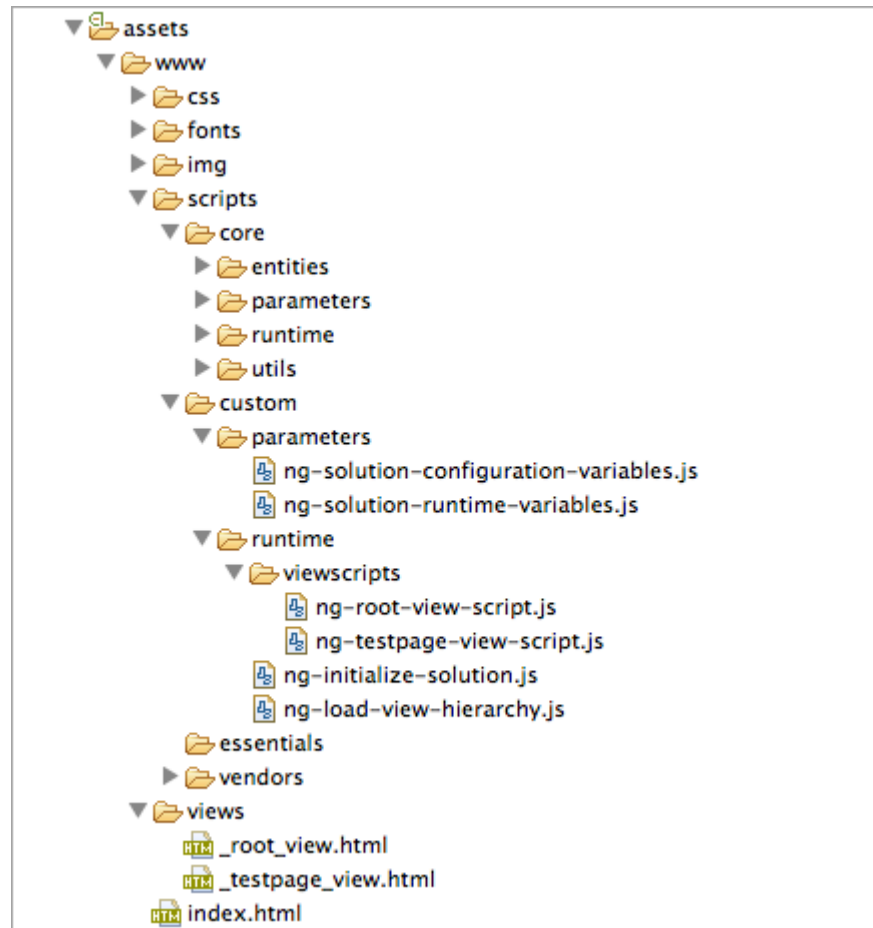
The folder structures for both are provided in the following sections:

- [NGOpenWebDesktopBase4.0 folder structure](#)
- [NGOpenWebPhonegapBase4.0 folder structure](#)

NGOpenWebDesktopBase4.0 folder structure



NGOpenWebPhonegapBase4.0 folder structure



Client custom code overview

Usually, the developer develops a View hierarchy. In this hierarchy, each view has an associated model. Let's say the name of the view is 'root', then it requires the developer to code its model called 'ng-root-view-script.js' as well as coding its HTML view '_root_view.html' [See these files in the folder structures above].

In addition to this, the configuration and runtime parameters like API endpoint details, and more can be overridden in JS configuration files located in `../custom/parameters/` folder.



Username

Password

[Reset password](#)

LOGIN

Implementing a Model ['ng-login-view-script.js']

```

/**
 * File defines the Root View Loaders and Event Handlers
 */

if (typeof (jQuery) !== 'function') {
    if (typeof (console) !== 'undefined' && console)
        console.info("Fatal Error : jQuery is not installed yet!");
    throw "jQuery not defined yet";
}

if (typeof (ngclientframework) === 'undefined')
    ngclientframework = {};
if (typeof (ngclientframework.views) === 'undefined')
    ngclientframework.views = {};
if (typeof (ngclientframework.views.login) === 'undefined')
    ngclientframework.views.login = {
        modelBuilder : function(viewObject) {

            $(document).ready(function () {
                require.undef("scripts/custom/runtime/ng-on-view-load");
                requirejs(["scripts/custom/runtime/ng-on-view-load"], function
(
util) {
            });

            $("#forgot_password").click($.proxy(function(event) {

                $NGRootView.showView('forgotPassword');
            }, viewObject));

            $("#create_profile").click($.proxy(function(event) {

                $NGRootView.showView('createProfile');
            }, viewObject));

            $("#login").click($.proxy(function(event) {
                var tempHTML;
                $(".errorLabel").remove();

                if($('#username').val() === '')
                {
                    tempHTML = "
<label class='errorLabel error-alert error-alert-login'>{{Username cannot be
blank}}</label>";

                    tempHTML = Mustache.render(tempHTML,
$NGInternationalizationDataInstance[viewObject.viewName]);

