

System Design Description

AT&T [Global Mobile Sales Platform]

|  |  |
| --- | --- |
| **Version Number:** | 1.7 |
| **Draft/Final as of:** | 03/18/2015 |
| **Printed on:** | [Date] |
| **Author** | AT&T/Wipro Technologies |
| **Owner** | MetLife |

|  |  |
| --- | --- |
| **Approval Signatures** | |
| Name | Title |
| Vlad Vaynrokh | Director of Information Technology |
|  |  |

|  |  |
| --- | --- |
|  |  |

Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date** | **Version** | **Author** | **Reviewed By** | **Change Description** |
| 10/Feb/2015 | 1.1 | Chella | Komal | Initial version |
| 21/Feb/2015 | 1.2 | Chella | Komal, Kishan | Updated based on V 1.1 feedback + added additional components |
| 04/ Mar/2015 | 1.3 | Ravinder | Kishan | Flow Charts |
| 09/Mar/2015 | 1.4 | Ravinder |  | Komal Review comments |
| 13/Mar/2015 | 1.5 | Ravinder |  | Komal Review comments |
| 17/Mar/2015 | 1.6 | Ravinder | Kishan | JSONStore synchronization |
| 18/Mar/2015 | 1.7 | Ravinder | Kishan | JSONStore synchronization and Customer Table on WL Server |

Table of Contents

1 Scope 5

1.1 System Design Definition Overview 5

1.2 Intended Audience 5

1.3 Identification 5

1.4 References 5

2 System Architectural Design 7

2.1 System Architecture 7

2.2 Application Logical Architecture 8

2.3 Environment (Dev, QAT, UAT, Production) 8

2.3.1 DEV Environment details: 8

2.3.2 QA environment details: 9

2.3.3 System Hardware Environment and Dependencies 10

2.3.4 System Software Environment and Dependencies 10

2.4 System Integration 11

3 Interface Definition 12

3.1 Interface Standards 12

3.1.1 Standards for Global Mobile Sales App Webservice calls 12

3.1.2 Request Header Information 12

3.1.3 Response Error Information 14

4 Processes 15

4.1 Global Components 15

4.1.1 File Transfer 15

4.1.2 Embedded Calculator Integration 19

4.1.3 Sample Web service invocation through Worklight adapter 21

4.1.4 JSON storage library 27

4.1.5 JSON Data Model 37

4.1.6 Send PDF through Email 46

4.1.7 Capture Signature 47

4.1.8 UI configurator 51

4.1.9 Multi-language support framework 58

4.1.10 Graph Control 61

4.1.11 Documents Viewer 64

4.1.12 Video Player 65

4.1.13 Idle Time check 65

4.1.14 PDF generator 66

4.1.14.1 HTML Template with dual language for the Confirmation of Advice PDF: 70

4.1.14.2 Table: 75

4.1.15 App Authenticity 76

4.1.16 FastClick.js 77

4.1.17 Wrapper Classes 77

4.1.18 IBM Worklight Logger 81

4.1.19 Webtrends Analytics 81

4.2 Flowcharts 81

4.2.1 User Authentication 81

4.2.1.1 Multiple Users login in same device and MetLife GSA App 83

4.2.2 App Flow 83

4.2.3 Products Download 84

4.2.4 Sales Illustration flow 86

4.2.5 Financial Need Analysis flow 95

4.2.6 Screen Navigation Flow 96

5 Glossary and Acronyms 97

6 Appendix 97

# Scope

## System Design Definition Overview

A System Design Definition (SDD) is a document containing detailed information about the design of a specific customer application solution. This document will describe all aspects of the system that will be created. This SDD may include some or all of the following: Architecture Diagram, process flow diagram, third party software, content repositories, software interfaces, processing logic, error handling, security, and general design considerations for requirements specified in the System Requirements Definition (SRD).

## Intended Audience

This document is intended for the following project team members:

**Architects** – Used to articulate customer and architect expectations on how approved requirements will be implemented.

**Development Leads** – Used to create Software Design Document and develop the system.

**Developers** – Used to develop the system

## Identification

A major-minor-patch-build identification number (e.g. version w.x.y.z) will be used to track the completion of project. W refers to the major release. X refers to the minor release, Y refers to patch release, and Z refers to build. During development the build release will increment with each build to QA.

The entire system produced by this project should be referred to as the [Global Mobile Sales Platform] version *[major, minor]*.

## References

|  |  |  |  |
| --- | --- | --- | --- |
| **Title** | **Author** | **Version** | **Date** |
| MetLife- GSP SRD Final ver 5 | Mahadevan | Version 5 | 02-19-2015 |
| PX\_Calculator\_Specification.docx | HP | V1.0 | NA |
| PX Documentation | HP | NA | NA |
| ProductExpress Deployment Pkg.zip | HP | NA | NA |
| Regression Test Cases input XML and output XML | HP | NA | NA |
| ProductXpress Calculator v3.7 | HP | V3.7 | April 2013 |
| Malaysia Screen Shots PPT reduced size | MetLife | App ver 1.2.0-151 |  |
| AmMetLife\_MOS\_User\_Manual\_v1.0\_EN |  | V1.0 | 11.27.2014 |
| AT&T\_MetLife\_Wireframes | Roopesh | Sign-off pending | Sign-off pending |
| GSP\_Platform\_specs\_ForMobileDevelopment | Komal | 2.1 |  |
| AmMetLifeInputRequirementMatrix\_0813 | MetLife | NA | 08-13 |
| ATT\_MetLife\_UI\_Screens | Roopesh | Sign-off pending | Sign-off pending |
| MetLife-GSA-Fields-Validations\_Msgs\_Translation\_V1.4 | Ravinder | 1.4 | 03-17-15 |
| AAG.pdf | MetLife | NA | NA |
| JobDesc.pdf | MetLife | NA | 28 Oct 2014 |
| ILF\_Summary.pdf | MetLife | NA | NA |
| Sales Illustration 5 - AmMetLife Lifestyle.pdf | Vlad Vaynrokh |  | 17 Dec 2014 |
| Sales Illustration 6 AmMetLife Link.pdf | Vlad Vaynrokh |  | 17 Dec 2014 |
| Sales Illustration 7 AmMetLife SecureBuilder.pdf | Vlad Vaynrokh |  | 17 Dec 2014 |
| Sales Illustration 9 AmMetLife SecureGuard Plus.pdf | Vlad Vaynrokh |  | 17 Dec 2014 |
| Sales Illustration 10 Secure Wealth.pdf | Vlad Vaynrokh |  | 17 Dec 2014 |

# System Architectural Design

## System Architecture



## Application Logical Architecture



## Environment (Dev, QAT, UAT, Production)

### DEV Environment details:

Below are the MetLife Development environment details:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Server Information** | | | | | | | |
| **Server Host** | **instance01.dev.sl.metlife.com** | | | |  | |  |
| **Server Type** | Virtual Server | | | |  | |  |
| **Location** | Dallas 5 | | | |  | |  |
| **Operating System** | Redhat EL6.0-64 Minimal for VSI | | | |  | |  |
| **RAM** | 8192 MB | | | |  | |  |
| **Processor** | 4 CORE | | | |  | |  |
| **Firewall** | Installed | | | |  | |  |
| **Disk Information** | | | | | | | |
| **System Disk (OS)** | 100 GB | | | |  | |  |
| **Swap Disk** | 2 GB | | | |  | |  |
| **System Disk** | 200 GB | | | |  | |  |
| **System Disk** | 200 GB | | | |  | |  |
| **System Disk** | 200 GB | | | |  | |  |
| **NAS** | 250 GB | | | |  | | /mnt/nas/ |
| **Evault** | 900 GB | | | |  | |  |
| **iSCSl:** | 0 GB | | | |  | |  |
| **IP Information** | | | | | | | |
| **Public IP** | 108.168.211.10 |  |  | | | | |
| **Private IP** | 10.81.155.222 |  |  | | | | |
|  |  |  |  | | | | |
|  |  |  |  | | | | |
| **Credential and URL Details** | | | | | | | |
|  | **Username** | | | **Password** | | **URL** | |
| **EL** | Root | | |  | |  | |
| **Nimsoft Robot** | Administrator | | |  | |  | |
| **DB2** |  | | |  | |  | |
| **DB2 Admin User** | mdasadm1:mdasgrp1 | | |  | |  | |
| **DB2 Instance Owner** | mb2inst1:mb2iadm1 | | |  | |  | |
| **DB2 Fence Owner** | mb2fenc1:mb2fadm1 | | |  | |  | |
| **WebSphere** |  | | |  | |  | |
| **WebSphere Runtime User** | wsadmin:wsadm | | |  | |  | |
| **WebSphere Admin Console** | mwasadm | | |  | | <https://10.81.155.222:9043/ibm/console> | |
| **IBM Worklight** |  | | |  | |  | |
| **Worklight Console** | wldevuser | | |  | | <https://10.81.155.222:9443/wl/console> | |
| **IBM Application Center** |  | | |  | |  | |
| **App Center Console** | appcenteradmin | | |  | | <https://10.81.155.222:9443/appcenterconsole> | |
| appcenteroffsite | | |  | |
| **App Center Client** | appcenterdev | | |  | | <https://instance01.dev.sl.metlife.com/applicetioncenter/installers.html> | |
| appcenterqa | | |  | |
| appcenteruat | | |  | |

### QA environment details:

|  |  |
| --- | --- |
| Installer QA URL:  URL: <https://mobile.qa.sl.metlife.com/appcenterconsole/installers.html> |  |
|  |
|  |

### System Hardware Environment and Dependencies

Hardware for Dev:

Mac machine / Windows Machine

Hardware for QAT:

Devices (Android screen size above 8 inch devices)

Windows Machine

### System Software Environment and Dependencies

Softwares for Dev:

1. Eclipse-juno classic
2. Android OS-Android 4.2 and above
3. XCode 6.1 and above
4. iOS- 8 and above (Deployment Target: IOS 7 onwards)
5. Cordova-4.2.0 with IBM Worklight
6. AngularJs-1.3.13
7. MobileFirst Platform (worklight)-6.1.0.02
8. Svn for Source Control Management
9. D3 JavaScript library to create graphs
10. Worklight Server access from MetLife

Softwares for QAT:

1. Android 4.2 and above
2. IOS 7 and above
3. Software to capture the bugs
4. Worklight Server access from MetLife

## System Integration

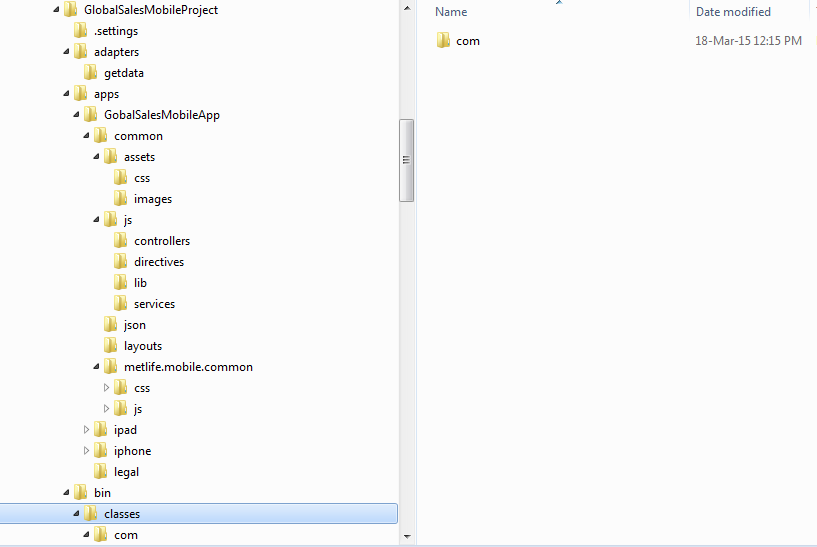
HP Embedded Calculator will be integrated with Global Mobile Sales platform to provide the flexibility to do the Sales illustration working offline.

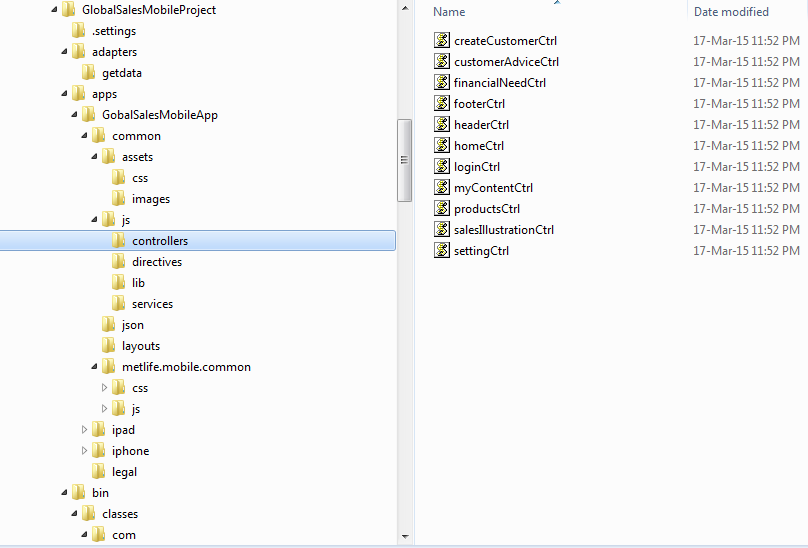
ProductExpress Deployment Pkg.zip

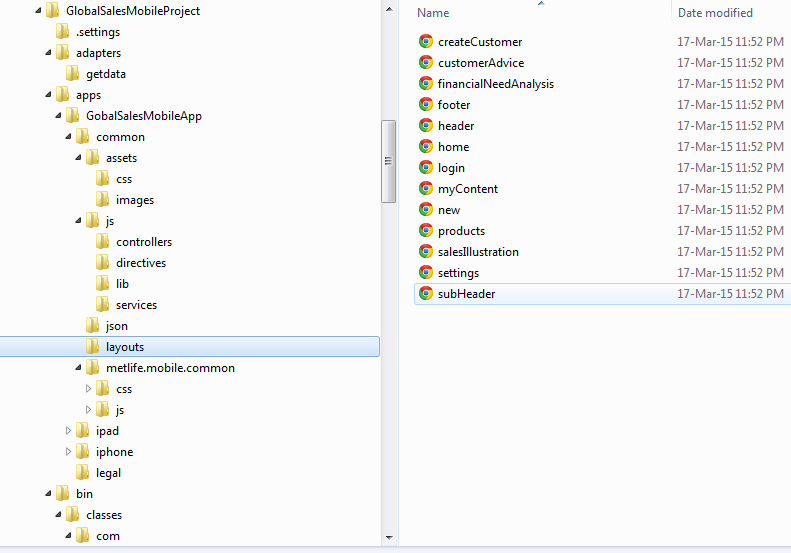
* PX\_Calculator\_Specification.docx and
* PX Test App will be the reference for integrating the Embedded Calculator in this app.

### App Directory Structure

Below directory structure will be followed while developing MetLife Global Sales Hybrid App.







# Interface Definition

## Interface Standards

### Standards for Global Mobile Sales App Webservice calls

Following are the standard list of parameters that must be included in all Webservice calls.