```

```

        $('#username').after(tempHTML);
    }

    if ($('#password').val()=='')
    {
        tempHTML = "
<label class='errorLabel error-alert error-alert-login' >{{Password cannot be
blank}}</label>";
        tempHTML = Mustache.render(tempHTML,
$NGInternationalizationDataInstance[viewObject.viewName]);
        $('#password').after(tempHTML);
    }

    if($(".errorLabel").length>0){
        if($NGBrowserTestingMode){
        }else{
            return;
        }
    }

    if($NGBrowserTestingMode){
        if($NGFastResumeEnabled){

if (NGCommonUtils.checkValue(NGAppDataUtils.getViewToBeResumed())){

            NGDeviceEventManager.onResume();

$NGRootView.showView(NGAppDataUtils.getViewToBeResumed());
        }else{
            $NGRootView.showView('dashboard');
        }
    }else{
        $NGRootView.showView('dashboard');
    }
    return;
}

    if(!$NGOfflineMode){

        spinner.showSpinner($NGDataUtilsImpl.getTextValue("Authenticating", "login"),
$NGDataUtilsImpl.getTextValue("Please Wait", "commonUtilities"), true);
        var self = this;
        var responseUser = authenticateUser(ngUser);

        if(responseUser){

            $NGjStorage.set($NGLoggedInUserID+"_offlineLoginAttempts",0);

var key = getSecretKey();
if(key){
var bucList = getBUCTemplateList(responseUser);
if(bucList){

```

```

getMasterListJson(function(status){
if(status){
if($NGFastResumeEnabled){
if(NGCommonUtils.checkValue(NGAppDataUtils.getViewToBeResumed())){
spinner.hideSpinner();
NGDeviceEventManager.onResume();

$NGRootView.showView(NGAppDataUtils.getViewToBeResumed());
}else{
spinner.hideSpinner();

$NGRootView.showView('dashboard');
}

}else{
spinner.hideSpinner();
$NGRootView.showView('dashboard');
}

spinner.hideSpinner();
event.preventDefault();
return;
} else{
bootbox.alert($NGDataUtilsImpl.getTextValue("Authentication Failed", "login"));
}
});
} else{
spinner.hideSpinner();
document.getElementById("username").value = "";
document.getElementById("password").value = "";
if($NGIsSessionExpired){
$NGIsSessionExpired = false;
return;
}
bootbox.alert($NGDataUtilsImpl.getTextValue("No BUCs are allowed for the User",
"login"));
}
} else{
bootbox.alert($NGDataUtilsImpl.getTextValue("Authentication Failed", "login"));
}
}
}else{
var enteredPassword =
document.getElementById("password").value;
var enteredUser = document.getElementById("username").value;
$NGLoggedInUserPassword= enteredPassword;
$NGLoggedInUserID = enteredUser;

var numberOfOfflineAttempts =
$NGjStorage.get($NGLoggedInUserID+"_offlineLoginAttempts",0);
if(numberOfOfflineAttempts <= 2){
if(NGCommonUtils.checkValue(enteredUser) &&
NGCommonUtils.checkValue(enteredPassword)){

```

```

        var ngUser =
NGAppDataUtils.getSavedNGUser(enteredUser);

        if(ngUser && ngUser != null){
            var encryptedKey =
NGAppDataUtils.getSecretKey(enteredUser);
            var decryptedKey = null;
            try{
                decryptedKey =
CryptoJS.AES.decrypt(encryptedKey,enteredPassword,{ format:
JsonFormatter }).toString(CryptoJS.enc.Utf8);
            }catch(err){

bootbox.alert($NGDataUtilsImpl.getTextValue("Incorrect Password", "login"));
            }
            if(NGCommonUtils.checkValue(decryptedKey)){
                NGAppDataUtils.setLoginStatus(true);

if(NGCommonUtils.checkValue(NGAppDataUtils.getViewToBeResumed())){

$NGRootView.showView(NGAppDataUtils.getViewToBeResumed());
            }else{
                $NGRootView.showView('dashboard');
            }

$NGjStorage.set(enteredUser+"_offlineLoginAttempts", 0);
            return;
        }else{

bootbox.alert($NGDataUtilsImpl.getTextValue("Incorrect Password", "login"));

        $NGjStorage.set(enteredUser+"_offlineLoginAttempts",
$NGjStorage.get(enteredUser+"_offlineLoginAttempts",0)+1);

        if( $NGjStorage.get(enteredUser+"_offlineLoginAttempts") >2 ){

bootbox.alert($NGDataUtilsImpl.getTextValue("Offline mode has been deactivated
due to three invalid attempts", "login"));
            $NGOfflineMode = false;
        }
    }

    }else{

bootbox.alert($NGDataUtilsImpl.getTextValue("Unknown User", "login"));
    }

    }else{
        bootbox.alert($NGDataUtilsImpl.getTextValue("Please
fill all the fields", "login"));
    }
}
}
}