1. Header information
   1. Must be included in all Webservice requests
2. Footer information
   1. Must be included in all Webservices response
3. Exception to above rule
   1. It will be clearly marked in the corresponding Webservice detail section (Webservice Detail section overrides the parameter definition included in this section
4. No information in HTTP headers

**Note -Refer “GSP\_Platform\_specs\_ForMobileDevelopment\_v.2.0” document for more information on this.**

### Request Header Information

| **Request**  **Parameter** | **Required** | **Type** | **Request Parameter Description** | **Valid Values** |
| --- | --- | --- | --- | --- |
| tenantCode | Y | String | Tenant code details. This is always same for all requests. | Constant value set to “DC.HKG.SALES.<tenant\_Name>” |
| locale | Y | String | This represents the language in which this Webservice should send the response in | If Language is ENGLISH  locale = “en\_US”  If Language is VIETNAMESE  Locale = “vn\_VN” |
| authenticationToken | Y | String | This parameter is used to track if this user is already authenticated. This ensures that once authenticated each subsequent service request need not re-validate the user  GSP application created this Authentication token as part of response to “Authentication” service response.  The calling application is expected to manage the same and include it in the subsequent Webservice request parameter  Validity  TTL for the Token will be defined and will get reset at every Successful service call  Note: GSP is not doing session management for MOS | Randomly assigned security token |
| Guid | Y | String | This is random number generated to keep track of transaction across Front end and Back end.  Requesting application will generate and send it as part of request | Randomly assigned 16 digit HEX Number |
| userId | Y | String | User ID of the logged in user |  |
| deviceId | Y | String | Device ID which is associated with the user.  Calling application determines this and sends it as part of all service requests  Note: This information is planned to be validated against information stored in Active Directory (One to one relation b/w Device ID and User ID) |  |
| sourceType | Y | String | Indicates the origin of the request | Should always be “mobile” for MOS |

**Sample request with header information:**

{

"transaction": {

"header": {

"tenantCode": "DC.HKG.SALES.VNM",

"locale": "en\_US",

"authenticationToken": "HEXA0000123456",

"guid": "HEX0000000001111",

"userId": "admin",

"deviceId": "macid13",

"sourceType": "mobile"

},

"type": "illustrationId",

"parameters": {

"productCode": "UL",

"planCode": "ULA1"

}

}

}

@Komal: Please confirm the value for "tenantCode": "DC.HKG.SALES.VNM",

### Response Error Information

Following standard data field will be included in the error scenario response body (Standard header information will be included as defined above.

| **Response Parameter** | **Required** | **Response Parameter Description** | **Valid Value** |
| --- | --- | --- | --- |
| errorCode | N | This tag is mandatory only if responseStatus = “ERR”  Standard Format will be followed which will be as follows  **Format: ERR**MNN   * ERR – Prefix code stands for error * M – Major error code. There are five major error code –   + 1 – JSON Payload/ Information in JSON related   + 2 – Data Validation related (Mandatory etc..)   + 3 – Business Rule validation failed   + 4 – FILE attachments related   + 5 – Other unrecoverable exception * NN – Minor error code. There will be various possible value for minor code returned. * Example of error code: ERR101, ERR204 |  |
| errorMessage | N | This tag is mandatory only if responseStatus = “ERR”  Brief description of the error that caused the rest service call failure |  |

**Sample response with error information:**

{

"transaction": {

"header": {

"tenantCode": "DC.HKG.SALES.VNM",

"locale": "en\_US",

"guid": "HEX0000000001111",

"sourceType": "mobile",

"responseStatus": "ERR"

},

"type": "authenticate",

"parameters": {

"errorCode": "ERR101",

"errorMessage ": " Invalid JSON request - JSON Structure mismatch found "

}

}

}

# Processes

## Global Components

### File Transfer

* Cordova File Transfer plugin will be used to download below files:

1. Product related files (with in app sandbox, details are provided below)
   1. Product Calculator PXDPZ files (Path is provided below)
   2. Product Details JSON file
   3. Calculator Dependency files
2. Content files (will be stored in external downloads directory – not in app sand box)
   1. Videos (/videos)
   2. Power point, PDF documents and presentation documents (/references)
   3. Brochures documents (/brochures)

Source Files Path (Location on Files Server):

WebSphere Application Server Root directory/ProductXPress/<product deployment pkg and dependency files >

WebSphere Application Server Root directory/Content/Videos/

WebSphere Application Server Root directory/Content/Brochures/

WebSphere Application Server Root directory/Content/presentations/

**Destination Files Path (Location on Android Device):**

Root directory of the application's sandbox: (cordova.file.applicationStorageDirectory)

Path: /data/data/<appid>/<downloaded file structure> of Calculator dependency files

/data/data/<appid>/ ProductXPress//<product deployment pkg and dependency files >

**Destination Files Path (Location on iOS Device):**

Root directory of the application's sandbox: (cordova.file.documentsDirectory)

Path: /Library/<downloaded file structure> of Calculator dependency files

/Library/<appid>/ ProductXPress/<product deployment pkg and dependency files >

* PXDPZ file and dependency files to be stored in folders in app sandbox as mentioned above.
* To transfer any type of file from server to mobile app, FileTransfer cordova plugin will be used.

**CLIENT (Mobile Device):**

* To download the files securely from the server, authentication token or guid or device id etc can be passed in the request header like other webservice calls.
* Source and Destination needs to be passed as parameters for the FileTransfer API.
* FileTransfer API provides the way to show the progress bar during each file download.
* Onprogress property will be called with a ProgressEvent whenever a new chunk of data is transferred.

var ft = new FileTransfer();

ft.onprogress = function(progressEvent) {

if (progressEvent.lengthComputable) {

loadingStatus.setPercentage(progressEvent.loaded / progressEvent.total);

} else {

loadingStatus.increment();

}

};

* Sample code for FileTransfer:

**var** app = {

       // Application Constructor

       initialize : **function**() {

**this**.bindEvents();

       },

       // Bind Event Listeners

       // Bind any events that are required on startup. Common events are:

       // 'load', 'deviceready', 'offline', and 'online'.

       bindEvents : **function**() {

              document.addEventListener('deviceready', **this**.onDeviceReady,**false**);

       },

       onDeviceReady : **function**() {

**var** sourcePath = "http://10.207.52.24:8080/test/video.mkv";

              app.fileDownload(sourcePath);

       },

       fileDownload : **function**(path) {

**var** ft = **new** FileTransfer();

ft.download(path, "/storage/emulated/0/video.mp4",

function(entry) {

alert("success :"+entry);

}, function(err) {

alert("Error: "+JSON.stringify(err));

},false,{

headers: {

**Authorization: " MetLife"**

}

};

app.initialize();

**SERVER (Files storage):**

MetLife Global Sales Application content such as Videos, Presentations, brochures PDF file and Product Deployment package, Product Details JSON files will be in a server/machine.

A web service will be created and deployed on WebSphere application server to accept the request with below parameters like below:

* Create a webservice with the url https://int.sales.MetLife.com.<tenantname>/gsp/rest/common/downloadfile and with the below major parameters:

filesource

filedestination

**Dependencies:**

* 1. A Port to be opened for file downloads from mobile device on public internet
  2. Web Application Server to be run on machine where files are placed and listens on the above port.
  3. Files are placed in WAS root directory
  4. Deploy the web service/servlet in WAS which listens for requests.

Sample service on File Server:

|  |
| --- |
| **public** **void** doFilter(ServletRequest request, ServletResponse response, FilterChain chain) **throws** IOException, ServletException {                      HttpServletRequestWrapper req = **new** HttpServletRequestWrapper((HttpServletRequest) request);        HttpServletResponseWrapper res = **new** HttpServletResponseWrapper((HttpServletResponse) response);                        String auth=req.getHeader("Authorization");    **if**(auth.equalsIgnoreCase("MetLife")){                                                  chain.doFilter(request,response);                                  }**else**{                                                  res.getWriter().print(auth+" is unauthorized user");                                                  res.setStatus(401);                                  }                    } |

So server will get the authorization token and allows the downloadable file to the client only if the authorization token is valid else it will throw unauthorized error code to the file transfer plugin.

* Reference URL:

<https://github.com/apache/cordova-plugin-file-transfer/blob/master/doc/index.md>

### Embedded Calculator Integration

* No service based calculator will be used for Global Sales mobile app. Calculator functionality will work offline; no adapter calls will be made to calculate the premium.
* **Note -** PX DP calculator is for android native version only. Currently support is not available for IOS, so calculate is not part of IOS app.
* Include the below PX Embedded Calculator Runtime environment and other Utility jar files under libs folder in Android Native project structure.

a.FiaJNI.jar

b.pxjavaruntime.jar

c.PxUtils.jar

d.PxVal.jar

=> Total jars size is 64 KB

* Include the below native libraries under \libs\armeabi of Global Mobile sales app project structure.

a.libCalculatorJNI.so

b.libexslt.so

c.libiconv.so

d.libxml2.so

e.libxslt.so

=> Total Native size is 24 MB

* Product express folder structure in app sandbox data/data/appid/ProductXpress
* Product express folder has install/etc, product and other dependency sub directories which comes in dependency files download
* Through JavaScript, call the JAVA API- **PxCalculatorHomeJNI.instance().initialize()** to initialize the calculator through JavaScript Bridge
* Input the key file path through API- **PxCalculatorHome.importKey()**
* Load the deployment package for respective product through an API- **PxCalculatorHome.loadDeploymentPackage(deploymentPackageFile, null, null, null)**, parameter is pxdpz file path from product folder in ProductExpress
* Assign the agent entered and selected values from Plan JSON to input XML mapped JSON.
* Convert the mapped JSON object to input XML stream using **xml2json.js library** API
* Call calculator API - **PxPushCalculator. calculate(input)** with this input XML string.
* Calculator.calculate API to get the output from PX Calculator.
* Convert XML output stream to JSON object using **xml2json.js library** API
* Populate the values in Table template with the values from JSON output object
* Input this JSON output object to API in D3.js library to generate the graph (X Axis and Y Axis values are configured for each product).
* Total Size of the Jars + Native Libraries is around 24.64 MB approx. All the distributable are necessary and PX system cannot execute with anything missing.
* Enable “Write External Storage Permission” in "AndroidManifest.XML".
* Sample code for loading the deployment package using the Embedded Calculator

PxCalculatorHome calculatorHome = PxCalculatorHomeJNI.instance();

calculatorHome.initialize(getProductXpressInstallPath());

calculatorHome.importKey(productPath+"/metlifekey.xml"); //importing key file

calculatorHome.loadDeploymentPackage("TestInput/MyDeployment2\_1\_0.pxdpz", null, null, null); //loading pxdpz file

PxPushCalculator calculator = calculatorHome.getPushCalculator();

String request;

// Construct the request string

String result = calculator.calculate(request);

* Refer “PX\_Calculator\_Specification.docx” for Request and Response object specification.
* Android sample app (PxTestApp) will be wrapped as a JavaScript custom plug-in so that this can be integrated in Global mobile sales app. This wrapper will support only Android as of now.
* For security reasons, the deployment package with “.pxdpz” file will be used as this is in encrypted format. MetLife key is required for the decryption. [Decryption mechanism needs to be checked with Embedded Calculator HP team]

Note:

* Suppose if the pxdpz file format does not work out, then the security can be applied by hiding the embedded calculator pxdpz file from the app sandbox. This can be easily achieved by added a dot in front of the filename.
* **Encryption/Decryption detail for pxdpz file**

Still the class information would be required to call these methods.

===== API Details =====

  /\*\*  
     Import key for deployment package decryption.  
       
     @param filename The filename (Including path) file containing the encryption key.  
   \*/  
  public void importKey(String filename)  
    throws PxException;  
  
  /\*\*  
     Remove key for deployment package decryption.  
       
     @param keyLabel The label of the encryption key to remove.  
   \*/  
  public void removeKey(String keyLabel)  
    throws PxException;  
      
  /\*\*  
     Relabel key for deployment package decryption.  
       
     @param oldLabel The label of the encryption key to relabel.  
     @param newLabel The new label of the encryption key.  
   \*/  
  public void relabelKey(String oldLabel, String newLabel)  
    throws PxException;  
      
  /\*\*  
     List of imported encryption key labels.  
       
     @return List containing the labels of all imported encryption keys.  
   \*/  
  public ArrayList<String> keyList()  
    throws PxException;

### Sample Web service invocation through Worklight adapter

* Worklight adapter is used to invoke the sample web service which returns the static JSON data.
* Steps to create worklight adapter:
* Click on the Worklight icon  cid:8e8df0f2-ab48-4538-9602-d61d026ec74f pulldown from the toolbar and select Worklight Adapter (or)

File => New Worklight Adapter from the menu bar

* + - Enter the project name
    - Choose Adapter type like HTTP Adapter, SQL Adapter
    - Enter name of the adapter
    - Click Finish
* Created adapter will have .xml and –impl.js files wherein the web service and its associated methods need to be configured.
* Below are the adapter files for ProductList sample web service.
* PRODUCT\_LIST.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!--

Licensed Materials - Property of IBM

5725-I43 (C) Copyright IBM Corp. 2011, 2013. All Rights Reserved.

US Government Users Restricted Rights - Use, duplication or

disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

-->

<wl:adapter name=*"PRODUCT\_LIST"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:wl=*"http://www.worklight.com/integration"*

xmlns:http=*"http://www.worklight.com/integration/http"*>

<displayName>PRODUCT\_LIST</displayName>

<description>PRODUCT\_LIST</description>

<connectivity>

<connectionPolicy xsi:type=*"http:HTTPConnectionPolicyType"*>

<protocol>http</protocol>

<domain>10.207.52.16</domain>

<port>8080</port>

<!-- Following properties used by adapter's key manager for choosing specific certificate from key store

<sslCertificateAlias></sslCertificateAlias>

<sslCertificatePassword></sslCertificatePassword>

-->

</connectionPolicy>

<loadConstraints maxConcurrentConnectionsPerNode=*"2"* />

</connectivity>

<procedure name=*"getProductDeploymentPackageList"*/>

</wl:adapter>

* PRODUCT\_LIST-impl.js

**function** getProductDeploymentPackageList()

{

path = getPath();

**var** input = {

method : 'get',

returnedContentType : 'json',

path : path

};

**return** WL.Server.invokeHttp(input);

}

**function** getPath(interest)

{

**return** '/test/Testservice';

**}**

* Once the adapter is created, it can be invoked from JavaScript code.
* Sample code snippet to invoke the above sample product list adapter

**var** input = {

adapter : 'PRODUCT\_LIST',

procedure : '*getProductDeploymentPackageList*',

parameters : []

};

WL.Client.invokeProcedure(input, {

onSuccess : loadSQLQueerySuccess,

onFailure : loadSQLQueeryFailure

});

* We will get the response JSON object from web service through adapter on success callback of the previous code.
* Below is the sample JSON response returned for *getProductDeploymentPackageList* adapter method

{

"isSuccessful": true,

"responseHeaders": {

"Content-Length": "2324",

"Content-Type": "application\/json",

"Date": "Wed, 11 Feb 2015 13:23:14 GMT",

"Server": "Apache-Coyote\/1.1"

},

"responseTime": 31,

"statusCode": 200,

"statusReason": "OK",

"totalTime": 31,

"transaction": {

"header": {

"guid": "HEX0000000001111",

"locale": "en\_US",

"responseStatus": "OK",

"sourceType": "mobile",

"tenantCode": "DC.HKG.SALES.<Tanent Name>"

},

"parameters": {

"products": [

{

"productCode": "UL",

"productName": "Universal Life",

"productURI": "http:\/\/www.example.org\/UL",

"productVersion": "0.8",

"subProducts": [

{

"DeploymentPackageType": "pxdpz",

"plans": null,

"subDeploymentPackageDeploymentDate": "02-04-2014",

"subDeploymentPackageName": "AmLife Lifestyle\_0\_33\_14\_0",

"subDeploymentPackageVersion": "1.9",

"subMarketingName": "AmLifestyle",

"subProductCode": " ULLA5 ",

"subProductName": "AmMetLife Lifestyle"

},

{

" subDeploymentPackageDeploymentDate ": "02-04-2011",

"DeploymentPackageType": "pxdpz",

"plans": null,

"subDeploymentPackageName": "AmLife Link\_0\_31\_15\_0",

"subDeploymentPackageVersion": "0.61",

"subMarketingName": "AmLink",

"subProductCode": "ULRP6",

"subProductName": "AmMetLife Link"

}

]

},

{

"productCode": "EN",

"productName": "Endowment Product",

"productURI": "http:\/\/www.example.org\/TP",

"productVersion": "0.8",

"subProducts": [

{

" subDeploymentPackageDeploymentDate ": "02-04-2015",

"DeploymentPackageType": "pxdpz",

"plans": [

{

"marketingName": "Plan1",

"planCode": "ALSB1",

"planName": "AmMetLife SecureBuilder "

},

{

"marketingName": "Plan2",

"planCode": "ALSB2",

"planName": " AmMetLife SecureBuilder "

},

{

"marketingName": "Plan3",

"planCode": "ALSB3",

"planName": " AmMetLife SecureBuilder "

},

{

"marketingName": "Plan4",

"planCode": "ALSB4",

"planName": " AmMetLife SecureBuilder "

}

],

"subDeploymentPackageName": "AmLife SecureBuilder\_0\_44\_22\_0.pxdpz",

"subDeploymentPackageVersion": "0.81",

"subMarketingName": "AmMetLife SecureBuilder",

"subProductCode": "ALSB",

"subProductName": "AmMetLife SecureBuilder "

},

{

" subDeploymentPackageDeploymentDate ": "08-04-2015",

"DeploymentPackageType": "pxdpz",

"plans": null,

"subDeploymentPackageName": "AmLife SecureGuard Plus\_0\_43\_25\_0",

"subDeploymentPackageVersion": "0.85",

"subMarketingName": "AmMetLife SecureGuard Plus",

"subProductCode": "BTASGP1EPA",

"subProductName": "AmMetLife SecureGuard Plus"

},

{

" subDeploymentPackageDeploymentDate ": "04-04-2015",

"DeploymentPackageType": "pxdpz",

"plans": null,

"subDeploymentPackageName": "AmLife Secure Wealth\_0\_32\_15\_0",

"subDeploymentPackageVersion": "0.1",

"subMarketingName": "Secure Wealth",

"subProductCode": "BTAS3G1WPA",

"subProductName": "Secure Wealth"

}

]

}

],

"userId": "admin"

},

"type": "getProductList"

}

}

### JSON storage library

* JSONStore will be used as a local storage medium.
* Data fetched from server will be pushed into JSONStore so that it is accessible offline.
* Enable JSONStore with encryption feature with options.password.
* To enable the JSON Store database in your project  structure> open the apps / <your\_app> folder and double click  the  application-descriptor.xml file to open it. Highlight “Optional Features” under the Overview section->Click Add->select JSONStore feature => Click OK
* Once the JSONStore is enabled, we can add collections (tables) by the following snippet.

// JSONStore init

**var** collectionName = 'users';

// Object that defines all the collections.

**var** collections = {

// Object that defines the 'people' collection.

users : {

// Object that defines the Search Fields for the 'people'

// collection.

searchFields : {

name : 'string',

age : 'integer',

address : 'string',

}

}

};

WL.JSONStore.init(collections).then(**function**() {

alert("JSONStore Init success");

}).fail(

**function**(errorObject) {

alert("JSONStore Init fail " + "errcode:" + errorObject.err

+ " msg:" + errorObject.msg);

});

* This collection initialization should be done every time before using it for adding/querying data.
  + JSON storage will be developed as a global library to which the collection object will be passed as a parameter.
  + JSON Storage will be used for below JSON data.
    - Messages JSON
    - PDF Configurator
    - User Details for Offline support
    - Customer Details
    - Customer Fact Find Data
    - Financial Need Analysis Data
    - Illustration Data
    - Worklight Logger data
    - Recently used customers etc
* JSON storage will be used for Offline data storage (instead of SQLite database), mainly to have the Sync support from IBM WorkLight framework.
* **Synchronization steps**

Below is the flow chart for Worklight JSONStore synchronization using push method:

Detailed steps, API format and sample code snippet are provided after flowchart.



* + Initialize the collection with name of the adapter to use for synchronization push. Sample provided below.
  + When a record is changed in collection, the provided adapter is called.
  + When network is available call push method for customer collection. Sample provided below
    - Document.addEventListener(WL.Events.WORKLIGHT\_IS\_CONNECTED,connectDetected, false)
    - Document.addEventListener(WL.Events.WORKLIGHT\_IS\_DISCONNECTED, disconnectDetected, false)
  + Calling a push would automatically synchronize all the modified records/documents in the collection with the back end server based in the order of document last modification date.

OR

* + Manually call getPushRequired to get the dirty documents in an array and then invokeProcedure to send changes to the adapter.
  + Create a SQLAdapter in IBM Worklight with adapter name.
  + Create the table for below provided fields in customer table.
  + Change the database configurations in adapter.xml. Driver class is used to connect to the database from the project.
  + Write the procedures for add, retrieve, replace, remove operations.
* **PUSH method to synchronize the documents changes:**

|  |
| --- |
| @method push  @param [options] {Options or Array of Documents or Document or \_id}  onSuccess (function, default: none, deprecated)  onFailure (function, default: none, deprecated)  @return {Promise} Resolved when the operation succeeds, returns an array.  Array returned is either empty (everything worked) or full of error responses.  Rejected when there is a failure.  var collectionName = ‘customer’;  WL.JSONStore.get(collectionName).push()  .then(function (res) {  //handle success  //res is an empty array if all documents reached the server  //res is an array of error responses if some documents failed to reach the server  })  .fail(function (errorObject) {  //handle failure  }); |

* + Below is the promise returned when push is called
  + Promise
    - The success callback will be called when all the documents have been pushed. If you get an empty array it means everything was pushed, if something fails that array will contain error objects.
    - onSuccess called if it was successful or there where you records to push (check the number of records to push with the getPushRequiredfunction),
    - onFailure returns an error code.
    - (Deprecated behavior) onsuccess: The success callbacks are called once per document. If you try to push ten documents, your success callback might get called nine times and the failure callback once.

**Collection Initialization format:**

|  |
| --- |
| @method init  @static  @param collections {Object}  collectionName (string [a-z, A-Z, 0-9])  searchFields (object, default: {})  additionalSearchFields (object, default: {})  adapter (object, default: {})  adapter.name (string) - Name of the Adapter  adapter.add (string) - Name of the add procedure  adapter.remove (string) - Name of remove procedure  adapter.load (object)  adapter.load.procedure (string) - Name of the load procedure  adapter.load.params (array) - Parameters sent to the load procedure  adapter.load.key (string) - Key in the response containing objects to add  adapter.accept (function, returns: boolean) - Called after push with the response from the adapter.  adapter.timeout (integer) - Timeout for the adapter call |

**Sample code to initialize the collection with adapter:**

|  |
| --- |
| var collectionName = customer;  //Object that defines all the collections  var collections = {};  //Object that defines the ‘customer’ collection  collections[collectionName] = {};  //Object that defines the Search Fields for the 'people' collection  collections[collectionName].searchFields = {dateOfBirth: 'string', custID: string'};  //Optional Worklight Adapter integration  collections[collectionName].adapter = {  name: 'customerAdapter',  add: 'addProcedureInCustomerAdapterName',  remove: 'removeProcedureInCustomerAdapterName',  replace: 'replaceProcedureInCustomerAdapterName',  load: {  procedure: 'getCustomers',  params: [],  key: "customers"  },  accept: function (data) {  return (data.status === 200);  }  },timeout: 3000  };  //Optional options object  var options = {};  //Optional username  options.username = 'metlife';  //Optional password  options.password = '123';  //Optional local key generation flag  options.localKeyGen = false;  //Optional clear flag  options.clear = false;  WL.JSONStore.init(collections, options)  .then(function () {  //handle success  })  .fail(function (errorObject) {  //handle failure  }); |

**Adapter method on Worklight Server:**

|  |
| --- |
| var insertStatement = WL.Server.createSQLStatement("insert operation for customer table");  function addCustomerCreateFormdata() {  return WL.Server.invokeSQLStatement({  preparedStatement : insertStatement,  parameters : []  });  } |

**Below is the “customer create form” data table structure:**

* Create database “GLOBAL\_SALES” in IBM Worklight Server DB2
* Create table name T\_CUSTOMER in GLOBAL\_SALES database
* Create the SQL adapter with insert,update and remove procedures as mentioned in steps.



CREATE TABLE IF NOT EXISTS 'T\_CUSTOMER'(

DATE\_OF\_BIRTH TIMESTAMP NOT NULL

SALUTATION CHAR(20) NOT NULL

CUST\_NAME CHAR(50) NOT NULL

CUST\_ID\_TYPE VARCHAR(50) NOT NULL

CUST\_ID VARCHAR(50) NOT NULL

GENDER SMALLINT NOT NULL

MARITAL\_STATUS SMALLINT NOT NULL

CONTACT\_TYPE SMALLINT NOT NULL

CONTACT\_NUM VARCHAR(20) NOT NULL

ALT\_CONTACT\_TYPE SMALLINT

ALT\_CONTACT\_NUM VARCHAR(20)

OCCUPATION CHAR(200)

OCCUPATION\_CATEGORY CHAR(200)

OCCUPATION\_CLASS SMALLINT

CUST\_EMAIL VARCHAR(200)

SMOKING\_HABIT SMALLINT NOT NULL

EST\_ANNL\_INCOME VARCHAR(20)

CUST\_IMG\_BIG BLOB

CUST\_IMG\_SMALL BLOB

PRFL\_CRTD\_DATE TIMESTAMP NOT NULL DEFAULT 0

PRFL\_UPD\_DATE TIMESTAMP NOT NULL DEFAULT 0

AGENT\_CODE VARCHAR(20) NOT NULL DEFAULT 0

LST\_SYNC\_TM\_STAMP TIMESTAMP NOT NULL DEFAULT 0

PRIMARY KEY (DATE\_OF\_BIRTH, CUST\_ID)

)

### JSON Data Model

Below table provides the JSON Objects fields to store in JSON Store Collections.

|  |  |  |  |
| --- | --- | --- | --- |
| **Collection Name** | **Sub Object** | **Key** | **Remarks/Comments** |
| ProductList | - | productCategory | Ex - Wealth |
|  | Product | productCode | Ex- UL |
|  |  | productName | Ex- Universal Life |
|  |  | productVersion |  |
|  |  |  |  |
|  | SubProduct |  | Objects repeated in Category like Wealth, Protection |
|  |  | subProductCode | Ex-ULLA5 |
|  |  | subProductName | Ex-AmMetLife Lifestyle |
|  |  | subMarketingName | Ex- AmLifestyle |
|  |  | subproductDownloadStatus |  |
|  |  | subPercentComplete |  |
|  |  | subProductDetailsVersion |  |
|  |  | subProductDetailsUpdateddate |  |
|  |  | subProductDeploymentPkgVersion |  |
|  |  | deploymentPackageType |  |
|  |  | subProductDeploymentPkgName |  |
|  |  | subProductDeploymentPkgPath |  |
|  |  | subProductDeploymentPkgUpdateDate |  |
|  |  |  |  |
| Customer |  | dob | Unique identifier for record/document |
|  |  | customerId |
|  |  | Salutation |  |
|  |  | customerName |  |
|  |  | idType |  |
|  |  | Occupation |  |
|  |  | occupationCategory |  |
|  |  | occupationClass |  |
|  |  | Email |  |
|  |  | smokingHabit |  |
|  |  | Gender |  |
|  |  | maritalStatus |  |
|  |  | Contacttype |  |
|  |  | Contact |  |
|  |  | alternateContactType |  |
|  |  | alternateContact |  |
|  |  | estAnnualIncome |  |
|  |  | customerImageBigPath |  |
|  |  | customerImageSmallPath |  |
|  |  | profileUpdatedDate |  |
|  |  | profileCreatedDate |  |
|  |  | agentCode | Prepopulate with Login User name |
|  |  | syncUPTimeStamp |  |
|  |  |  |  |
| Settings |  | agentCode | Multiple agent records are inserted in to this collection when different users are logged in and their settings are entered. This logged in username field is prepopulated in settings screen.  agentCode is the unique identifier which is same login userName |
|  |  | agentName |  |
|  |  | email |  |
|  |  | contactNo |  |
|  |  | selectedLang |  |
|  |  | currencyType |  |
|  |  | currencyImagePath |  |
|  |  | imagePath |  |
|  |  |  |  |
| CFF |  | dob |  |
|  |  | id | Unique identifier for record/document |
|  |  | profileName |
|  |  | cffStatus |  |
|  |  | updatedDate |  |
|  |  | createdDate |  |
|  |  | customerAdviceChoice |  |
|  |  | agentDeclaration |  |
|  |  | noOfexistingPolicies |  |
|  |  | totalscore |  |
|  |  | preference |  |
|  |  | noOfFamilyMembers |  |
|  |  | noOfRecommendations |  |
|  | riskprofile |  |  |
|  |  | investForDuration |  |
|  |  | iCanAcceptRisk |  |
|  |  | tolerance |  |
|  | financialneeds |  |  |
|  |  | financialNeed |  |
|  |  | alreadyPlanned |  |
|  |  | toDiscuss |  |
|  |  | priority |  |
|  |  | remarks |  |
|  |  | comment |  |
|  | existingPolicies |  |  |
|  |  | policyOwner |  |
|  |  | lifeAssured |  |
|  |  | company |  |
|  |  | planType |  |
|  |  | deathBenefits |  |
|  |  | disabilityBenefits |  |
|  |  | criIllRider |  |
|  |  | otherBenefit |  |
|  |  | annualPremium |  |
|  |  | premiumType |  |
|  |  | frequency |  |
|  |  | startDate |  |
|  |  | maturityDate |  |
|  |  | projectedSumMaturity |  |
|  |  | affordability |  |
|  |  | protection |  |
|  |  | retirement |  |
|  |  | education |  |
|  |  | savings |  |
|  |  | investment |  |
|  |  | otherIncome |  |
|  |  | otherIncomeComment |  |
|  | FamilyMember |  |  |
|  |  | memberName |  |
|  |  | relationship |  |
|  |  | gender |  |
|  |  | dob |  |
|  |  | occupation |  |
|  |  | occupationCategory |  |
|  |  | occupationClass |  |
|  |  | contactType |  |
|  |  | contact |  |
|  |  |  |  |
|  | RecordOfAdvice | selectedIllustrations |  |
|  |  | planType |  |
|  |  | sumCovered |  |
|  |  | policyOwnerName |  |
|  |  | premium |  |
|  |  | recommendingReason |  |
|  |  | otherRecommendReasons |  |
|  |  | term |  |
|  |  | additionalCoverage |  |
|  |  | frequency |  |
|  |  | bought |  |
|  |  | actionDescription |  |
|  | DeclarationAndAck |  |  |
|  |  | customerDeclaration |  |
|  |  | customerSignImage |  |
|  |  | agentStatus |  |
|  |  | agentSignImage |  |
|  |  |  |  |
| FNA |  | dob | Unique identifier for record/document |
|  |  | id |
|  |  | designedFor |  |
|  |  | createDate |  |
|  |  | lastUpdateDate |  |
|  | incomeRA |  |  |
|  |  | annualIncome |  |
|  |  | incomePercentage |  |
|  |  | desiredIncome |  |
|  |  | inflationRate |  |
|  |  | projectionRate |  |
|  |  | yearsProtection |  |
|  |  | presentValue |  |
|  |  | capitalRequired |  |
|  |  | existingFunds |  |
|  |  | additionalCapital |  |
|  | childEducation |  |  |
|  |  | childName |  |
|  |  | projectedInflation |  |
|  |  | currentAge |  |
|  |  | uniEntryAge |  |
|  |  | yearsLeftToUni |  |
|  |  | year1 |  |
|  |  | year2 |  |
|  |  | year3 |  |
|  |  | year4 |  |
|  |  | total |  |
|  |  | existingSavings |  |
|  |  | existingSavingsYield |  |
|  |  | futureYearlySavings |  |
|  |  | futureYearlySavingsYield |  |
|  |  | savingsProjection |  |
|  |  | addlFundsRequired |  |
|  |  | invstReturnProjection |  |
|  |  | optionAYearly |  |
|  |  | optionBMonthly |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| Illustration |  | dob | Unique identifier for record/document |
|  |  | id |
|  |  | createdDate |  |
|  |  | lastUpdatedDate |  |
|  |  | yearlyPremium |  |
|  |  | halfYearlyPremium |  |
|  |  | quarterlyPremium |  |
|  |  | monthlyPremium |  |
|  |  | totalRidersPremium |  |
|  |  | fundPortfolioName |  |
|  | lifeAssured |  |  |
|  |  | salutation |  |
|  |  | lifeAssuredName |  |
|  |  | dob |  |
|  |  | gender |  |
|  |  | occupation |  |
|  |  | occupationClass |  |
|  |  | smokingHabit |  |
|  |  | isPlicyOwner |  |
|  | policyOwner |  |  |
|  |  | salutation |  |
|  |  | policyOwnerName |  |
|  |  | relationship |  |
|  |  | dob |  |
|  |  | gender |  |
|  |  | occupation |  |
|  |  | occupationClass |  |
|  |  | smokingHabit |  |
|  | selectedProduct |  |  |
|  |  | productCategory |  |
|  |  | subProductName |  |
|  |  | subProductCode |  |
|  | plan |  |  |
|  |  | planCode |  |
|  |  | planName |  |
|  |  | paymentFrequency |  |
|  |  | coverageTerm |  |
|  |  | paymentTerm |  |
|  |  | yearlyPremium |  |
|  |  | sumAssured |  |
|  |  | insuranceProtection |  |
|  |  | sustainibilityOption |  |
|  |  |  |  |
|  | rider | Code | This rider object is repeated for each selected rider |
|  |  | Name |  |
|  |  | coverageTerm |  |
|  |  | paymentTerm |  |
|  |  | sumAssured |  |
|  |  | Premium |  |
|  |  |  |  |
|  | fundType | fundName | This fundType object is repeated for each percentage entered fund type |
|  |  | fundCode |  |
|  |  | percentage |  |
|  |  |  |  |
|  | topUp | Year | This top up object is repeated for each percentage entered top up |
|  |  | amount |  |
|  |  |  |  |
|  | preferences | <TBD> | Will be defined during coding phase |
|  |  |  |  |
| Credentials |  | User name | Multiple user records are inserted in to this collection when different users are logged in. This logged in username field is prepopulated in settings screen |
|  |  | Password |  |
|  |  |  |  |
|  |  |  |  |
| GSAContent | documentID | documentType |  |
|  |  | documentName |  |
|  |  | docMarketingName |  |
|  |  | documentPath |  |
|  |  | docDeploymentDate |  |
|  |  | documentVersion |  |
|  |  | associatedProductCategory |  |
|  |  | associatedProductName |  |
|  |  | associatedProductCode |  |
|  |  | associatedSubProductName |  |
|  |  | associatedSubProductCode |  |
|  |  | associatedSubMarketingName |  |