```

```

        bootbox.alert($NGDataUtilsImpl.getTextValue("Offline mode
has been deactivated due to three invalid attempts", "login"));
    }

    }, viewObject));

function authenticateUser(ngUser) {

    $NGLoggedInUserID = document.getElementById("username").value;
    $NGLoggedInUserPassword =
document.getElementById("password").value;
    var ngUser = new NGUser($NGLoggedInUserID, null, "1");
    ngUser.userName = $NGLoggedInUserID;
    ngUser.emailId = "neeraj.j@newgen.co.in";
    ngUser.password = $NGLoggedInUserPassword;
    var localNgDevice = NGAppDataUtils.getNGDevice();
    ngUser.devices.push(localNgDevice);
    NGAppDataUtils.setNGUser(ngUser);

    var identityProvider = NGIdentityProvider.instantiate("od");
    var authuser = identityProvider.authenticate(ngUser);
    if(authuser != null){
        NGAppDataUtils.setLoginStatus(true);
        return authuser;
    }elseif(authuser == "response_broken"){
        return false;
    }else{
        document.getElementById("username").value = "";
        document.getElementById("password").value = "";
        spinner.hideSpinner();
        bootbox.alert($NGDataUtilsImpl.getTextValue("Authentication
Failed", "login"));
        return false;
    }
}

function getSecretKey() {
    if(!
NGCommonUtils.checkValue(NGAppDataUtils.getSecretKey($NGLoggedInUserID))){
var secretKey = $NGConfigUtilsImpl.getEncryptionKeyFromServer();
if(secretKey == "response_broken"){
return;
        } elseif(secretKey == null){
return false;
        } else{
return true;
        }
}
}

```



```

    }
}

function getBUCTemplateList(respUser){
var templateManagerImpl = NGTemplateManager.instantiate("CORE");
    var bucArray =
templateManagerImpl.getBUCTemplateListFromLocalStore();
    if(bucArray == null){
bucArray = templateManagerImpl.getBUCTemplateListFromServer(respUser);
if(bucArray == "response_broken"){
    return;
    } elseif(bucArray == null){
returnfalse;
    } else{
returntrue;
    }
}
}

function getMasterListJson(callback){
    NGMasterManager.getMasterListJson(function(data){
        if(!NGCommonUtils.checkValue(data)){
            var masterList = new Array();
            masterList = $NGDataUtilsImpl.getMasterListFromServer();
            if(masterList === "response_broken"){
                return;
            }
            if(NGCommonUtils.checkValue(masterList)){
NGMasterManager.setMasterListJson(masterList);
                $.each(masterList, function(index, masterName){
                    var masterObject = new NGMaster();
                    masterObject.masterName = masterName;
                    var masterData =
$NGDataUtilsImpl.getMasterDataFromServer(masterObject);
                    if(masterData === "response_broken"){
                        return;
                    }

                    if(NGCommonUtils.checkValue(masterData)){

                        NGMasterManager.setMasterDataForMasterKey(masterObject.masterName,masterData);
callback(true);

                    } else{

callback(false);

                    }
                });
            } else{

```

```

callback(false);
        }
    } else{
callback(false);
        }
    });
}

},

    eventHandler : function(event, viewObject) {
    }
};

var modelBuilders = [ngclientframework.views.login.modelBuilder];
var eventHandlers = [ngclientframework.views.login.eventHandler];

```

Calling APIs from the Model

```

modelBuilder : function(viewObject) {
//alert("click add");
$("#login").click($.proxy(function(event) {
    var apiEndPointDetails = "1|4.0|IdentityProvider|default|authenticate";
    var mod = "MODE_SYNCHRONOUS";
    var imeiNo = "911304252412527";
    var anyotherDeviceUniqueId = "";
    var rdbmsInstanceId = "";

    //var organization = new NGOrganization(organizationId);
    var device = new NGDevice(imeiNo, anyotherDeviceUniqueId, rdbmsInstanceId);

// filling the user information into user
$NGLoggedInUserID = document.getElementById("username").value;
$NGLoggedInUserPassword = document.getElementById("password").value;
var user = new NGUser($NGLoggedInUserID, 1);
user.userName = $NGLoggedInUserID;
user.password = $NGLoggedInUserPassword;

//
var request = new NGAPIRequest(mod, apiEndPointDetails);
request.user = user;
request.packedResources.push(user);
request.device = device;

// call the API for authentication
var response = NGAPICallManager.callApi(request, callback);
var packedResources = JSON.stringify(response.packedResources);
if(packedResources.length > 25) {

```

```
        //alert("Authentication Successful!!!!!!");  
        event.preventDefault();  
        this.showView('testpage');  
    }  
    else {  
        alert("Authentication Fail!!!!!!");  
    }  
    }, viewObject));
```

Form formats available in NEMF

NEMF provides two types of forms – Singlepane and Multipane.

To configure a Singlepane form, follow the below steps:

1. Set the value of `$NGFormFormat = "SINGLEPANE"` in `ng-solution-configuration-variables.js`.
2. If you are configuring the form views on the server side, add an XML (as it is already implemented) for the view with view markup and model.
3. If the form views are configured on the client side, create views and view scripts, as already implemented, and load them whenever required.

Singlepane forms give the below look and feel:

4:35 PM Thu 17 Dec

Online

Account Opening

ClaimerForm

Basic Details

Applicant Details (1)

+

Application ID *

Application ID

Salutation

--Select--

Gender*

Name *

Name

Date of Birth

Date of Birth

Marital Status *

--Select--

Contact Number

Contact Number

Email Id

Email Id

Id Card Type*

--Select--

Id Card Number*

SAVE

PREVIEW

CANCEL

To configure a Multipane form, follow the below steps:

1. Set the value of `$NGFormFormat = "MULTIPANE"` in `ng-solution-configuration-variables.js`.
2. If you are configuring the form views on the server side you need to add more than one XML for one form. One XML will be of the parent and others of the sections or child sections in the form. For example, if you have 3 sections in the form, namely, Basic Details, Applicant Details, and Services Required, you will have to configure four XMLs for the parent, basic details, applicant details, and services required.
3. Similar is the case if you want to configure a form on the client side.
4. The nomenclature for the forms if configuring on the client application is as mentioned below:
 - for parent section — `_<bucName>_<formName>_parentSection_view.js` and `ng-<bucName>_<formName>_parentSection-view-script.js`
 - for 1st child section — `_<bucName>_<formName>_childSection1_view.js` and `ng-<bucName>_<formName>_childSection1-view-script.js`
 - for 2nd child section — `_<bucName>_<formName>_childSection2_view.js` and `ng-<bucName>_<formName>_childSection2-view-script.js`
 - for nth child section — `_<bucName>_<formName>_childSection<n>_view.js` and `ng-<bucName>_<formName>_childSection<n>-view-script.js`
5. The nomenclature of the XMLs, if configuring on the server side, is as mentioned below:
 - for parent section — `_<bucName>_<formName>_parentSection_4.5.xml`
 - for 1st child section — `_<bucName>_<formName>_childSection1_4.5.xml`
 - for 2nd child section — `_<bucName>_<formName>_childSection2_4.5.xml`
 - for nth child section — `_<bucName>_<formName>_childSection<n>_4.5.xml`

Multipane forms give the below look and feel:

Account Opening ClaimerForm

Applicant Details

0

Application ID *

Salutation

--Select--

Gender*

Name *

Date of Birth

Version: 626

Account Opening ClaimerForm

Service Required

ATM Cum Debit Card

ATM Cum Debit Card Details

Applicant-Id	Card-Type	Name-on-Card
	--Select-- <div></div>	
SMS Notification	<div></div>	
Email Notification	<div></div>	
Cheque Box	<div></div>	
Internet Banking	<div></div>	
Phone Banking	<div></div>	



Currently, only one Singlepane form is supported. If you want to have multiple forms in your app, then use the multipane forms only.

Creating platform specific builds for mobile platforms

Perform the below steps to create a platform-specific build:

1. Create the PhoneGap project for the selected platform.
2. Create a folder in the project with the name *asset*.
3. Create a folder *www* inside the asset folder.
4. Copy all the files and folders from web content into *www*.

Visit <https://build.phonegap.com/> for more help.