**Collection Sample:**

|  |
| --- |
| app.service('$initCollections', function($jsonStore){          this.init= function(){                    var productList = 'productList';                  var customer = 'customer';                  var settings = 'settings';                  var cff= 'cff';                  var fna = 'fna';                  var illustration = 'illustration';                  var credentials = 'credentials';                  var myContent = 'myContent';                  var pdfID = 'pdfID';                  var collections = {};                  collections[productList] = {};                  collections[productList].searchFields = {productCode: 'string', productName: 'string', productCategory: 'string', productVersion: 'string', productImage: 'string'};                  collections[subProductObject] = {};                  collections[subProductObject].searchFields = {subProductCode: 'string', subProductName: 'string', subproductDownloadStatus: 'string', subPercentComplete: 'string', subProductDetailsVersion: 'string', subProductDetailsUpdateddate: 'string', subProductDeploymentPkgVersion: 'string', subProductDeploymentPkgName: 'string', subProductDeploymentPkgPath: 'string', subProductDeploymentPkgUpdateDate: 'string'};                  collections[customer] = {};                  collections[customer].searchFields = {dob: 'string', id: 'string', salutation: 'string', customerName: 'string', idType: 'string', occupation: 'string', occupationCategory: 'string', occupationClass: 'string', email: 'string', smokingHabit: 'string', gender: 'string', maritalStatus: 'string', contacttype: 'string', contact: 'string', alternateContactType: 'string', alternateContact: 'string', estAnnualIncome: 'string', customerImageBigPath: 'string', customerImageSmallPath: 'string', updatedDate: 'string', createdDate: 'string'};                  collections[settings] = {};                  collections[settings].searchFields = {agentCode: 'string', agentName: 'string', email: 'string', contactNo: 'string', selectedLang: 'string', currencyType: 'string', currencyImagePath: 'string', imagePath: 'string'};                  collections[cff] = {};                  collections[cff].searchFields = {dob: 'string', id: 'string', profileName: 'string', cffStatus: 'string', updatedDate: 'string', createdDate: 'string', customerAdviceChoice: 'string', agentDeclaration: 'string', noOfexistingPolicies: 'string', totalscore: 'string', preference: 'string', noOfFamilyMembers: 'string', noOfRecommendations: 'string', riskProfile:’string’};  var riskProfile = {investForDuration: 'string', iCanAcceptRisk: 'string', tolerance: 'string'};                  collections[fna] = {};                  collections[fna].searchFields = {dob: 'string', id: 'string', designedFor: 'string', createDate: 'string', lastUpdateDate: 'string', incomeRAObject:’string’, childEducationObject:’string’};                 var incomeRAObject = {annualIncome: 'string', incomePercentage: 'string', desiredIncome: 'string', inflationRate: 'string', projectionRate: 'string', yearsProtection: 'string', presentValue: 'string', capitalRequired: 'string', existingFunds: 'string', additionalCapital: 'string'};                  var childEducationObject = {childName: 'string', projectedInflation: 'string', currentAge: 'string', uniEntryAge: 'string', yearsLeftToUni: 'string', year1: 'string', year2: 'string', year3: 'string', year4: 'string', total: 'string', existingSavings: 'string', existingSavingsYield: 'string', futureYearlySavings: 'string', futureYearlySavingsYield: 'string', savingsProjection: 'string', addlFundsRequired: 'string', invstReturnProjection: 'string', optionAYearly: 'string', optionBMonthly: 'string'}; |

Reference – Please refer to [MetLife Interface Specification Document](#_References) for product list and product details JSON object.

### Send PDF through Email

Below are the steps for Email feature:

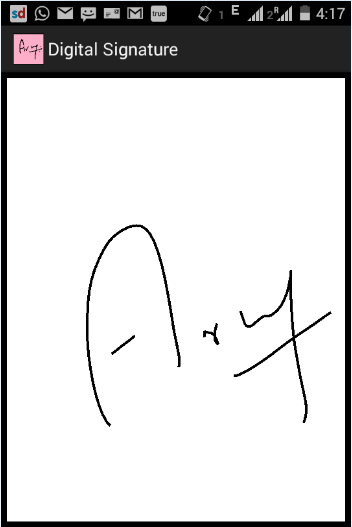
* 1. User touches on Email Icon
  2. Invoke Cordova EmailComposure plugin which in turn will call default email functionality of mobile device OS.
  3. Attach the generated PDF file
  4. MetLife Agent can add “To” and “Cc” email ids as required.
  5. Email functionality will work if the emails are configured on the device.

### Capture Signature

* Signature Capture will be developed as a library wherein user can draw his signature.
* HTML5 Canvas Drawing app will be used as a base reference to implement the Signature concept.

<https://github.com/krisrak/html5-canvas-drawing-app>

* Canvas Drawing feature is implemented as a common/reusable angular directive with the above reference. **Refer below for more information on this.**

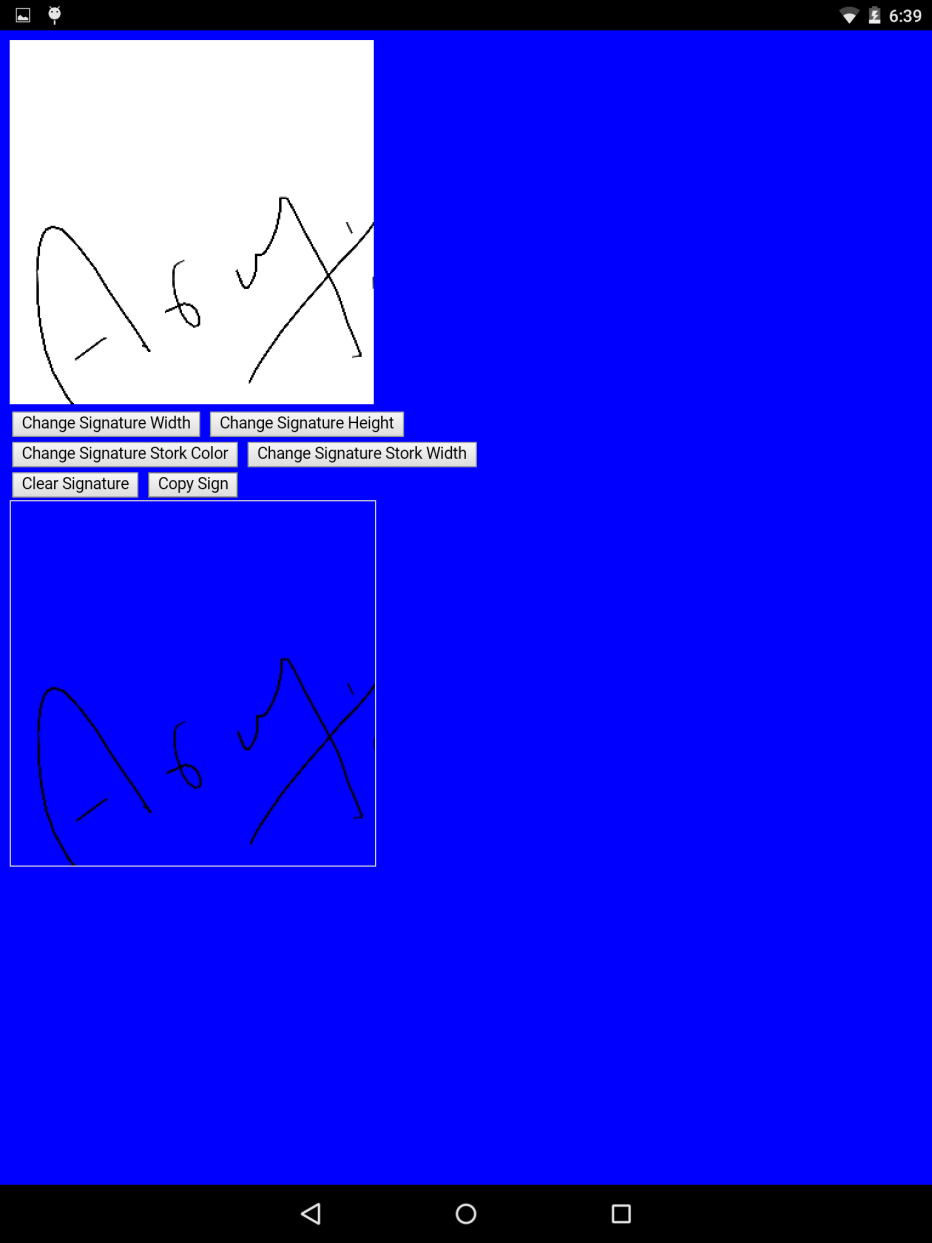


Signature is created as a global component using AngularJS directive.

**Advantages of this component:**

* Signature control width and height can be set based on the app need.
* If no width and height specified, it takes 200 pixels by default.
* The drawn signature can be fetched and shown in another canvas with transparency background. Means only the signature image will be captured and saved in a file if required.

**Output:**



Below are the steps to use signature library in any Angular mobile app.

**Steps to use Signature library**

1. Add AngularJs library files
2. Create AngularJs application module.
3. Attach signature directive to the created application module.
4. Add signature tag in html will create a component to sign over.
   * <signature SignatureWidth="300" SignatureHeight="300" signatureScope="xyz"></signature>
5. Below are the custom attributes which the signature element accepts.
   1. signatrueWidth: To set the width of the signature component (width set to 200px if signatureWidth is not available).
   2. signatureHeight: To set the height of the signature component (width set to 200px if signatureHeight is not available).
   3. signatureBgColor: To set the background color of the signature component (background color set to white if signatureBgColor is not available).
   4. signatureStrokeWidth: To set the stroke width of the signature component (stroke width set to 1px if signatureStrokeWidth is not available).
   5. signatureStrokeColor: To set the stroke color of the signature component (stroke color set to black if signatureStrokeColor is not available).
   6. signatureScope: To create a scope for signature through which the signature methods can be invoked.
6. Below are the methods provided via signatureScope object.
   1. getSignatureData(callback): returns the drawn image data of the signature to the callback function.
   * $scope.xyz.getSignatureData(function(imageData) { //xyz is the signature scope object.
   * ctx.putImageData(imageData, 0, 0); //context of temporary canvas to display the image received from getSignatureData
   * });
   1. clearSignature(): resets signature component.
   * $scope.xyz.clearSignature(); //xyz is the signature scope object.
   1. Setter and Getter methods for all the signature attributes (signatureWidth, signatureHeight, signatureBgColor, signatureStrokeWidth, signatureStrokeColor) are available via signature scope object.
   * $scope.xyz.setSignatureWidth(200); //xyz is the signature scope object.
   * $scope.xyz.getSignatureWidth();

NOTE:

1. Signature scope object should be created to invoke all the above mentioned methods of signature library.
2. Created signature scope object is available inside controller’s $scope object.

### UI configurator

* There are different scenarios where the Global Mobile Sales app need the UI configurator. This feature is mainly required to make the app features flexible so that it can support different UI for different countries.
* Scenarios:
  + 1. Module and Sub Module configurator
    2. In few places, the columns in a table need to be configured. For ex., The number of columns / data types of columns might vary depends on the product plan chosen. This needs to be defined at run time.
    3. UI needs to be adjusted based on dynamic data. For example, the number of products could be 2 or 10 in future. The number of rows in rider could be 5 or 10 or 100.
* Module and Submodule configurator needs to be defined in JSON format as follows.

Module Name: Customer Fact Find

Visibility: Y

Sub Module Name: Customer Advice Choice and declaration

Visibility: Y

Sub Module Name: Disclosure of Agent status

Visibility: Y

Sub Module Name: My financial goals process

Visibility: Y

Sub Module Name: Risk profiling process

Visibility: Y

Sub Module Name: Existing policy Details process

Visibility: N

Sub Module Name: Family details process

Visibility: N

Module Name: Financial Needs Analysis

Visibility: Y

Sub Module Name: Income Protection Process

Visibility: Y

Sub Module Name: Education Investment Need Process

Visibility: N

Module Name: Sales Illustration Process

Visibility: Y

etc

* 1. This JSON object will be downloaded via adapter after the user successful login.This configuration will be used to decide the visibility of the module or the sub module as well as the app navigations.
* **Sample to show or hide sub modules under Customer Fact Find:**

**Note:**

* “visibilitypage” field contains the tabpage name. In this case, it is set as tab1, tab2 etc.
* id field in the below JSON is used for internal logic.
* The tabpage name should be part of the id field.
* “Visibility” defines whether this tabpage should be created or not.
* The tabpages can be set as visible: yes or no based on the country. It is flexible to change this settings in the below JSON at any point of time. The app will behave as per that.
* The tabpage navigations also will change as per the below JSON settings.

**Sample JSON data (to set the visibility of sub modules)**

Note: Modules and Sub modules are not added in the below JSON data.

This is just to show how the app handles the visibility of the tabs and the navigations between the tabs as well identifying whether the tab is clicked once.

**There are totally 6 tabs in CFF, out of which tab3 is set as invisible.**

[

{

"Visibility": "yes",

"visibilitypage": "customeradvice",

"id": "footer\_tab1"

},

{

"Visibility": "yes",

"visibilitypage": "agentdisclosure",

"id": "footer\_tab2"

},

{

**"Visibility": "no",**

"visibilitypage": "financialgoals",

"id": "footer\_tab3"

},

{

"Visibility": "yes",

"visibilitypage": "riskprofiling",

"id": "footer\_tab4"

},

{

"Visibility": "yes",

"visibilitypage": "existingpolicy",

"id": "footer\_tab5"

},

{

"Visibility": "yes",

"visibilitypage": "familydetails",

"id": "footer\_tab6"

},

]

UI Configurator is made as a separate component with AngularJS module. Below are the steps to use UI Configurator.

**Steps to use UI** **Configurator:**

1. Add UI Configurator library file.
2. Add UI Configurator to the application module.
   * var app=angular.module("MetLife",['uiConfigurator']);
3. Add ui tag in html wherever the dynamic elements to be placed.
   * <ui uiScope="xyz"></ui> //xyz is the UI configurator scope object.
4. Every <ui> tag should contain separate uiScope object as an attribute.
5. This uiScope object will be available under controllers scope object ($scope).
6. createUi() is the method available in uiScope which will get respective JSON object (ui configurator object) as parameter by which the configurator renders elements.
   * $scope.xyz.createUi(object);
7. Elements created through ui configurator object will get the following key parameters as its descriptor.
   * **Mandatory**:
   1. tagName: Tag name of the element like div, input and button will be specified here.
   * **Optional:**
   1. Properties: All the attribute of the element like class, ng-bind, and ng-click will be added here.
      1. All AngularJS related attribute values should be prepended with $parent.
   * Example: ng-bind=”name” should be written as ng-bind=”$parent.name”.
   1. hide: it will Boolean value. Element will be hidden if it is true. Default is false.
   2. child: all the child elements will be specified as JSON Array.
8. **Sales Illustration Plan page**
   1. HTML Template for each Product (will optimize to use 2 templates based on the analysis of products details data and Malaysia App User Manual)
   2. Every field Label and title in the HTML template is referred as key in Language JSON file to support multiple languages.
   3. JSON Object for Plan page to store the entered and selected values is defined in JSON Data model section
   4. JSON object mapped to input XML file
   5. Build the calculation input fields in mapped JSON object from Plan page JSON object
   6. Calculation part is explained in Embedded Calculator Integration section above.
   7. **HTML Template for Plan page:**

<body>

<!-- main content -->

<div style="position:relative">

<div class="main\_body" ng-controller="MainCtrl">

<div class="main\_body\_inner sales\_plan\_main">

<h2>{{customerCreate.customerName}}</h2>

<div class="sales\_plan\_top\_right"><span>Plan</span><p>Wealth-AmMetlife Lifestyle</p></div>

<div class="sales\_plan\_inner">

<div class="sales\_plan\_head" name="login">

<span></span>

<span>{{$Language.SI.plan.totalPremium}}</span>

<label>Yearly<input type="text" /></label>

<label>Half Yearly<input type="text" /></label>

<label>Quarterly<input type="text" /></label>

<label>Monthy<input type="text" /></label>

<button>Calculate</button>

</div>

<div class="sales\_plan\_body" name="login">

<ul>

<li>

<div class="sales\_plan\_bar" ng-click="basicplantab=!basicplantab">

<label>Basic Plan</label>

<label>Basic Premium</label><input type="text" />

</div>

<div class="sales\_plan\_description " ng-hide="basicplantab">

<ul>

<li ng-repeat="basicPlan in data.basicPlan">

<label>{{basicPlan}}</label>

<input type="text" />

</li>

</ul>

</div>

</li>

<li>

<div class="sales\_plan\_bar" ng-click="riderstab=!riderstab">

<label>Riders</label>

<label>Basic Premium</label><input type="text" />

</div>

<div class="sales\_plan\_description" ng-hide="riderstab">

<table>

<thead>

<tr>

<th>Code</th>

<th>Rider Name</th>

<th>Term</th>

<th>Sum Assured</th>

<th>Premium</th>

<th></th>

</tr>

</thead>

<tbody>

<tr ng-repeat="rider in data.riders">

<td>{{rider.code}}</td>

<td>{{rider.RiderName}}<button class="info\_btn"></button></td>

<td><input type="text" />{{rider.Term}}</td>

<td><input type="text" value="RM 0.00"/>{{rider.SumAssured}}</td>

<td>{{rider.Premium}}</td>

<td><span class="{{rider.color}}"></span></td>

</tr>

</tbody>

</table>

</div>

</li>

<li>

<div class="sales\_plan\_bar" ng-click="fundtypetab=!fundtypetab">

<label>Fund Type</label>

</div>

<div class="sales\_plan\_description" ng-hide="fundtypetab">

<table>

<tbody>

<tr>

<td>Fund Portfolio</td>

<td> <select ng-model="fundportfolioselected" ng-options="option.name for option in data.fundportfolio" ng-change="fundportfoliochange()"></select></td>

</tr>

<tr>

<td>Fund Name</td>

<td>Percentage</td>

</tr>

<tr ng-repeat="fund in funds">

<td>{{fund.name}}</td>

<td><input type="text" placeholder="{{fund.percentage}}"/></td>

</tr>

<tr>

<td>Total Fund Percentage</td>

<td>100%</td>

</tr>

</tbody>

</table>

</div>

</li>

<li>

<div class="sales\_plan\_bar" ng-click="topuptab=!topuptab">

<label>Top Ups</label>

</div>

<div class="sales\_plan\_description " ng-hide="topuptab">

<table>

<tbody>

<tr >

<td ng-repeat="topup in data.topups">

<label>{{topup}}</label><br>

<input type="text"/>

</td>

</tr>

</tbody>

</table>

</div>

</li>

</ul>

</div>

</div>

1. UI tag will get replaced once the respective dynamic elements are rendered.

### Multi-language support framework

* One language JSON file will be created for each language with all the labels and titles in the app.
* Very first time, when the app is opened, translation file for English will be downloaded.
* The data will be stored in JSON storage for Offline use.
* When the user changes the language, that respective translation file will be loaded in to JSON storage and current language JSON data will be overridden.
* Agent will have language change option during login and any time after login through Settings Menu.

Language Localization is made as a separate component with AngularJS service. Below are the steps to use Language library.

* Assign all static data like Labels, titles etc. using angular expressions with reference to the downloaded language JSON file.
* By this approach all the static data can be translated to the desired language throughout the application dynamically.
* Labels and titles are considered for multi-language support framework
* App will be tested only with English Language option. Multi-Language support design framework is provided in this design document.
* Supporting of Left to Right and Right to Left script languages:
* Currently CSS is developed for Left to Right script languages such as header, footer, content grid alignment.
* Create separate CSS files with same name for RTL script and place in respective folders for CSS files. If language is LTR, render CSS from LTR folder, else render CSS from RTL folder.
* MetLife will take ownership to test the Left to Right and Right to Left Languages support.
* Below is the Sample Language JSON for customer create form and sales illustration plan sections.

|  |
| --- |
| **Language Object:**  {      **"common"**:{         **"dateOfBirth"**:"Date of Birth",       **"gender"**:"Gender",       **"male"**:"Male",       **"female"**:"Female",       **"na"**:"NA",       **"anb"**:"ANB",       **"save"**:"Save",       **"smokingHabit"**:"Smoking Habit",       **"yes"**:"Yes",       **"no"**:"No",       **"customerFactFind"**:"Customer Fact Find",       **"occupationCategory"**:"Occupation Category",       **"occupation"**:"Occupation",       **"occupationClass"**:"Occupation Class",       **"selectProduct"**:"Select Product"    },    **"createCustomer"**:{         **"customerDetails"**:{            **"customerDetails"**:"Customer Details",          **"customerName"**:"Customer Name",          **"idNumber"**:"ID Number",          **"occupation"**:"Occupation",          **"occupationClass"**:"Occupation Class",          **"email"**:"Email",          **"martialStatus"**:"Marital Status",          **"contact"**:"Contact",          **"alternativeContact"**:"Alternative Contact",          **"estimatedAnnualIncome"**:"Estimated Annual Income"       }    },    **"salesIllustration"**:{         **"basicInformation"**:{            **"basicInformation"**:"Basic Information",          **"lifeAssuredDetails"**:{               **"lifeAssuredDetails"**:"Life Assured Details",             **"lifeAssuredsName"**:"Life Assured's Name",             **"isLifeAssuredThePolicyOwner"**:"Is life assured the policy owner?"          },          **"policyOwnerDetails"**:{               **"policyOwnerDetails"**:"Policy Owner Details",             **"policyOwnerName"**:"Policy Owner Name",             **"relationship"**:"Relationship"          },          **"selectCustomer"**:"Select Customer"       },       **"plan"**:{            **"planTitle"**:"Plan:Wealth-AmMetLife Lifestyle",          **"totalPremium"**:"Total Premium",          **"yearly"**:"Yearly",          **"halfYearly"**:"Half Yearly",          **"quarterly"**:"Quarterly",          **"monthly"**:"Monthly",          **"calculate"**:"Calculate",          **"basicPlan"**:{               **"basicPlan"**:"Basic Plan",             **"basicPremium"**:"Basic Premium",             **"planCode"**:"Plan Code",             **"planName"**:"Plan Name",             **"paymentFrequency"**:"Payment Frequency",             **"sumAssured"**:"Sum Assured(RM)",             **"recommendedAmount"**:"Recommended Amount",             **"policyTerm"**:"Policy Term",             **"premiumTerm"**:"Premium Term",             **"yearlyPremium"**:"Yearly Premium",             **"insurancePortion"**:"Insurance Portion"          },          **"selectProduct"**:"Select Product",          **"brochure"**:"Brochure",          **"save"**:"Save",          **"reference"**:"Reference",          **"illustration"**:"Illustration"       }    } } |

**Steps to use Language service.**

1. Add Language service to the application module.

app.service('$Language',function($rootScope,$http){

this.changeLanguage=function(language){

$http.get('json/'+language+'.json').then(function(values) //as of now language object is stored as .json file

{

$rootScope.$Language = values.data;

});

};

});

1. setLanguage() is the method provided by $Language service which takes language option as a parameter.
2. Upon the specified language option setLanguage() method will load the respective language object to the application $rootScope so that it will be accessible throughout the application.
3. Default language should be set by calling setLanguage() in the run method of the application module.
   * var app=angular.module("MetLife",[]);
   * app.run(function($rootScope,$Language) {
   * $Language.setLanguage("Language\_EN");
   * });
4. Application Language can be changed dynamically by calling setLanguage() method.
5. Language object is created in the form of module->component.
6. These language objects are available under $Language object throughout the application.
   * <label ng-bind="$Language.sales\_illustration.basicPlan"></label>

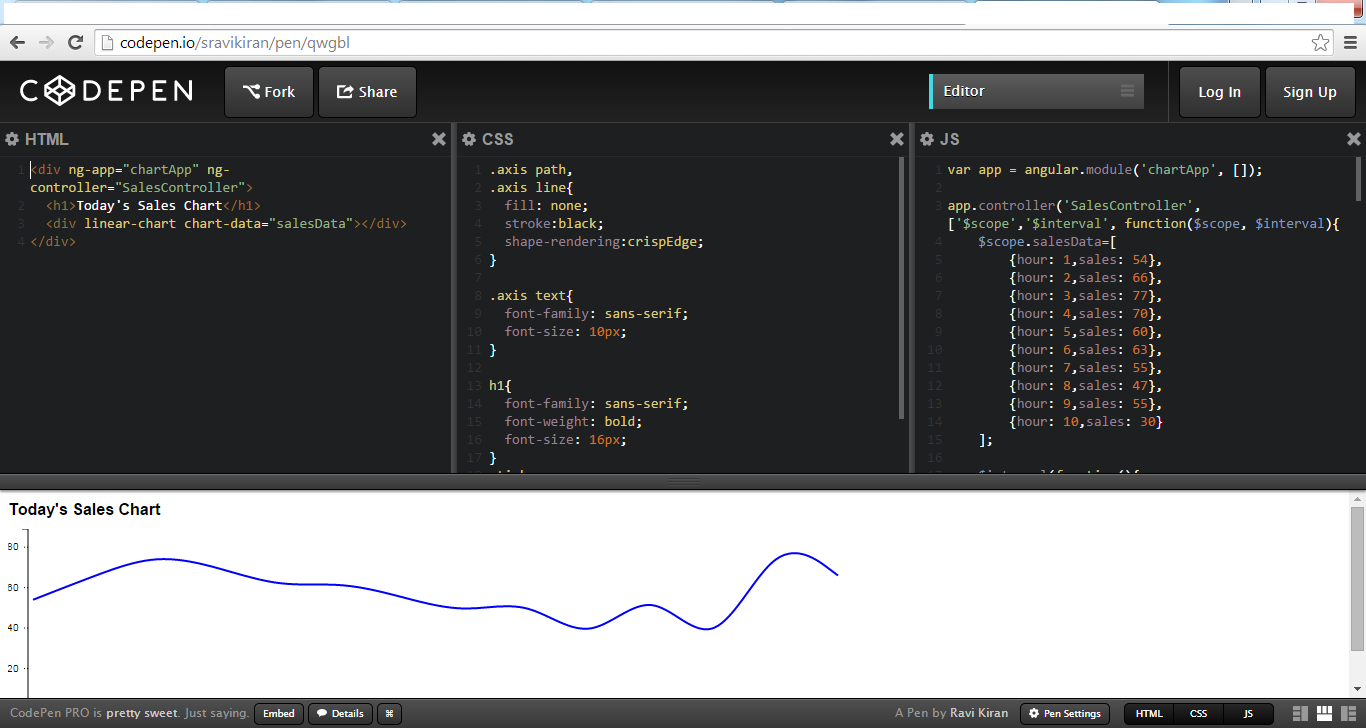
### Graph Control

* D3 JavaScript library will be used along with Angular JS to create graphs.
* Graph colors are configurable
* Calculator output XML will be converted to JSON using xml2json.js library.
* Output JSON file has table and column values.
* X axis bar graph values to be configured for each product and Y axis value will be maximum of selected columns values (need to compare last row values)
* Reference URL:

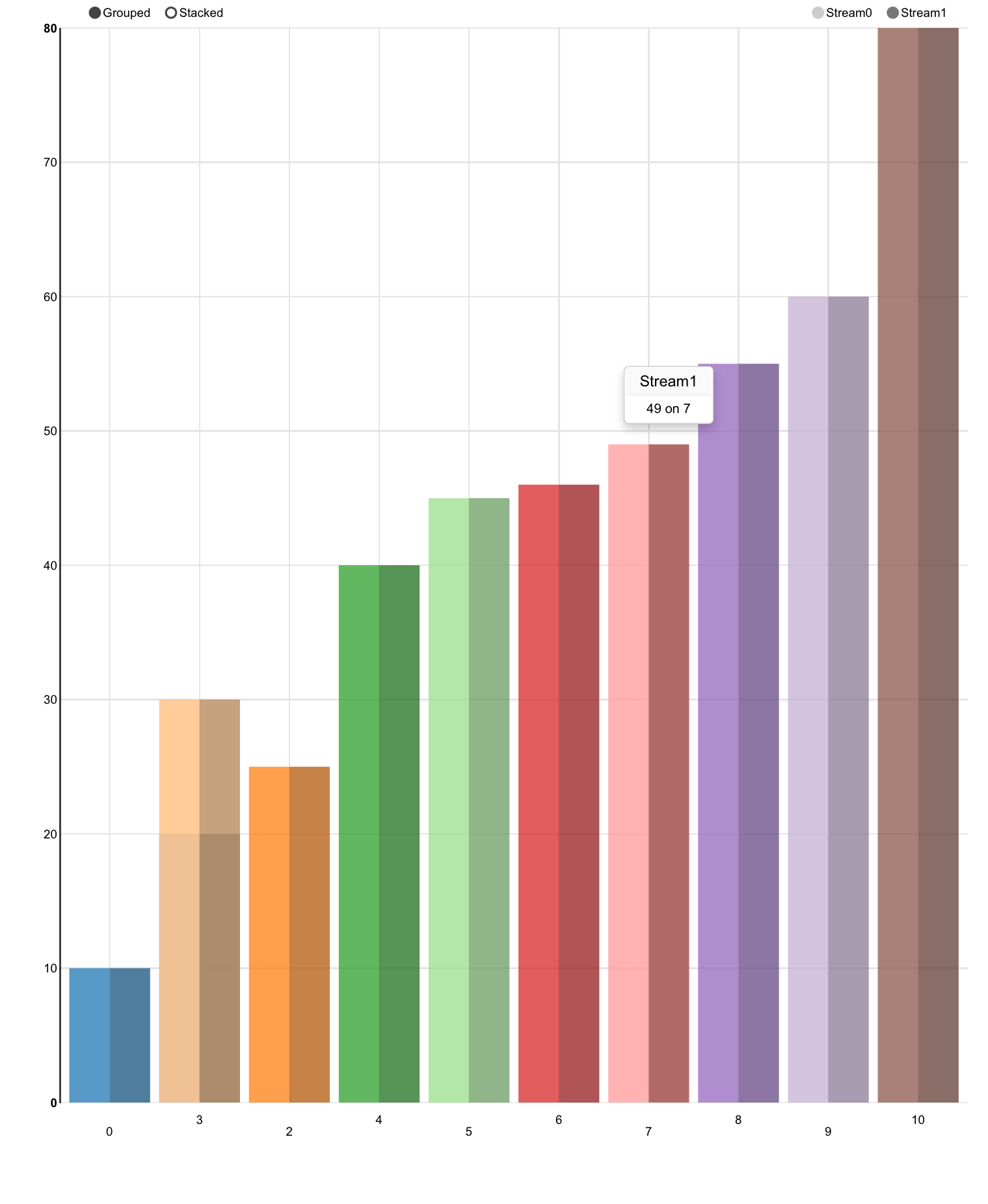
<https://github.com/mbostock/d3/wiki/Gallery>

<http://www.sitepoint.com/creating-charting-directives-using-angularjs-d3-js/>

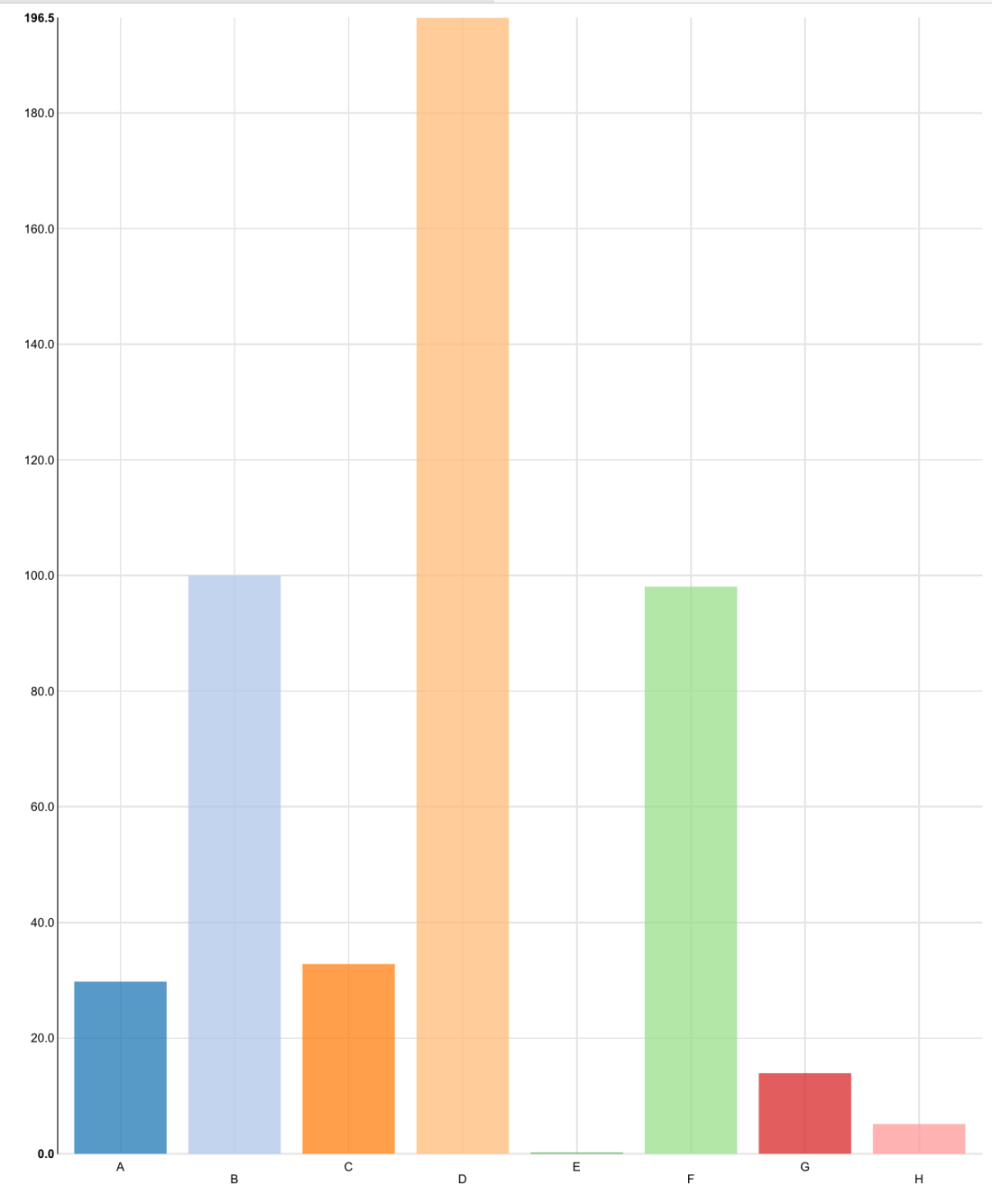
* Sample code to implement a simple graph using D3.js and Angular js.



* **Sample graph-1 created using D3.js and Angular js.**



* **Sample graph-2**



### Documents Viewer

* AngularJS PDF viewer directive will be used with pdf.js to display the PDF in angular app. <NOTE: PoC is still in progress to provide in app viewing experience>
* NOTE : PPT, XL files not able to view in in-app viewer, so to provide a consistent behavior, external native app viewer will be used to view the Microsoft files content and pdf files. This was discussed with Komal, Kishan and agreed by Saurabh.
* Dependencies – mobile device to have installed app to view Microsoft files (ex-open office) and Adobe files viewer to view the files.

### Video Player

* Ng-src will be used to play the video.
* Play Video will be created as an angular directive to play videos anywhere on the app.
* Height and Width of the video control can be set based on the screen design.
* Sample code to create Play Video directive:

**Angular Custom directive:**

app.directive('playvideo', function () {

        return {

            restrict: 'EA',

            replace: true,

            transclude: true,

            scope: {

                video: '@'

            },

            template: '<div class="playvideo">' +

**'<iframe ng-src="{{ video }}"></iframe>' +**

                      '</div>',

            link: function (scope, element, attrs) {

                var ratio = (attrs.height / attrs.width) \* 100;

            }

        };

    });

**HTML code using Playvideo directive**

<playvideo video="http://vimeo.com/112713789" height="50"width="50"></playvideo>

<playvideo video="http://www.youtube.com/embed/JMl8cQjBfqk"width="50" height="50"></playvideo>​

### Idle Time check

* ng-idle angularjs module will be used to detect and respond to app idle time.
* App idle time can be configured in AppConfig JSON object. Default idle time is 15 min.
* Below steps depicts the App idle time behavior:
  + 1. Application is in Login state
    2. App is idle for more than 15 min, redirect to Login screen.
    3. App is in background for more than 15 min and app is brought to resume/foreground, redirect to login screen.
    4. App is in background for less than 15 min and app is brought to resume/foreground, show the same screen Agent was working on.
    5. Extend the login session till download is complete. Do not redirect to Login screen when any file transfer in progress.

### PDF generator

* JSPDF library will be used to generate the PDF files from HTML view.
* Generation of PDF is used in below modules:
  + 1. Confirmation of Advice PDF with signature (Dual Language)
    2. Financial Need Analysis
       1. Income Protection (Single Language)
       2. Child Education Fund (Single Language)
    3. Sales Illustration (separate PDF for each product) (Single Language)
* **Implementation Logic:**

1. For each product, a HTML template will be prepared based on the PDF files provided.
2. For each product, a JSON object template mapped to Product Xpress calculator input XML file will be prepared.
3. User entered data stored in JSON Object and same will be used in HTML Template.
4. Basic data is populated from salesIllustration.basicInfo JSON object.
5. Selected riders are taken from salesIllustration.plan JSON
6. Selected fund types are taken from salesIllustration.plan JSON
7. Language translation files have Label as Key and value will be respective language text. Based on the selected language, respective language file will be loaded in to model and values will be put in the HTML template.
8. JSON data is applied to the HTML template including the translation.
9. The final HTML template is passed as an input to jsPDF to generate the PDF document.
10. When a PDF template has dual language, based on selection, 2 language translation JSON files to be loaded in this Controller. PDF HTML template will like below:

Sample : Label in English/ Label in second language

|  |  |
| --- | --- |
| Assured Name in first language / <Assured Name in second language> | User entered value |

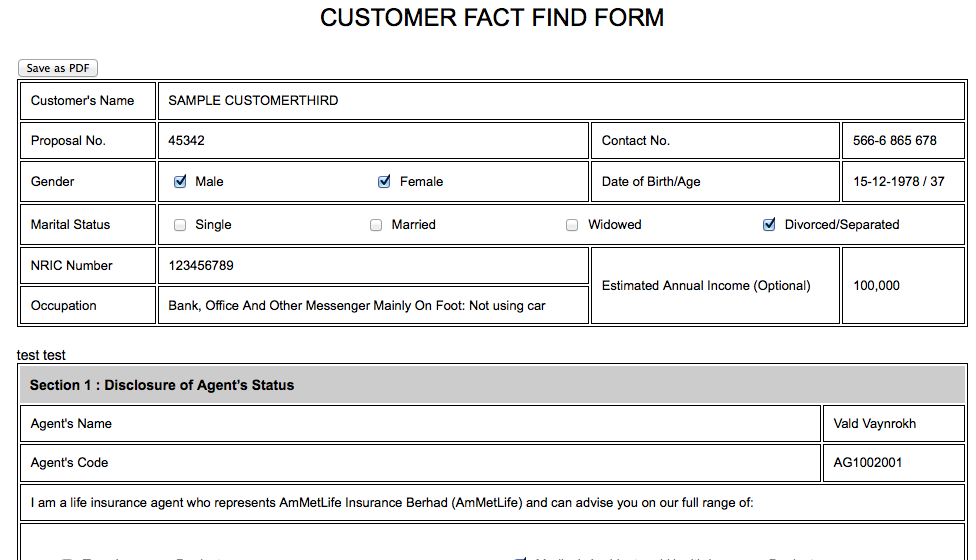
**Note:**

If there are similar layouts for different product plans, the HMTL view can be combined together. This call will be taken during the development.

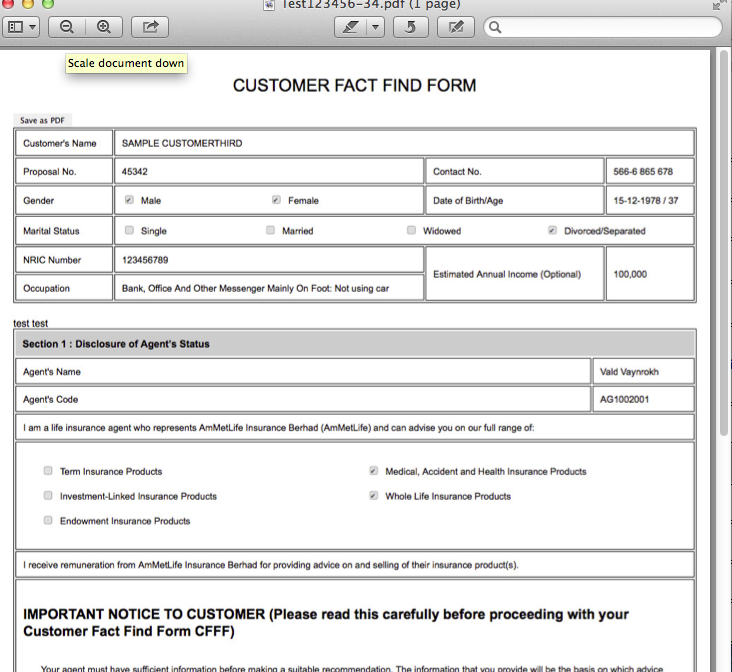
**About jsPDF library**

* JSPDF is an open-source library for generating PDF documents using nothing but JavaScript.
* It uses various functions to create various elements of PDF pages.  
    
  For example:
* **doc.text(x, y, 'string');**  
  Will print the string given in single quotes starting from the position given as point (x,y).  
    
  Instead of using a string, we can select a tag from a HTML page using JavaScript or jQuery.
* **doc.save('filename.pdf');**  
  Will save the document with the name "filename".
* **doc.addPage();**  
  Gets an extra page in the PDF file.
* **doc.setFontType('stylename');**  
  Changes the style of the font such as to italic or bold.
* **doc.setFont('fontfaceName');**  
  Provides the font face, like Times New Roman, Comic, Arial and so on.

**Sample HTML View:**



**Sample PDF View:**



**Angular JS code to save the PDF file using jsPDF:**

function saveAsPdf(){

var pdf = new jsPDF('p','pt','a4');

pdf.addHTML(document.body,function() {

pdf.save('Sample Customer Fact Find.pdf');

});

Note:

[**jsPDF**](https://mrrio.github.io/jsPDF/doc/symbols/jsPDF.html#constructor)(orientation, unit, format)

Creates new jsPDF document object instance

**jsPDF**(orientation, unit, format)

Creates new jsPDF document object instance

**Parameters:**

**orientation**

One of "portrait" or "landscape" (or shortcuts "p" (Default), "l")

**unit**

Measurement unit to be used when coordinates are specified. One of "pt" (points), "mm" (Default), "cm", "in"

**format**

One of 'a3', 'a4' (Default),'a5' ,'letter' ,'legal'

**Returns:**

*{*[*jsPDF*](https://mrrio.github.io/jsPDF/doc/symbols/jsPDF.html)*}*

#### HTML Template with dual language for the Confirmation of Advice PDF:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title>test</title>

<link href="css/reset.css" rel="stylesheet" type="text/css" />

<link href="css/style.css" rel="stylesheet" type="text/css" />

</head>

<body>

<div class="full main\_div">

<h1>{{$Language.cff.customerFactFindForm}}/ {{$SecondaryLanguage.cff.customerFactFindForm }}</h1>

<div class="main\_content">

<div class="content\_inner">

<table>

<tbody>

<tr class="border\_bottom">

<td> {{$Language.customerProfile.customerName}}/ {{$SecondaryLanguage.customerProfile.customerName}}</td>

<td colspan="4">{{data. customerName}}</td>

</tr>

<tr>

<td>{{$Language.customerProfile.proposalNo}}/ {{$SecondaryLanguage.customerProfile.proposalNo}}</td>

<td colspan="2"></td>

<td>{{$Language.customerProfile.contactNo}}/ {{$SecondaryLanguage.customerProfile.contactNo}}</td>

<td colspan="3">{{data.contactno}}</td>

</tr>

<tr class="border\_bottom">

<td>{{$Language.customerProfile.gender}}/ {{$SecondaryLanguage.customerProfile.gender}}</td>

<td colspan="2" class="two\_label">

<label><input type="checkbox" />{{$Language.customerProfile.male}}/ {{$SecondaryLanguage.customerProfile.male}}</label>

<label><input type="checkbox" />{{$Language.customerProfile.female}}/ {{$SecondaryLanguage.customerProfile.female}}</label>

</td>

<td>{{$Language.customerProfile.dob}}/ {{$SecondaryLanguage.customerProfile.dob}}</td>

<td>{{data.dob}} / {{data.age}}</td>

</tr>

<tr>

<td>{{$Language.customerProfile.maritalStatus}}/ {{$SecondaryLanguage.customerProfile.maritalStatus}}</td>

<td colspan="4" class="four\_label">

<label><input type="checkbox" /> {{$Language.customerProfile.single}}/{{$SecondaryLanguage.customerProfile.single}} </label>

<label><input type="checkbox" /> {{$Language.customerProfile.married}}/{{$SecondaryLanguage.customerProfile.married}} </label>

<label><input type="checkbox" /> {{$Language.customerProfile.divorced}}/{{$SecondaryLanguage.customerProfile.divorced}} </label>

<label><input type="checkbox" /> {{$Language.customerProfile.widowed}}/{{$SecondaryLanguage.customerProfile.widowed}}/ </label>

</td>

</tr>

<tr>

<td>{{$Language.customerProfile.nricno}}/ {{$SecondaryLanguage.customerProfile.nricno}}</td>

<td colspan="2">{{data.nricno}}</td>

<td rowspan="2">{{$Language.customerProfile.estAnnIncome}}/ {{$SecondaryLanguage.customerProfile.estAnnIncome}}</td>

<td rowspan="2">{{data.plan. estAnnIncome }}</td>

</tr>

</tr>

<tr class="border\_bottom">

<td>{{$Language.customerProfile.occupation}}/ {{$SecondaryLanguage.customerProfile.occupation}}</td>

<td colspan="2">{{data.occupation}}</td>

</tr>

</tbody>

</table>

</div>

<div class="content\_inner">

<table>

<thead>

<th colspan="5">{{$Language.pdf.section}}/ {{$SecondaryLanguage.pdf.section}} 1 : {{$Language.cff.dislosureOfAgentStatus}}/ {{$SecondaryLanguage.cff.dislosureOfAgentStatus }}</th>

</thead>

<tbody>

<tr>

<td>{{$Language.settings.agentName}}/ {{$SecondaryLanguage.settings.agentName}}</td>

<td colspan="4">Vald Vaynrokh</td>

</tr>

<tr class="border\_bottom">

<td>{{$Language.settings.agentCode}}/ {{$SecondaryLanguage.settings.agentCode}}</td>

<td colspan="4">{{data.agentCode}}</td>

</tr>

<tr class="border\_bottom">

<td colspan="5">I am a life insurance agent who represents AmMetLife Insurance Berhad (AmMetLife) and can advise you on our full range of:</td>

</tr>

<tr>

<td colspan="5">I receive remuneration from AmMetLife Insurance Berhad for providing advice on and selling of their insurance product(s).</td>

</tr>

<tr>

<td colspan="5">

<h2>IMPORTANT NOTICE TO CUSTOMER (Please read this carefully before proceeding with your Customer Fact Find Form CFFF)</h2>

<ul>

<li>Your agent must have sufficient information before making a suitable recommendation. The information that you provide will be the basis on which advice is

given.</li>

<li>If you choose not to provide all the relevant information requested, your agent may not be able to provide you with suitable advice and as a result, you may

risk making a financial commitment to a life insurance policy inappropriate to your needs.</li>

<li>Your agent is required to preserve the confidentiality of information disclosed by you and restrict the use of such information only for the purpose of

recommending life insurance product(s).</li>

<li>You must ensure that important information regarding the policy is disclosed to you and that you understand that information. Where something is not clear,

you should seek an explanation from the agent or AmMetLife.</li>

<li>Before you make a decision to purchase any life insurance policy, you must satisfy yourself that the product(s) meets your insurance needs and resources.</li>

</ul>

</td>

</tr>

</tbody>

</table>

</div>

<div class="content\_inner">

<table>

<thead>

<th colspan="5">Section 2 : Customer Advice Choice and Declaration</th>

</thead>

<tbody>

<tr>

<td><label><input type="checkbox" />Option A</label></td>

<td colspan="4">I/We wish to disclose all information requested for in this form (fill in all sections except sections 7 & 9).</td>

</tr>

<tr>

<td><label><input type="checkbox" />Option B</label></td>

<td colspan="4">I/We wish to disclose partially information requested for in this form (fill in all sections except sections 3, 7 & 9).</td>

</tr>

<tr>

<td><label><input type="checkbox" />Option C</label></td>

<td colspan="4">I/We wish to receive product information only and do not wish to disclose any information requested for in this form (fill in sections 5, 6

& 8 only).</td>

</tr>

<tr>

<td colspan="5">

<div class="signature\_main">

<div class="signature\_border"></div>

<span>Customer’s Signature</span>

</div>

<div class="signature\_main">

<div class="signature\_border"></div>

<span>Date</span>

</div>

</td>

</tr>

</tbody>

</table>

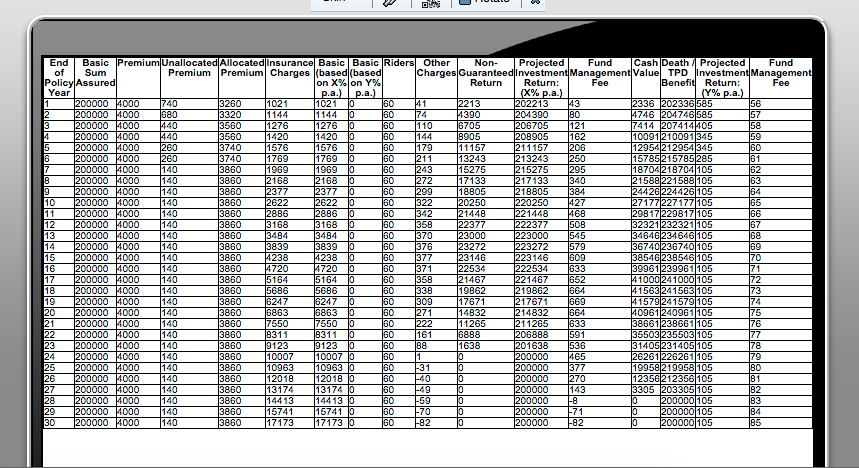
</div>

</div>

</div>

</body></html>

#### Table:



**Code snippet for Table generation using output XML from PX Calculator:**

|  |
| --- |
| function LinkPlanTableCtrl($scope, $http,$location,$rootScope,$adapterUtility,$commonUtility,$jsonStore,$compile) {  var htmldata="<table><thead><tr>";  $http.get('json/calculatoroutput-table.JSON.txt').then(function(values)  {  $scope.datatoparse=JSON.parse(JSON.stringify(values.data.CalculationOutput.Calculation.Policy.Features.Tables.SetElement));  $scope.header=$scope.datatoparse[0].TableData.Headers.SetElement;  $scope.column=$scope.datatoparse[0].TableData.Columns.SetElement;  for(var i=0;i<$scope.header.length;i++){  htmldata+='<th style="border:1px solid black;font-size:10px;">'+$scope.header[i].Name+'</th>';  }  htmldata+='</tr></thead><tbody>';  for(var r=0;r<$scope.column[0].Values.SetElement.length;r++){  htmldata+='<tr>';  for(var c=0;c<$scope.column.length;c++){  htmldata+='<td style="border:1px solid black;font-size:10px;">'+$scope.datatoparse[0].TableData.Columns.SetElement[c].Values.SetElement[r].value+'</td>';    }  htmldata+='</tr>';  }  htmldata+='</tbody></table>';  var tableelement=document.getElementById("tabledata");  tableelement.appendChild( $compile(htmldata)($scope)[0]);  alert(htmldata);  });  } |

### App Authenticity

* **Add securityTest="MetLifeCustomAppAuthenticitySecurity"**in application-descriptor.xml
* Note that for App Authenticity to **really work**, you **must use** the Consumer edition of IBM Worklight.
* **Steps to enable App Authenticity:**

1. Configure a security test that has the wl\_authenticityRealm realm:

<customSecurityTest name="customTests">

<test realm="wl\_antiXSRFRealm" step="1"/>

<test realm="wl\_authenticityRealm" step="1"/>

<test realm="wl\_remoteDisableRealm" step="1"/>

<test realm="wl\_anonymousUserRealm" isInternalUserID="true" step="1"/>

<test realm="wl\_deviceNoProvisioningRealm" isInternalDeviceID="true" step="2"/>

</customSecurityTest>

1. Place this securityTest on the environment(s) in application-descriptor.xml, for example:

<android version="1.0" securityTest="customTests">

1. For Android, generate the public signing key:

<android version="1.0" securityTest="customTests">

<worklightSettings include="true"/>

<security>

<encryptWebResources enabled="false"/>

<testWebResourcesChecksum enabled="false" ignoreFileExtensions="png, jpg, jpeg, gif, mp4, mp3"/>

<publicSigningKey>MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBdfdsfdsfiQKBgQCPCbaCfAfnAqQ12/S5LLfA4cBz/3INyLRPhFGSVFztdWNzPhrna1xDc8/3V1sUIW2odfdfddfdfffdR2n3dAnNMVydfdfdfv68gmU5qVCN4LxSKKRAj7VVbhBxBIEt5MbY+c0o7NZ2Pgu/moJee8Wgu8veZ1TJntYn+cWCYuI/JSnA9nsskwhSdPHK32edsfsdfdfQIDAQAB</publicSigningKey>

</security>

</android>

1. For iOS, place the app bundleId:

<iphone bundleId="com.worklight.test" version="1.0" securityTest="customTests">

1. Run As > Run on Worklight Development Server

Now the App Authenticity dropdown should be enabled in Worklight Console.

### FastClick.js

* Reference URL: [**https://docs.angularjs.org/api/ngTouch/directive/ngClick**](https://docs.angularjs.org/api/ngTouch/directive/ngClick)
* The above URL says it is incorporated with ngTouch/ngclick but if there is any concern during the development then fastclick.js will be considered.

### Wrapper Classes

1. All IBM Worklight adapters need to be accessed via wrapper classes. For example, Worklight adapter needs to be called through a wrapper. This needs to be followed throughout the implementation mainly to decouple worklight objects from Metlife Global Sales app.
2. Adapter wrapper is an separate angular module which needs to be included in application module for usage
   * angular.module('adapterWrapper', []).service('$adapterUtility', function(){
   * this.invoke= function(p1,p2,p3,p4,p5,p6){
   * var options={},invocationData={}; **//Adapter wrapper class**
   * if(typeof p6 !== "undefined") { //p1,p2,p3,p4,p5,p6 are formal parameters
   * invocationData = {
   * adapter : p1,
   * procedure : p2,
   * parameters : [p3],
   * compressResponse : p4
   * };
   * options = {
   * onSuccess : p5,
   * onFailure : p6
   * };
   * }
   * else if(typeof p5 !== "undefined") {
   * invocationData = {
   * adapter : p1,
   * procedure : p2,
   * parameters : [p3]
   * };
   * options = {
   * onSuccess : p4,
   * onFailure : p5
   * };
   * }
   * else if(typeof p4 !== "undefined") {
   * invocationData = {
   * adapter : p1,
   * procedure : p2,
   * parameters : []
   * };
   * options = {
   * onSuccess : p3,
   * onFailure : p4
   * };
   * }
   * WL.Client.invokeProcedure(invocationData, options);
   * };
   * });
   * **//Attaching Adapter Wrapper class to application module.**
   * **var** app = angular.module('MetLife', ['common','adapterWrapper']);
3. $adapterUtility is the service provided by Adapter wrapper class which holds the method invoke() for invoking any adapter calls.
4. Invoke() method can take maximum of 6 parameters and minimum of 4 parameters.
5. The four mandatory parameters for invoke() method are adapterName, procedureName, successCallback and failureCallback.
6. If number of parameter is 6:

Following are the parameters that need to be included and in the following Sequence:

adapterName: Name of the adapter.

procedureName: Name of the procedure inside that adapter.

parameter: Parameters that need to be passed for POST request. Parameter should be in string format. If it is in JSON format, it need to be converted into string.

compressResponse: It takes boolean value. Whether to compress the response.

successCallback: If the result is success, successCallback function is called.

failureCallback: If the result is error, successCallback function is called.

$adapterUtility.invoke(adapterName,procedureName,parameter,compressResponse,successCallback,failureCallback);

}

function successCallback(result)

{

//result object will hold the responseData from the webservice

}

function failureCallback(Error)

{

//error object will hold error code and error message.

}

1. If number of parameter is 5:
   * Following are the parameters that need to be included and in the following Sequence:

adapterName: Name of the adapter.

procedureName: Name of the procedure inside that adapter.

parameter: Parameters that need to be passed for POST request. Parameter should be in string format. If it is in JSON format, it need to be converted into string.

successCallback: If the result is success, successCallback function is called.

failureCallback: If the result is error, successCallback function is called.

$adapterUtility.invoke(adapterName,procedureName,parameter,successCallback,failureCallback);

}

function successCallback(result)

{

//result object will hold the responseData from the webservice

}

function failureCallback(Error)

{

//error object will hold error code and error message.

}

1. If number of parameter is 4 (for GET request):

Following are the parameters that need to be included and in the following Sequence:

adapterName: Name of the adapter.

procedureName: Name of the procedure inside that adapter.

successCallback: If the result is success, successCallback function is called.

failureCallback: If the result is error, successCallback function is called.

$adapterUtility.invoke(adapterName,procedureName,successCallback,failureCallback);

}

function successCallback(result)

{

//result object will hold the responseData from the webservice

}

function failureCallback(Error)

{

//error object will hold error code and error message.

}

### IBM Worklight Logger

IBM Worklight Logging will be used for app debugging purpose. Below are the details:

Use WL.Logger.debug and edit server.xml to view the log in the trace.log file

1. Open the Servers view in Eclipse
2. Expend the Worklight Development Server entry
3. Double-click on Server Configuration (server.xml)
4. Switch to Source tab
5. Uncomment this line: <logging traceSpecification="com.worklight.\*=debug=enabled"/>
6. After invoking adapter procedure, find the log in <eclipseWorkspace>\WorklightServerConfig\servers\worklight\logs\trace.log

Re-deploy the adapter before attempting to view the logs.

Alternatively,  
Use WL.Logger.warn or WL.Logger.error; these logs will display in the Worklight Development Server Console view.

### Webtrends Analytics

Below are the steps for including web trend analytics in android project. IOS link is provided in the last point.

1. Add the Webtrends JAR file to your Android project (Project Properties > Java Build Path > Libraries > Add External Jars)
2. Edit AndroidManifest.xml to supply permissions (minimum: INTERNET and ACCESS\_NETWORK\_STATE).
3. Import the Webtrends.xml file into your project.
4. Change the wt\_dc\_dcsid value to your DCSID, and change the wt\_dc\_timezone value to match the time zone of the Webtrends data source, expressed as an offset from GMT. Provide values for wt\_dc\_app\_name, wt\_dc\_app\_publisher, wt\_dc\_app\_category, and wt\_dc\_app\_version also.

The DCSID is the unique ID associated with a Webtrends data source. If you don't have a DCSID or you don't know what a data source is, contact your Webtrends administrator.

1. Import the Webtrends Analytics class by adding these imports: import com.webtrends.mobile.analytics.\*; import com.webtrends.mobile.android.\*;
2. Use automatic methods and extend WebtrendsApplication and/or WebtrendsActivity, or manually call WebtrendsConfigurator.ConfigureDC(this) and call an event collection method. See Initialize Application Tracking and Extend the Activity for details.
3. You can compile your application and start tracking right away. For advanced analytics capability, use the Event Collection methods to instrument your application.
4. Web Trend Analytics Setup instructions:
   1. <http://help.webtrends.com/en/ios/>
   2. <http://help.webtrends.com/en/android/>
5. As discussed during the design meetings, scope is to call the manual instrumentation at one or two places to check the webtrends analytics.
6. MetLife will test the Web Trend Analytics in UAT.

## Flowcharts

### User Authentication

Following flowchart describes the basic Login Flow:



**Processing and Implementation Logic for Login screen:**

* Very first time when the user logs into the app, user credentials won’t be present in JSONStore.
* Agent should be online to login to the app the very first time. If agent is offline and tries to login, display the error message “Network connectivity is required to login to the app for the very first usage”
* Once username and password entered and clicks on Login button, LoginAuthentication webservice is called which validates the user credentials.
* LoginAuthentication webservice returns the JSON object with only 1 user details as [admin@gsp.com](mailto:admin@gsp.com) as the username and “admin1234#” as the password.
* After the successful authentication of the user, the username and password gets stored in JSON storage with encrypted format. This will be used for offline authentication.
* If the user has not used the app for 15 mins for example, the app will do auto logout. This 15 mins value is configurable in AppConfig JSON object.
* Next time when the user logs in, first check whether the network connectivity is there. If yes, verify the user credentials against the server. If valid, take the user to Home screen if at least one product is downloaded. If invalid, alert the user to enter the right credentials. If there is no network connectivity, then verify the user credentials with the local JSON storage.

#### Multiple Users login in same device and MetLife GSA App

* When user logged in, user credentials are stored in JSONStore if not present already and username and agent details are prepopulated in Settings screen if already available in Settings JSON object.
* Customer create has the association with agent code, who entered customer details and created insurance quote.
* Customer data for that particular agent will be displayed in Home, Customer History and select customer search screens

### App Flow

Following flow chart describes the app flow for different module.

Main points:

* 1. Agent can start the flow from customer create, customer fact find form, FNA and Sales Illustration in sequence or in any order as shown in below flowcharts.
  2. FNA is optional to go to Sales Illustration.
  3. Agent can directly start the FNA flow without selecting customer by selecting FNA icon in the bottom footer.
  4. Agent can directly start the Sales Illustration flow without selecting customer by selecting Illustration icon in the bottom footer.
  5. Agent can do Sales Illustration after FNA without selecting customer in direct flow.
  6. Agent entered customer profile data can be saved any time with a profile name. Or it must be saved before navigating to record of advice section.
  7. If agent clicks on FNA or CFF or Illustration icons from footer while the current form is not saved OR “not entered data in all screens and not moved to profile status screen yet”, then Agent will be alerted with message “Unsaved data will be lost, do you want continue ? (Options OK, CANCEL)”
  8. After completing CFF or FNA or Illustration, user will be directed to profile status screen.
  9. Fields list in below chart is not complete, it is for indicative purpose.



### Products Download

Following flowchart provides the detailed flow for downloading of common data, product deployment package, product list, product details and content.



### Sales Illustration flow

Following flowchart provides the detailed flow for sales illustration modules. (View this flowchart in 200% for readability)



**Processing and Implementation Logic for Sales Illustration Plan screen:**

* Sales Illustration – Plan page HTML Template is defined for each product
* Sales Illustration – JSON object is defined for each product mapped to PX Calculator Input XML file
* Sales Illustration – PDF HTML Template is defined for each product
* HTML Template will contain below sections
* **NOTE – Refer to MetLife-GSA-Fields-Validations\_Msgs\_Translation file attached at the end of this document to build the HTML template and rules to be followed.**
  + 1. Product Category - Protection
       1. AMMetLife LifeStyle (Investment Linked Product)
          1. Basic info
          2. Plan

Basic plan (Fields present in UI + "InsurancePortion", "SustainabilityOption")

Riders

Fund (Fund name, percentage)

Top up - Yes

Preferences

* + - 1. AMMetLife Link (Investment Linked Product)
         1. Basic info
         2. Plan

Basic plan (< No Product Details JSON for this plan specification document>)

Riders

Fund Type

Top up

Preferences

* + 1. Fund (Fund code, fund name, percentage)Product Category - Wealth
       1. AMMetLife SecureGuard Plus
          1. Basic info
          2. Plan

Basic plan (Fields in UI screen)

Riders (code, name, coverage term, premium term, sum assured, premium)

Fund Type not applicable

Top up not applicable

Preferences - Yes

* + - 1. AMMetLife SecureWealth
         1. Basic info
         2. Plan

Basic plan (Fields in UI screen)

Riders (code, name, coverage term, sum assured, premium)

Fund Type not applicable

Top up not applicable

Preferences – Yes

* + - 1. AMMetLife SecureBuilder
         1. Basic info
         2. Plan

Basic plan (< No Product Details JSON in specification document for this plan>)

Riders (code, name, coverage term, premium term, sum assured, premium)

Fund Type not applicable

Top up not applicable

Preferences- Yes

* Follow the above flow chart to populate the default field values in input fields from Product Details JSON.
* Once Agent enters or selects applicable riders, fund type, preferences, top-up, assign this data to sales Illustration JSON object.
* Product specific calculate JSON object is created with exact mapping to Product Xpress Calculator API input XML.
* Product specific JSON object is converted to XML file using xml2json.js Library API
* Invoke PX calculate API with this input xml file
* Get the output XML by calling calculate API
* Output XML contains table data with column header names, column values and header names mapping.
* Fill the Table column values in PDF HTML template for this product. Each product will have its own PDF HTML Template.
* Fill the basic info values in PDF HTML template first basic info section
* Fill the selected Fund types as applicable in Fund types section
* Fill the selected riders as applicable in riders table section
* HTML Template to JSON Object MAPPING:

|  |  |
| --- | --- |
| **HTML/Configurator Template** | **JSON Object** |
|  |  |
|  | [{](javascript:;) |
|  | ["CalculationInput":{](javascript:;) |
|  | ["DeplR":{](javascript:;) |
| illustration.plan.basic.planName | **"\_dep-name"**:"AmLife Secure Wealth", |
|  | **"\_ver-sel"**:"last", |
|  | **"\_\_prefix"**:"clc" |
|  | }, |
|  | ["CalculationData":{](javascript:;) |
|  | ["Policy":{](javascript:;) |
|  | ["Features":{](javascript:;) |
| illustration.plan.basic.policyTerm | ["CoverageTerm":{](javascript:;) |
| Logic to mark these parameters as true and keep entered value to be discussed<TBD>. This need to be tested in calculator by including and not including this params and check variation in output XML. | **"\_xsi:nil"**:"true" |
|  | }, |
|  | ["DeductibleLevel":{](javascript:;) |
|  | **"\_xsi:nil"**:"true" |
|  | }, |
| If extended coverage option is selected | ["ExtendedCoverageOption":{](javascript:;) |
|  | **"\_xsi:nil"**:"true" |
|  | }, |
| illustration.plan.basic.insurancePortion | ["InsurancePortion":{](javascript:;) |
|  | **"\_xsi:nil"**:"true" |
|  | }, |
| illustration.plan.basic.paymentFrequency | **"PaymentFrequency"**:"1", |
| illustration.plan.basic.premiumTerm | ["PaymentTerm":{](javascript:;) |
|  | **"\_xsi:nil"**:"true" |
|  | }, |
| illustration.plan.basic.planCode | ["PlanCode":{](javascript:;) |
|  | **"\_xsi:nil"**:"true" |
|  | }, |
| illustration.plan.basic.planCode | ["ProductCode":{](javascript:;) |
|  | **"\_xsi:nil"**:"true" |
|  | }, |
| illustration.plan.basic.sustainabilityOption | ["SustainabilityOptionTerm":{](javascript:;) |
|  | **"\_xsi:nil"**:"true" |
|  | }, |
| illustration.plan.basic.policyTerm + ANB | ["TargetAge":{](javascript:;) |
|  | **"\_xsi:nil"**:"true" |
|  | }, |
| illustration.plan.basic.sumAssured | ["TargetValue":{](javascript:;) |
|  | **"\_xsi:nil"**:"true" |
|  | }, |
| illustration.plan.basic.yearlyPremium | ["TotalPremium":{](javascript:;) |
|  | **"\_xsi:nil"**:"true" |
|  | }, |
| If withdrawal option is selected | **"WithdrawalOption"**:"2" |
|  | }, |
|  | ["Links":{](javascript:;) |
|  | ["Coverage":{](javascript:;) |
|  | ["Coverage":{](javascript:;) |
|  | ["Features":{](javascript:;) |
| illustration.plan.basic.planCode | **"CoverageCode"**:"BTAS3G1WPA", |
| illustration.plan.basic.policyTerm | ["CoverageTerm":{](javascript:;) |
|  | **"\_xsi:nil"**:"true" |
|  | }, |
| illustration.plan.basic.sumAssured | **"InsuredAmount"**:"15000", |
|  | ["PaymentTerm":{](javascript:;) |
|  | **"TermType"**:"Duration", |
| illustration.plan.basic.premiumTerm | **"Term"**:"10" |
|  | }, |
| illustration.plan.basic.planCode | ["PlanCode":{](javascript:;) |
|  | **"\_xsi:nil"**:"true" |
|  | } |
|  | }, |
|  | ["Links":{](javascript:;) |
| if Payor - "Payor", if Lifeassured - "Insured" | **"Insured"**:"" |
|  | }, |
|  | **"\_id"**:"ID26" |
|  | } |
|  | }, |
|  | **"FundInvestment"**:"", |
|  | ["Insured":{](javascript:;) |
|  | ["Insured":{](javascript:;) |
|  | ["Features":{](javascript:;) |
| if Payor - "Payor", if Lifeassured - "Insured" | **"RoleCode"**:"Insured" |
|  | }, |
|  | ["Links":{](javascript:;) |
|  | ["Person":{](javascript:;) |
|  | ["Reference":{](javascript:;) |
|  | **"\_ref"**:"ID49", |
|  | **"\_\_prefix"**:"clc" |
|  | } |
|  | } |
|  | }, |
|  | **"\_id"**:"ID39" |
|  | } |
|  | }, |
|  | ["Owner":{](javascript:;) |
|  | ["Owner":{](javascript:;) |
|  | ["Features":{](javascript:;) |
|  | ["RoleCode":{](javascript:;) |
|  | **"\_xsi:nil"**:"true" |
|  | } |
|  | }, |
|  | ["Links":{](javascript:;) |
|  | ["Person":{](javascript:;) |
|  | ["Reference":{](javascript:;) |
|  | **"\_ref"**:"ID49", |
|  | **"\_\_prefix"**:"clc" |
|  | } |
|  | } |
|  | }, |
|  | **"\_id"**:"ID43" |
|  | } |
|  | }, |
|  | **"TopUp"**:"" |
|  | }, |
|  | **"\_id"**:"ID1" |
|  | }, |
|  | ["Person":{](javascript:;) |
|  | ["Features":{](javascript:;) |
| illustration.basicinfo.lifeassured.ageNextBirthday | **"AgeNextBirthday"**:"17", |
| illustration.basicinfo.lifeassured.dateofBirth values is entered | ["BirthDate":{](javascript:;) |
|  | **"\_xsi:nil"**:"true" |
|  | }, |
| illustration.basicinfo.lifeassured.gender (male-1, Female - 2) | **"Gender"**:"2", |
| illustration.basicinfo.lifeassured.occupationClass | **"OccupationClass"**:"1", |
| illustration.basicinfo.lifeassured.smokingHabit | **"SmokingHabit"**:"N" |
|  | }, |
|  | **"\_id"**:"ID49" |
|  | }, |
|  | **"\_\_prefix"**:"clc" |
|  | }, |
|  | ["Calculation":{](javascript:;) |
|  | ["Policy":{](javascript:;) |
|  | ["Validations":{](javascript:;) |
|  | **"All\_validations"**:"" |
|  | }, |
|  | ["Features":{](javascript:;) |
|  | **"WithdrawalOption"**:"", |
|  | **"RequiredAmount"**:"", |
|  | **"Tables"**:"" |
|  | }, |
|  | ["Links":{](javascript:;) |
|  | ["Coverage":{](javascript:;) |
|  | ["Coverage":{](javascript:;) |
|  | ["Features":{](javascript:;) |
|  | **"PremiumPerMode"**:"" |
|  | }, |
|  | **"\_ref"**:"ID26" |
|  | } |
|  | }, |
|  | ["Insured":{](javascript:;) |
|  | ["Insured":{](javascript:;) |
|  | ["Links":{](javascript:;) |
|  | ["Person":{](javascript:;) |
|  | ["Reference":{](javascript:;) |
|  | **"\_ref"**:"ID49", |
|  | **"\_\_prefix"**:"clc" |
|  | } |
|  | } |
|  | }, |
|  | **"\_ref"**:"ID39" |
|  | } |
|  | }, |
|  | ["Owner":{](javascript:;) |
|  | ["Owner":{](javascript:;) |
|  | ["Links":{](javascript:;) |
|  | ["Person":{](javascript:;) |
|  | ["Reference":{](javascript:;) |
|  | **"\_ref"**:"ID49", |
|  | **"\_\_prefix"**:"clc" |
|  | } |
|  | } |
|  | }, |
|  | **"\_ref"**:"ID43" |
|  | } |
|  | } |
|  | }, |
|  | **"\_ref"**:"ID1" |
|  | }, |
|  | ["CalculationDates":{](javascript:;) |
|  | ["At":{](javascript:;) |
|  | **"\_\_prefix"**:"clc", |
|  | **"\_\_text"**:"2014-09-18" |
|  | }, |
|  | **"\_\_prefix"**:"clc" |
|  | }, |
|  | ["InputData":{](javascript:;) |
|  | **"\_scenario"**:"main", |
|  | **"\_\_prefix"**:"clc" |
|  | }, |
|  | **"\_name"**:"main test", |
|  | **"\_\_prefix"**:"clc" |
|  | }, |
|  | **"\_xmlns:xsi"**:"http://www.w3.org/2001/XMLSchema-instance", |
|  | **"\_xmlns:clc"**:"http://www.solcorp.com/ns/ProductXpress/CalculationInputOutput/CalculatorElement", |
|  | **"\_xmlns"**:"http://www.example.org/AmLife\_Secure\_Wealth", |
|  | **"\_detailed-errors"**:"true", |
|  | **"\_request-id"**:"1", |
|  | **"\_\_prefix"**:"clc" |
|  | } |
|  | } |

### Financial Need Analysis flow

Following flowchart provides the detailed flow for FNA modules.



### Screen Navigation Flow

Following flowchart describes the detailed screen navigation flow.



# Glossary and Acronyms

| Term or Acronym | Definition |
| --- | --- |
| API | Application Programming Interface |
|  |  |

# Appendix

File Name - MetLife-GSA-Fields-Validations\_Msgs\_Translation\_V1.3

Contains: Mandatory fields, dependencies between fields, validations, alert messages, information notifications, Rules for fields, common data.

File Link:

