Insurance Management Information System (IMIS) Application
Functional Design Specifications
Functional Area: Insurees and Policies, Administration of registers,
Management of claims and general management

Created by:	Exact Software Ltd
Deliverable Owner:	Swiss TPH
Deliverable Approvers:	
Last Revision Date:	24 April 2017
Version & Status:	17.3.11

Contents

1. Overview

1.1. Introduction

This document cover the design of IMIS Master Version Phase 3 version 17.3.11 for Insurees and policies , administration of registers,Management of Claims and general management, based on the specification of the Insurance Management Information System for the new Community Health Fund structure.

We will elaborate on the technologies used and will highlight certain design aspects of the application.

1.2. Scope

This document will cover the overall design of the Insurance Management Information System for the new IMIS Master version Phase 3: Insurees and policies, administration of registers, management of claims and general management.

The document will cover six main areas of the development in the following order:

- Overall system architecture
- Programming approach, standards and methods
- User Interface design
- Database design
- Window and web services
- Online, Offline and Mobile Phone concept
- external librariesand third party components used

The application is developed with a multi-tier approach. A separate section will cover the overall 'skeleton' of the IMIS application by elaborating on the internal structure of the Visual Studio solution. Separate paragraphs will cover the coding standards and user security aspects.

The user interface design chapters will start with the menu-structure of the application. Thereafter we will provide the design of each interface separately and will cover the functional area by providing a brief overview of the interface, the menu-link, reference to use cases and the main programming methods and properties used within the interface (programming model).

Based on the data model, a full database design documented with its main properties and constraints. Also a graphical representation included.

A separate section will further discuss the use of mobile phones within the IMIS application. A brief description provided on the interface technology used for the transfer of data between phones and the central database.

This functional design document in conjunction with the draft specification of the Insurance Management Information System version 17.3.11 should remain the final authority for developers.

1.3. Audience

This document used to develop and implement components for the IMIS application. They translate the functionality described in this document directly into the actual functionality of the Administration of registers and Management of Insurees and policies.

This document is also used by testers to define test scenarios for the functional areas and to test application security.

1.4. Assumptions

Readers will be familiar with terminology used within the IMIS project.

1.5. Document History

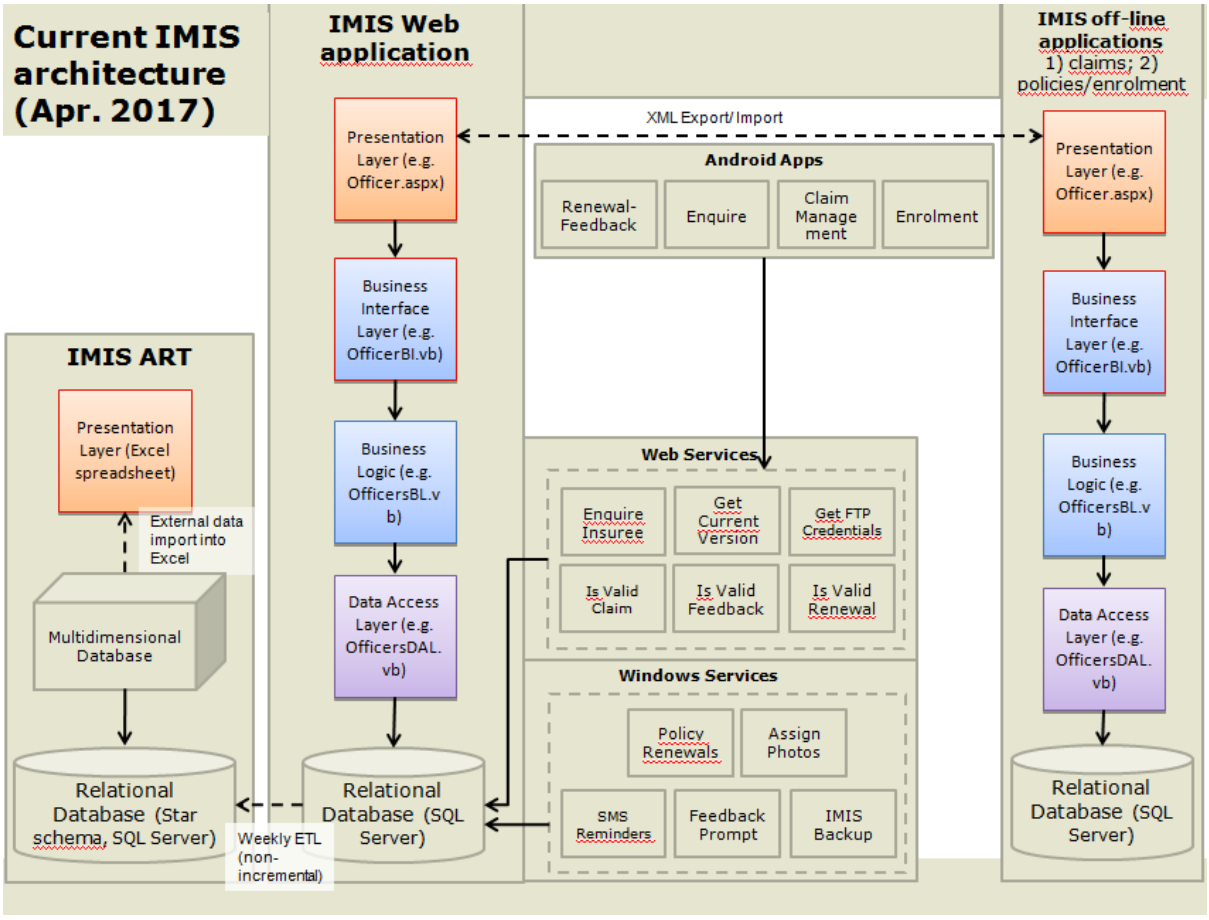
Ver.	Description	Author	Date
16.1.1	IMIS Master phase 1	Hans van Hoppe	24-10-16
16.2.0	IMIS Master phase 2	Hans van Hoppe	15-11-16
17.3.11	IMIS Master Phase 3	Rogers Obed	24-04-17

2. System Architecture

This chapter will cover the overall architectural design of the IMIS system. The design can be divided in several components:

- IMIS online web application
- IMIS offline web application
- Andriod phone applications
- Windows services and Web services
- IMIS data warehouse

See below a graphical overview.



2.1.1.IMIS online web Application

The IMIS online web application with its 4-Tier architecture is covered in Chapters 3,4,5 and 6.

2.1.2.IMIS offline web Application

IMIS offers, besides the online application, also the so-called 'offline' applications. IMIS offers 2 offline applications:

- Offline 'enrollment' office (CHF-offline)
- Offline Health Facility (HF-offline)

The offline applications are used in case connectivity is non existing or relatively poor. The online and offline components are exactly the same in terms of coding and architecture. The IMIS offline web application with its 4-Tier architecture is covered in Chapters 3,4,5 and 6.

2.1.3. Android phone applications

IMIS has four different applications:

- Renew-Feedback application - used by the enrollment officer to re-new insuree policies. The application is also used to provide health facility feedback from insurees.
- Enquire application - used for checking insuree details.
- Claim management application - used by the claim administrator to register claim details from the insurees.
- Enrollment application- used for enrolling insuree by taking their pictures and provides the insurance number.

For more details on Android phone applications, please refer to Chapter 7.

2.1.4. Window services and Web services

Windows services are used for automatic database backup, update insurance policy expiry details and to send SMS to insurees; while Web services are used for connecting between the android application and the core IMIS application.

Web services are also used to execute background activities such as in-background extracts. For more details, please refer to Chapter 8.

2.1.5. IMIS Data warehouse

Data warehouse is a system used for data analysis and reporting. For more information on the IMIS Data warehouse, please refer to the document on ***IMIS Data warehouse Deployment.***

3. Programming approach, standards and methods

This chapter will further elaborate the architecture of the IMIS solution, coding standards and the methodology of storing images.

3.1. Multi Tier development

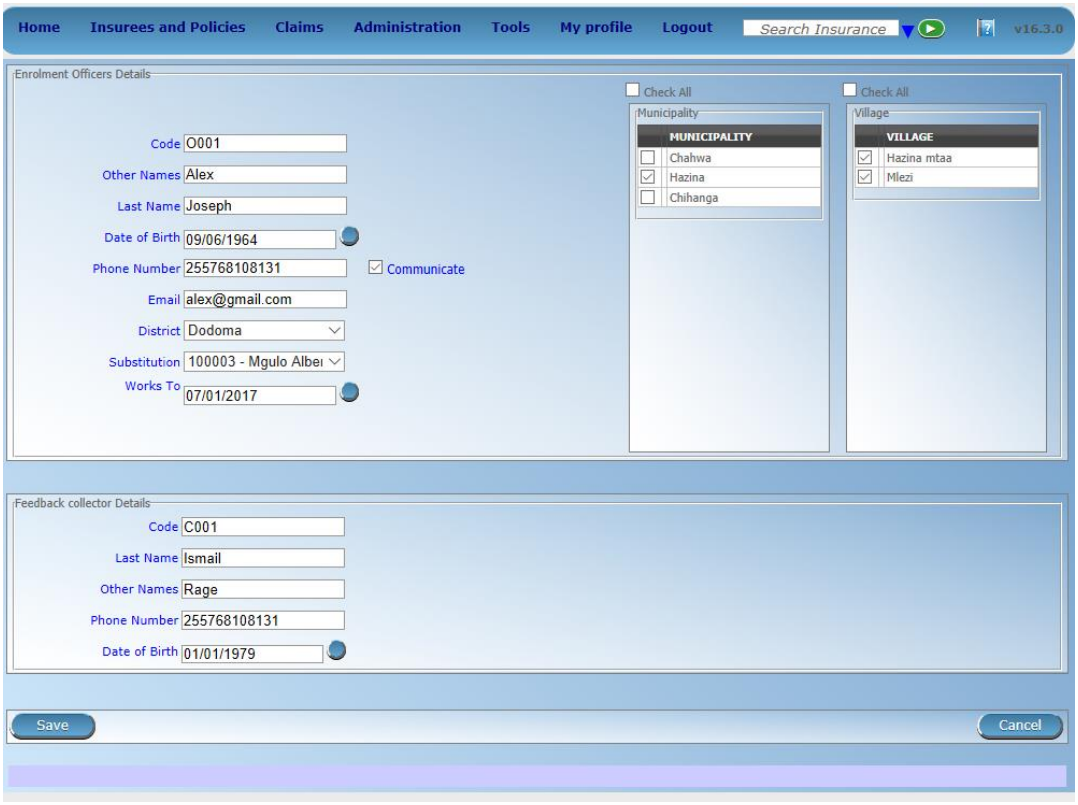
IMIS is developed with a multi tier approach. Each Tier is represented in the Visual Studio solution as a separate project.
The following projects (tiers) can be found in the IMIS Solution:

Presentation Layer	→ IMIS
Business Interface Layer	→ IMIS_BI
Business Logic Layer	→ IMIS_BL
Data Access Components Layer	→ IMIS_DAL

At the base of the application is the Data Access Layer (DAL) which communicates via a standard library with the SQL Server. Above the DAL is the Business Logic Layer (BLL) which references the DAL and is used to store all methods used in the application. The next layer is the Business Interface Layer (BIL) which acts purely as an organization layer, here only the method calls are stored. Finally the web pages are stored in the Presentation Layer (PL).
The example below shows how the application references each layer with the Officer Entity

Presentation Layer 	The page Officer.aspx is used to enter and edit officer (Enrollment Assistant) data and use the Officer Entity. While Validation of the page is handled client side within the aspx page, all server side actions are handled by referencing methods found in the Business Interface Layer.
Business Interface Layer 	In the Business Interface Layer, each page has a separate class which stores all the methods available for the page. No programming code is found in these classes the layer acts purely as a container to store specific methods for specific forms. Each method references methods in the Business Logic layer
Business Logic Layer 	The Business Logic Layer is the part of the program that contains nearly all the logic for the application. Events triggered from the presentation layer are sent via the Business Interface Layer to the Business Logic Layer where most of the programmatic decision making methods are found. In most cases the methods will reference the Data Access Layer before returning the results back to the presentation layer.

Data Access Layer OfficerDAL OfficerDAL	The Data Access layer is used for storing all methods used to communicate with the database via a Standard Library Class . In the case of IMIS, each Entity has a separate class for calls to the SQL Server. Methods are referenced from the Business Logic Layer and results are sent back to the Presentation Layer.
SQL Server Class Exact.DLL	The SQL Server contains standard methods native to SQL Server. All class within the DAL SQL namespace reference the Exact.DLL which communicates directly with the SQL Server.



When the form loads, the page load event is fired, references to the entity eOfficer and the Business Interface class IMIS_BI.OfficerBI

```
Dim eOfficer As New IMIS_EN.tblOfficer
Dim Officer As New IMIS_BI.OfficerBI
```

3 methods are called, getDistricts, to load the districts, GetOfficers, to load the list of substitution officers in the selected District and LoadOfficer to load the details of the officer if editing.

```
Officer.GetDistricts(imisgen.getuserId(session("user")), True)
```

imisgen.getuserId("user") indicates the User loged in , True indicates that the first row is a user prompt

```
Officer.GetOfficers(1)
```

1 indicates the district selected

```
Officer.LoadOfficer(eOfficer)
```

eOfficer is the entity of the officer

IMIS Business Interface Layer – Class Officer

```
Public Class OfficerBI
    Public Function SaveOfficer(ByRef eOfficer As IMIS_EN.tblOfficer) As Integer
        Dim saveData As New IMIS BL.OfficersBL
        Return saveData.SaveOfficers(eOfficer)
    End Function
    Public Function GetDistricts(ByVal userId As Integer, Optional ByVal showSelect As Boolean
= False) As DataTable
        Dim Districts As New IMIS BL.LocationsBL
        Return Districts.GetDistricts(userId, showSelect)
    End Function
    Public Function GetOfficers(ByVal DistrictId As Integer, ByVal showSelect As Boolean) As
DataTable
        Dim getDataTable As New IMIS_BL.OfficersBL
        Return getDataTable.GetOfficers(DistrictId, showselect)
    End Function
    Public Sub LoadOfficer(ByRef eOfficers As IMIS_EN.tblOfficer)
        Dim loadEntity As New IMIS_BL.OfficersBL
        loadEntity.LoadOfficer(eOfficers)
    End Sub

End Class
```

Each method now references the specific logic class from the Business Logic class
GetDistricts calls the method from the class LocationsBl, while GetOfficers and LoadOfficer calls methods from the OfficerBL class.

```
Public Class LocationsBL
    Private imisgen As New GeneralBL
    Public Function GetDistricts(ByVal userID As Integer, ByVal showSelect As Boolean,
Optional IncludeNational As Boolean = False) As DataTable
        Dim Districts As New IMIS_DAL.LocationsDAL
        Dim dt As DataTable = Districts.GetDistricts(userID)
        If IncludeNational Then
            Dim dr As DataRow = dt.NewRow
            dr("DistrictId") = -1
            dr("DistrictName") = imisgen.getMessage("M_NATIONAL")
            dt.Rows.InsertAt(dr, 0)
        End If

        If dt.Rows.Count > 1 Then
            If showSelect = True Then
                Dim dr As DataRow = dt.NewRow
                dr("DistrictId") = 0
                dr("DistrictName") = imisgen.getMessage("T_SELECTDISTRICT")
                dt.Rows.InsertAt(dr, 0)
            End If
        End If
        Return dt
    End Function

Public Class OfficersBL

Public Function GetOfficers(ByVal DistrictId As Integer, ByVal showselect As Boolean) As
DataTable
        Dim getDataTable As New IMIS DAL.OfficersDAL
        Dim dtOfficer As DataTable = getDataTable.GetOfficers(DistrictId)

        If showselect = True Then
            Dim dr As DataRow = dtOfficer.NewRow
            dr("OfficerID") = 0
            dr("Code") = imisgen.getMessage("T_SELECTOFFICER")
            dtOfficer.Rows.InsertAt(dr, 0)
        End If
        Return dtOfficer
    End Function
Public Sub LoadOfficer(ByRef eOfficers As IMIS_EN.tblOfficer)
        Dim load As New IMIS DAL.OfficersDAL
        load.LoadOfficer(eOfficers)
    End Sub
```

Each reference from the BLL, references a similar method from the specific DAL class.

```
Public Function GetDistricts(ByVal UserID As Integer) As DataTable
        Dim data As New EXACT.ExactSQL
        data.setSQLCommand("select * from tblDistricts order by DistrictName",
CommandType.Text)
        data.params("@UserID", SqlDbType.Int, UserID)

        Return data.Filldata
    End Function

Public Class OfficersDAL
Public Function GetOfficers(ByVal DistrictId As Integer) As DataTable
        Dim data As New EXACT.ExactSQL
        data.setSQLCommand("select tblOfficer.*,Districtname from tblOfficer inner join
tblDistricts on tblOfficer.DistrictID = tblDistricts.DistrictID where tblOfficer.LegacyId is
Null AND  tblOfficer.DistrictID  = @DistrictID ORDER BY LastName", CommandType.Text)
        data.params("@DistrictID", SqlDbType.Int, DistrictId)
        Return data.Filldata
    End Function

    Public Sub LoadOfficer(ByRef eOfficers As IMIS_EN.tblOfficer)
        Dim data As New ExactSQL
        Dim dr As DataRow
        data.setSQLCommand("select * from tblOfficer where OfficerID=@OfficerId",
CommandType.Text)
        data.params("@OfficerId", SqlDbType.Int, eOfficers.OfficerID)
        dr = data.Filldata()(0)
        If Not dr Is Nothing Then
            Dim eDistricts As New IMIS_EN.tblDistricts
            eDistricts.DistrictID = dr("DistrictID")
            eOfficers.tblDistricts = eDistricts
            eOfficers.Code = dr("Code")
            eOfficers.LastName = dr("LastName")
            eOfficers.OtherNames = dr("OtherNames")
            eOfficers.DOB = IIf(dr("DOB") Is DBNull.Value, Nothing, dr("DOB"))
            eOfficers.Phone = IIf(dr("Phone") Is DBNull.Value, Nothing, dr("Phone"))
            eOfficers.WorksTo = IIf(dr("WorksTo") Is DBNull.Value, Nothing, dr("WorksTo"))
            Dim eofficer2 As New IMIS EN.tblOfficer
            eofficer2.OfficerID = IIf(dr("OfficerIDSubst") Is DBNull.Value, 0,
dr("OfficerIDSubst"))
            eOfficers.tblOfficer2 = eofficer2
            eOfficers.VEOCode = IIf(dr("VEOCode") Is DBNull.Value, Nothing, dr("VEOCode"))
            eOfficers.VEOPhone = IIf(dr("VEOPhone") Is DBNull.Value, Nothing, dr("VEOPhone"))
            eOfficers.VEOLastName = IIf(dr("VEOLastName") Is DBNull.Value, Nothing,
dr("VEOLastName"))
            eOfficers.VEOOtherNames = IIf(dr("VEOOtherNames") Is DBNull.Value, Nothing,
dr("VEOOtherNames"))
            eOfficers.VEODOB = IIf(dr("VEODOB") Is DBNull.Value, Nothing, dr("VEODOB"))
            If Not dr("ValidityTo") Is DBNull.Value Then
                eOfficers.ValidityTo = dr("ValidityTo").ToString
            End If
            eOfficers.EmailId = dr("EmailId").ToString
            eOfficers.PhoneCommunication = If(dr("PhoneCommunication") Is DBNull.Value, False,
dr("PhoneCommunication"))
        End If
    End Sub
```

The Data Access Layer returns data from the SQL Server and loads it into the entity, where it is returned to the presentation layer to load the controls within the page. Errors are handled only within the presentation level, if an exception is triggered, handled in the presentation layer, using the Try and Catch. Errors are reported back to the user, by means of the information panel at the bottom of each page.

3.2. Coding Standards and Conventions

In all classes, the following standards and or prefixes could be found. Hereunder we elaborate on the prefixes and their meaning.

3.2.1.Entities

Prefix	Description	Example
E	The Entity for the current data object	eOfficer

3.2.2.Methods

Prefix	Description	Example
Load	Loads data into the entity	LoadOfficer(eOfficer)
Get	Returns data into a datatable	GetOfficer(DistrictId)
Save	Saves data back to the Database	Save(eOfficer)
Insert	Insert of data into the database	Insert(eOfficer)
Update	Update of data into the database	Update

3.2.3.Controls

Prefix	Description	Example
Txt	TextBox	txtLastName
b_	button	btnSave
Ddl	Drop Down List	ddlDistricts
Chk	Check box	chkLegacy
Pnl	Panel	pnlTop
Grv	Gridview	grvOfficers

3.2.4.Objects

Prefix	Description	Example
Dt	Datatable	dtDistricts
Dr	DataRow	dr*
Data	SQL Instance	data(Filldata)
I	Integer	iCount
Str	String	strLocation
B	Boolean	bIsPoverty
Dbl	Double	dblTotal
Dec	Decimal	decPremiumTotal
A	Array	aArray
Dte	Date	dteExpiry

3.2.5.Resources

Prefix	Description	Example
L_	Label test	L_LASTNAME
V_	Validation text	V_LASTNAME
B_	Button text	B_SAVE
M_	Message text	M_NOOFFICERS

*In the case of the data row, normally if only one entity is being referenced a simple dr, dr represents the data row, however in the case of a procedure using multiple tables; the prefix will be followed by the name of the entity.

3.3. Photo Storage

Photos of insurees are captured by the mobile phones and are to be transmitted to the central server by the mobile phone application. In case the mobile network coverage is sufficient, the sending of photographs is done automatically by the mobile phone application (Android). In case there is no connectivity, the photographs could be loaded onto a flash disk or any other media manually. The IMIS application has a folder structure for photo submissions and a separate folder structure for 'consumed' associated photographs.

e.g. C:\XX\YY\SubmittedPhotos
C:\XX\YY\UpdatedPhotos

At the moment of 'updated' photographs with the insure, the photograph will be transferred from the folder 'SubmittedPhotos ' toward the folder 'UpdatedPhotos'. The association process is based upon the actual filename of the photo-image. The process of association takes place when we enroll insurees for the first time or when we recall the insuree page and a positive relationship could be established between the Insuree and a photograph found in the submission folder. This process will be covered in more detail later.

The following convention for the filename:

<Insurance NUMBER>_<ENROLLCODE>_<DDMMYYYY>.XXX

< Insurance NUMBER> = QR Code assigned to the insure
<DDMMYYYY> = Date of photograph taken
<ENROLLCODE> = Enrollment officers code
<XXX> = File extension of the photo image e.g. JPG, PNG etc…

Although any file size is allowed to be copied to the Central server, we should set a maximum size of 200Kb. The file format we suggest is JPEG.

4. IMIS Menu structure and Interface flow

The IMIS application consists of many individual ASPX pages which are initiated via the following controls:

- Menu buttons
- Action buttons
- Hyperlinks

Hereunder we will cover the IMIS menu structure with its submenus and actions.

4.1. Main menu

The options '**Home'** and '**Logout'** are not having sub-menus and will navigate directly toward the following pages:

[Home] → Home.aspx
[Logout] → Logout.aspx
[Insurance Number enquiry] → IMIS.MASTER embedded JQuery/AJAX code

As seen above, the main (top level) menu will also have a few sub-menus:

- Insurees and Policies
- Claims
- Administration
- Tools

4.2. Menu Insurees and Policies

The above menu will have four options which will open web pages:
[Families/Groups] →FindFamily.aspx
[Insurees] →FindInsuree.aspx
[Policies] →FindPolicy.aspx
[Contributions] →FindPremium.aspx

4.3. Menu Claims

The above menu has three options which will open web pages:

[Health Facility Claims] → FindClaims.aspx
[Review] → ClaimOverview.aspx
[Batch Run] → ProcessBatches.aspx

4.4. Menu Administration

This menu has only one submenu as shown above:

- Price Lists

The other options open up webpages as shown below:

[Products] → FindProduct.aspx
[Health Facilities] →FindHealthfacility.aspx
[Medical Items] →FindMedicalItem.aspx
[Medical Services] →FindMedicalService.aspx
[Users] →FindUser.aspx
[Enrolment Assistants] →FindOfficer.aspx
[Claim Administrator] →ClaimAdministrator.aspx
[Payers] →FindPayer.aspx
[Locations] →Locations.aspx

4.5. Menu Price Lists

The options Medical Items and medical Services open up the following pages:

[Medical Items] →FindPriceListMI.aspx
[Medical Services] →FindPriceListMS.aspx

4.6. My Profile

My profile has only one sub menu which is change password, the Change Password sub menu open up ChangePassword.aspx page

4.7. Menu Tools

This menu has no further submenu and would open pages directly.

[Upload Diagnosis List] →UploadICD.aspx
[Policy Renewals] →PolicyRenewals.aspx
[Feedback Prompt] →FeedbackPrompt.aspx
[Extracts] → IMISExtracts.aspx
[Reports] →Reports.aspx
[Utilities] → Utilities.aspx
[Funding] → AddFunding.aspx
[Email Settings] →EmailSettings.aspx

5. Functional Area Description

This chapter will cover the actual screen designs and screen flow of the IMIS application. Each interface will be presented with the following:

- Actual user interface design
- Interface object name
- Menu-Link , redirections and /or Action Link(s)
- Reference to Use Cases
- Brief description of interface
- Class entities (top level)

Some features are to be found on several interfaces. These mechanisms will be described generally rather than for each interface separately.

5.1. Generic interface features

The following features could be found on several interfaces:

- Add, Edit, Delete and Save buttons
- Cancel button
- 'Historical' records
- Pages tab
- Data grids
- Hyperlinks
- Multi Language aspects
- Session expiry
- Mandatory fields
- General messages and current menu path

5.1.1. Add, Edit, Delete, Save and buttons

The cancel button is always found in the lower section of the screen on the right corner side. The cancel button will cancel any entry on a screen (if any) and will subsequently navigate back to the actual caller (menu or screen).
All interfaces activated under the **administration menu** (except Locations) will have the 'add', 'edit' and 'delete' buttons in the lower section of the screen as shown below:

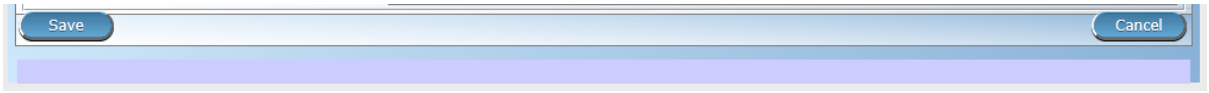
The add button will generally open up a new page for the entry of a new record.
All entries are empty but the system will automatically populate entries in case only one option is available (e.g. District or product. Etc)
Similar wise the 'edit' button will open up a new page for but all fields are loaded with the information of the selected record.
The delete button will display a message prompting the operator to confirm the delete action. On confirmation, the record will be 'logically' deleted from the system by flagging the ValidityTo field with a timestamp. The 'datagrid' will thereafter refresh.

The interfaces for 'family/group overview' and 'locations' have the 'add', 'edit' and 'delete' buttons elsewhere as shown below:

Add-button → green '+' (plus)
Edit-button → yellow pencil
Delete → red crossed symbol

The actions on clicking these buttons are similar as previously described for the buttons on the lower section (administration menu).

The save button most of the time is found in the lower section in the left corner as shown below:



By clicking the save button, the record will be inserted or updated after data validation passed (as peruse cases). The save button will bring you back to the calling interface.

5.1.2. User Security

User security in IMIS is built around pre-defined roles:

- Enrolment Assistant
- Manager
- Accountant
- Clerk
- Medical Officer
- Scheme Administrator
- IMIS Administrator
- Receptionist
- Claim Administrator
- Claim Contributor

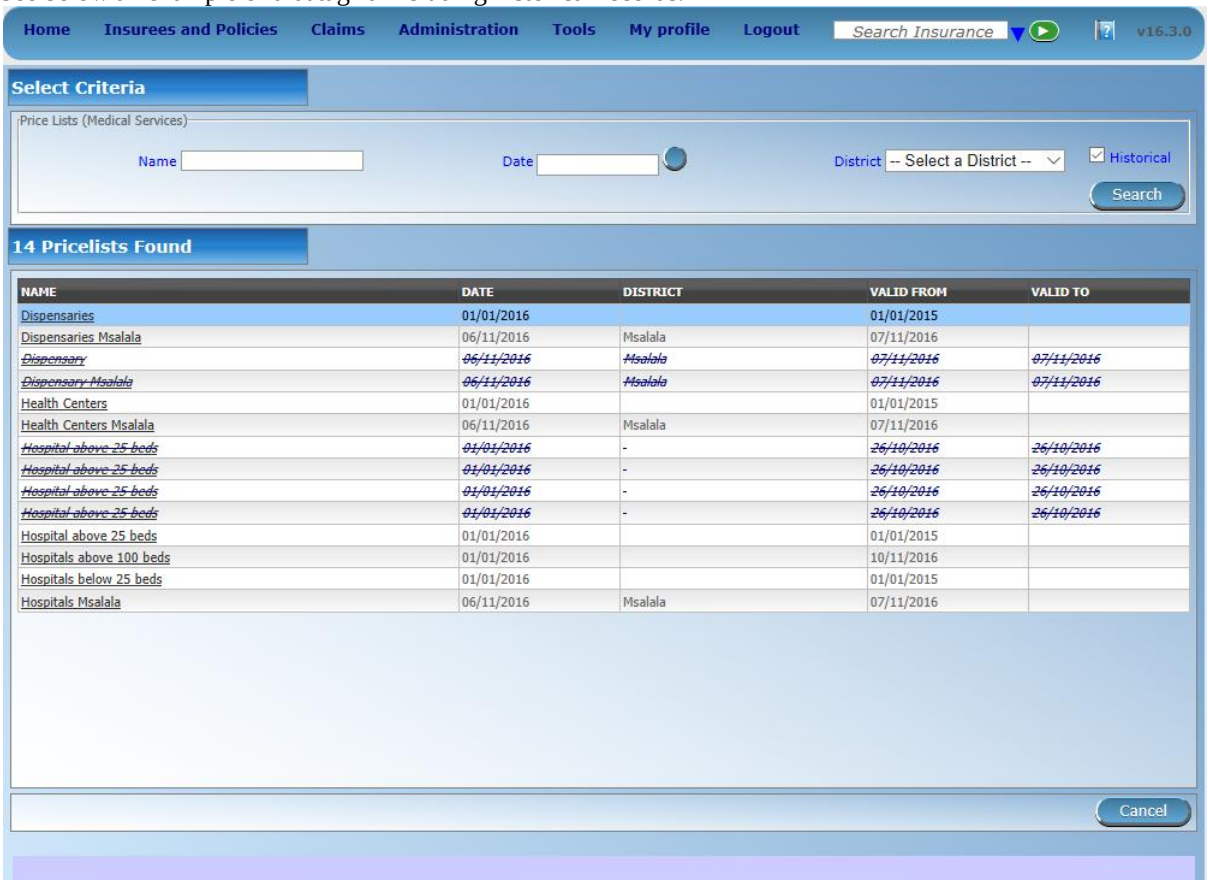
At the moment of logging in, IMIS it fetch security value that will relate the operator to one or more security roles. Depending on the role(s) of the operator, menu-options and screen functionality will be enabled for use or disabled if no rights were assigned.

All actions in the system are audited by storing the UserID with the information. An audit report is available in the reports section.

5.1.3. 'Historical' records

IMIS register all its records with timestamps for the validity of the record. All records that are flagged with a ValidityTo date have been updated or deleted via the IMIS interfaces or services.
To be able to view the different versions of a record (e.g. an Insuree that had overtime several changes in its information) one would be able to click the 'Historical' checkbox. By clicking this box, the datagrid will prevail all versions of the records and will strike-out the records that are not current. In this 'historic' mode one cannot change information but is able to view the information by clicking the view button in the lower section of the screen. The full record contents of the 'historic' record will be shown.

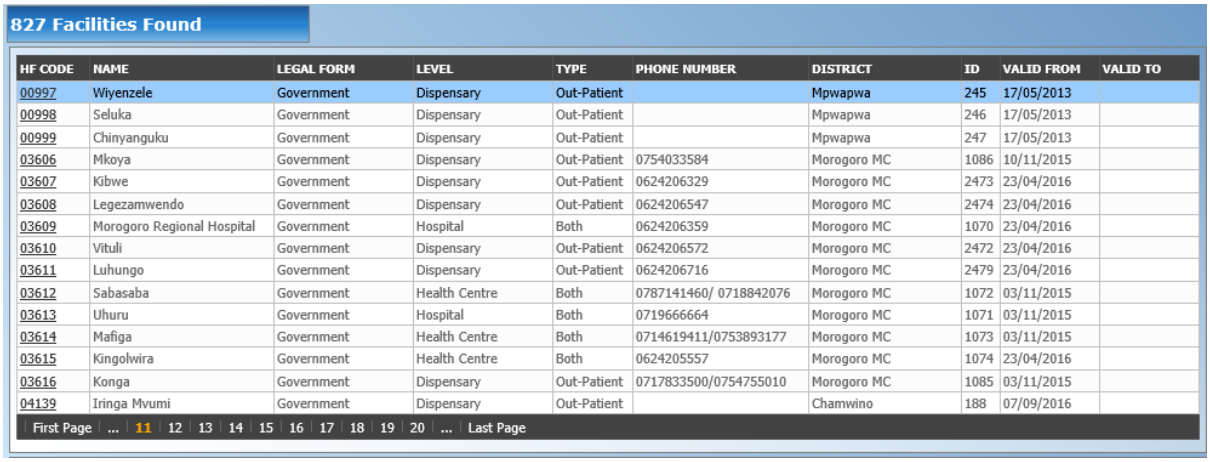
See below an example of a datagrid including historical records.



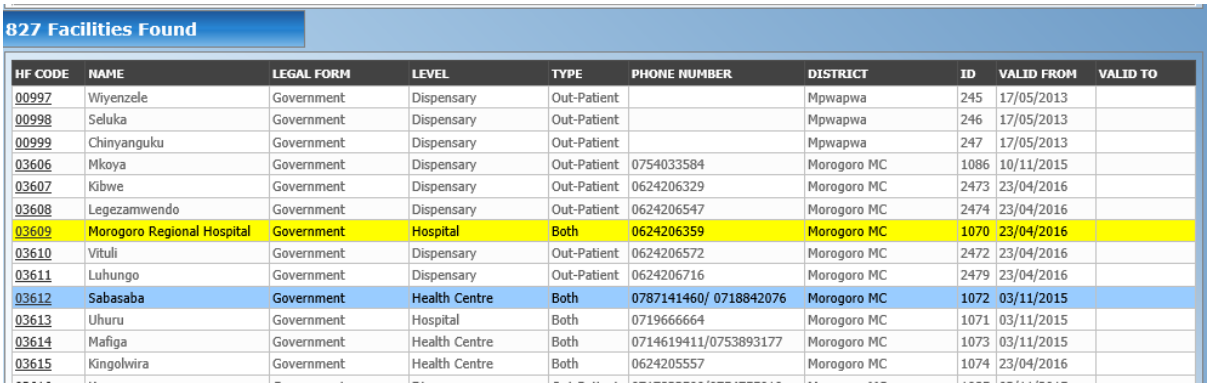
5.1.4. Data grids, hyperlinks and pages

As seen in the previous chapter (menus), most of the menu options in IMIS will open up search interfaces. Each search interface will display the result of the search action (after clicking the search button) within a data grid.

Depending on the expected data load, data is already populated in the grids whilst other search screens will only provide information in the grid after setting the criteria and clicking 'search'. For example, the health facilities search screen will automatically display all health facilities available applicable to the current user. The Insuree search page would initially not show any data in the grid as too many records would be unnecessarily loaded. In most of the search screens one can find in the bottom of the screen a navigation bar for pages as shown below:



In the example above, 15 facilities (rows) are shown in the datagrid. However, there are many pages available with health facilities that match the search criteria. To see more facilities one could click on any of the pages available or navigating to the last or first page. After clicking on another page number, the grid will reload a new set of entries that satisfied the search criteria.



827 Facilities Found

HF CODE	NAME	LEGAL FORM	LEVEL	TYPE	PHONE NUMBER	DISTRICT	ID	VALID FROM	VALID TO
00997	Wiyenzele	Government	Dispensary	Out-Patient		Mpwapwa	245	17/05/2013	
00998	Seluka	Government	Dispensary	Out-Patient		Mpwapwa	246	17/05/2013	
00999	Chinyanguku	Government	Dispensary	Out-Patient		Mpwapwa	247	17/05/2013	
03606	Mkoya	Government	Dispensary	Out-Patient	0754033584	Morogoro MC	1086	10/11/2015	
03607	Kibwe	Government	Dispensary	Out-Patient	0624206329	Morogoro MC	2473	23/04/2016	
03608	Legezamwendo	Government	Dispensary	Out-Patient	0624206547	Morogoro MC	2474	23/04/2016	
03609	Morogoro Regional Hospital	Government	Hospital	Both	0624206359	Morogoro MC	1070	23/04/2016	
03610	Vituli	Government	Dispensary	Out-Patient	0624206572	Morogoro MC	2472	23/04/2016	
03611	Luhungo	Government	Dispensary	Out-Patient	0624206716	Morogoro MC	2479	23/04/2016	
03612	Sabasaba	Government	Health Centre	Both	0787141460/ 0718842076	Morogoro MC	1072	03/11/2015	
03613	Uhuru	Government	Hospital	Both	0719666664	Morogoro MC	1071	03/11/2015	
03614	Mafiga	Government	Health Centre	Both	0714619411/0753893177	Morogoro MC	1073	03/11/2015	
03615	Kingolwira	Government	Health Centre	Both	0624205557	Morogoro MC	1074	23/04/2016	

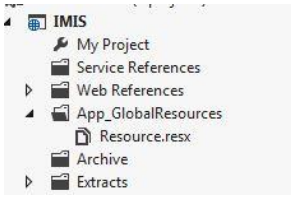
All the data grids with a yellow (highlighted) row for indicating your current mouse position in the grid. By moving the mouse up and down the yellow row will follow the mouse pointer. By clicking a row, the row gets selected. The selected row is always indicated by the color blue. In case we need to modify/view or delete a record, one should first select the row by clicking on it (marking it blue) and thereafter clicking the desired action (Edit, Delete, and View)

Most grids have the first column as a Hyperlink. Instead of selecting the record (marking blue) and clicking the desired action button, one could alternatively click directly on the hyperlink to 'Edit' the ('clicked on') record.

Some columns, such as expiry date, valid from and to, will have sort order mechanisms built in to allow the operator to sort the grid contents ascending or descending. The sort order change will be performed after clicking the column header.

5.1.5.Multi Language aspects

IMIS developed in multi-language: English and the other language which will be provided with IMIS Administration. The manner of changing from one language to another is depending on the user profile. The default language will be English. English will be an Interface Culture used for loading labels (menu and pages), messaging, grid headers etc…
The actual language elements are kept in so called resource files. Each individual language element can be found in these resource files.



As seen above, we currently only have one resource file included in the project: Resource.resx (default English). Additional languages could be added by simply adding an extra resource file with the correct translations and allowing the user to select this new language via the interface of the user definition.

Each resource file has a similar structure and holds all translated elements. See below 2 separate snippets of the resource file for Swahili.

L_OTHERNAMES	Majina mengine
L_OUTPATIENT	Mgonjwa aliyetoka
L_PASSPORT	Namba ya pasipoti
L_PASSWORD	Namba ya kuingilia
L_PAYER	Walipaji
L_PHONE	Namba ya simu

The above section is an example of labels, starting with 'L_'.
The section below shows some translation for validation messages starting with 'V_'

V_GENDER	Naomba uchague jinsia
V_LASTNAME	Naomba uingie kwenye jina la mwisho
V_MARITAL	Naomba uchague aina ya jinsia
V_OTHERNAMES	Naomba uingie majina mengine

Also for data driven some of drop down has been added with the multi language scenario example of the drop down is Relation, the figure below show the table structure where the data stored. On selection the system will select according to user language

RelationId	Relation	SortOrder	AltLanguage
1	Father	1	Baba
2	Brother	2	Kaka
3	Sister	3	Dada
4	Cousin	4	Binamu
5	Uncle	5	Mjomba

The process of obtaining the correct translation from resource is at HTML code level in each page as shown below for the label and validation text for the field 'other names'.

```
<td class="FormLabel">
<asp:Label
ID="L_OTHERNAMES"
runat="server"
Text='<%$ Resources:Resource,L_OTHERNAMES %>'>
</asp:Label>
</td>
<td class ="DataEntry">
<asp:TextBox ID="txtOtherNames" runat="server" Width="150px">
</asp:TextBox>
</td>
<td>
<asp:RequiredFieldValidator
ID="RequiredFieldOtherNames" runat="server"
ControlToValidate="txtOtherNames"
SetFocusOnError="True"
ValidationGroup="check"
Text='<%$ Resources:Resource,V_OTHERNAMES %>'>
</asp:RequiredFieldValidator>
    <asp:RegularExpressionValidator ID="RegularExpressionValidator2" runat="server"
        ControlToValidate="txtOtherNames" ErrorMessage="Only alphabets are allowed."
        ValidationExpression="^[a-zA-Z]+$" ValidationGroup="check"></asp:RegularExpressionValidator>
</td>
```

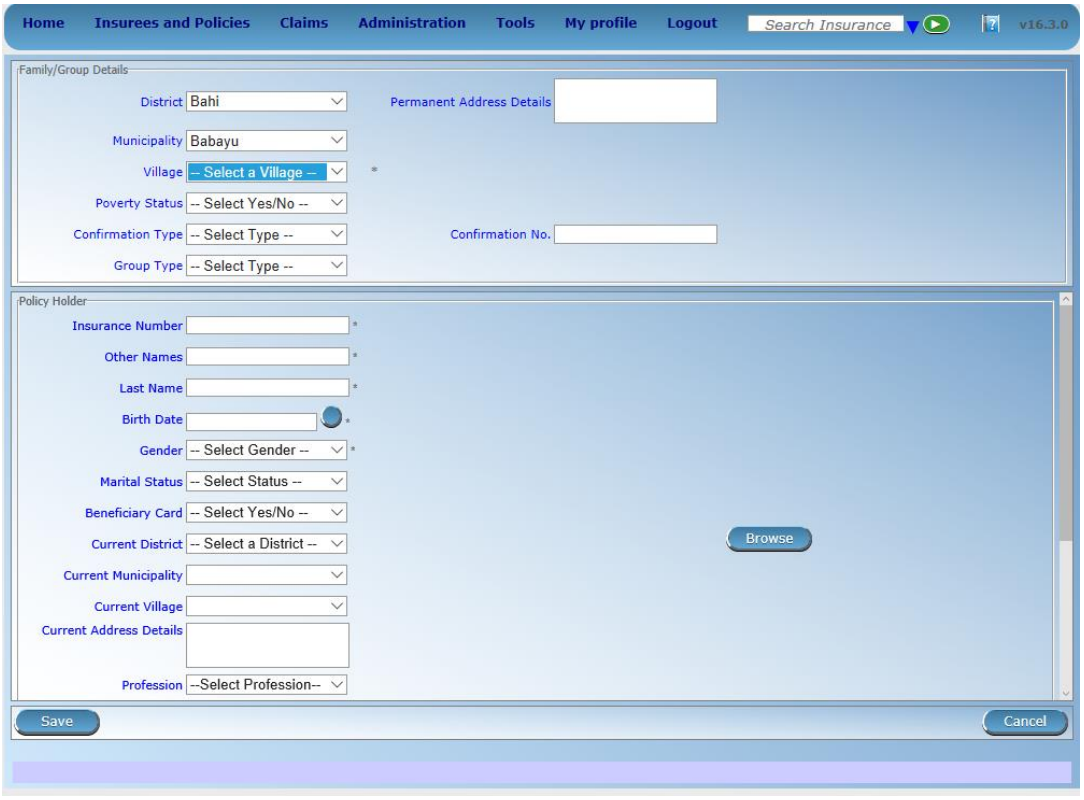
5.1.6.Session expiry

The session expiry has been set at 20 minutes as a default. However, this can be changed to another time interval as required via the web.config file and IIS configuration.
If a session expires the operator will have to login once again.

5.1.7.Mandatory fields

While working with an entry (adding/editing) screens, the operator will see directly which data is incomplete or not correctly entered. The message is shown on the screen on the right side of the actual control in 'red' marked text. This text will automatically disappear at the moment the text is entered.
This first level validation takes place in the presentation tier on the local machine (client side). However more validations will take place in the Business Logic Layer in private functions.

An example of client side validation is shown below:



5.1.8. Status bar and Popup

The 'purple' status bar on the bottom of the screen will act as a message bar that will always show the current menu-route. For example if one would have clicked the option "Facilities" under the Administration menu and thereafter selected a health facility for editing, the status bar will show: Administration-Health facilities-Modify Health facility.
The status bar will also be used for popup messages such as 'Saved successfully' these messages will disappear within 10 seconds. The menu path will again, thereafter, be shown in this status bar. In case of more important messages, IMIS will provide popup messages.

5.2. Web pages

The presentation Tier of IMIS will consist of several web pages (aspx pages) that will contain different programming techniques such as:

- HTML
- AJAX
- JQUERY
- JSCRIPT

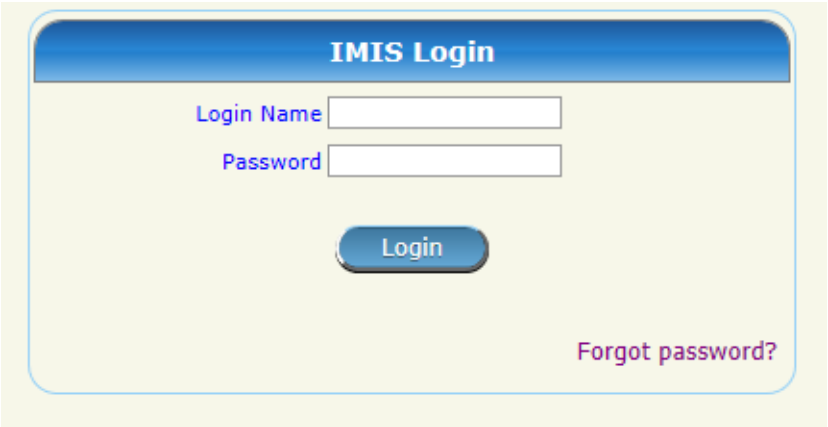
The web pages are based upon Style sheets. This will enforce uniformity throughout the interfaces and will simplify design changes in future amendments. The web.config file will host the connection string for connectivity to the database and will also be used to host certain application defaults such as folder locations for photographs and any interfacing with external applications such as the mobile phone (file transfers) and EPICOR.

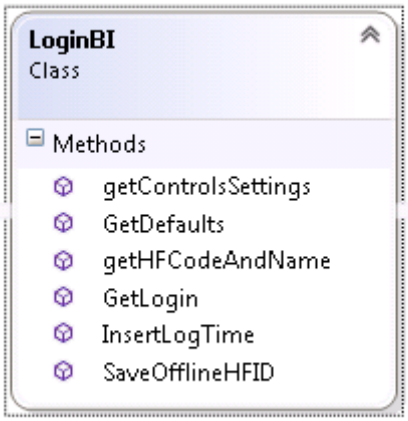
In the following paragraphs we will cover the interfaces:

- AddFunding.aspx
- ChangeFamily.aspx
- ChangePassword.aspx
- Claim.apx
- ClaimAdministrator.aspx
- claimfeedback.aspx
- ClaimOverView.aspx
- ClaimReview.aspx
- Default.aspx
- Download.aspx
- EmailSettings.aspx
- Error.html
- FaindClaim.aspx
- Family.aspx
- FeedbackPrompt.aspx
- FindClaimAdministrator.aspx
- FindFamily.aspx
- FindHealthFacility.aspx
- FindInsuree.aspx
- FindMedicalItem.aspx
- FindMedicalService.aspx
- FindOfficer.aspx
- FindPayer.aspx
- FindPolicy.aspx
- FindPremium.aspx
- FindPriceListMI.aspx
- FindPriceListMS.aspx
- FindProduct.aspx
- FindUser.aspx

- ForgotPassword.aspx
- General.vb
- Global.asax
- HealthFacility.aspx
- Home.aspx
- IMIS.MASTER (used for Menu and Quick Inquiry)
- Imis_Gen.vb
- IMISExtracts.aspx
- Insuree.aspx
- Locations.aspx
- Logout.aspx
- MedicalItem.aspx
- MedicalService.aspx
- MoveLocations.aspx
- Nojs.html
- Officer.aspx
- OverviewFamily.aspx
- Payer.aspx
- Policy.aspx
- PolicyRenewals.aspx
- Premium.aspx
- PremiumCollection.aspx
- pREMIUMdistribution.aspx
- PriceListMI.aspx
- PriceListMS.aspx
- ProcessBatches.aspx
- Product.aspx
- Redirect.ASPX
- Redirect.htm
- Report.aspx
- Reports.aspx
- UploadICD.aspx
- UploadICD.aspx
- User.aspx
- Utilities.aspx
- WebConfig

5.2.1.Default.aspx

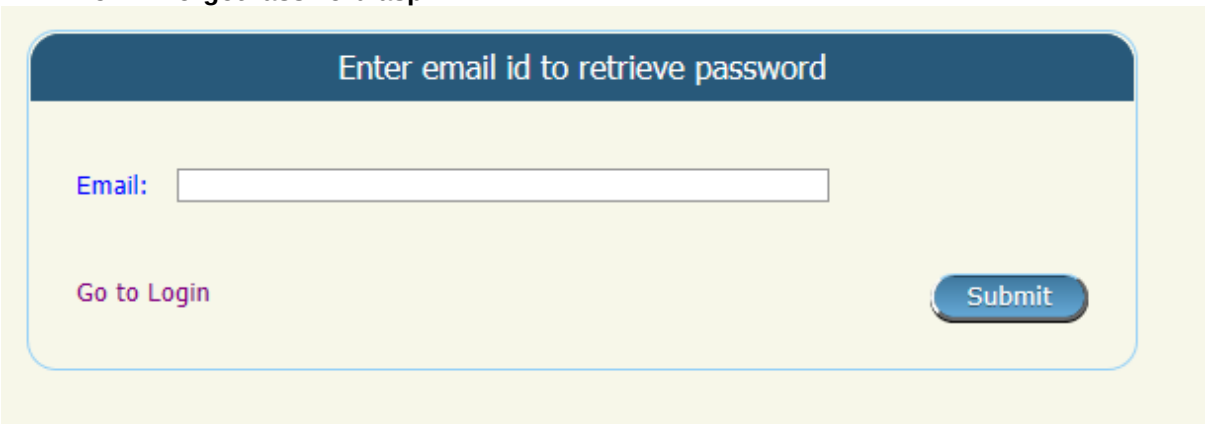


Interface object name:	***Default.aspx***
Menu path:	***Logout***
Hyperlinks/Redirections:	***Opens when IMIS Starts***
Action Button:	
Use case Reference	***N/A***
Class Diagram	

Brief description:

This interface opens up when the user starts IMIS. A similar design will be used for the Login on mobile phones in case we use web-pages within the phone App. After successful login procedure the use is taken to the home page. Further to this, at this moment we will switch the language to Swahili or English using the so called Interface Cultures within the application and making use of the required resource file (English/Swahili).
For the mobile phone only English will be used.

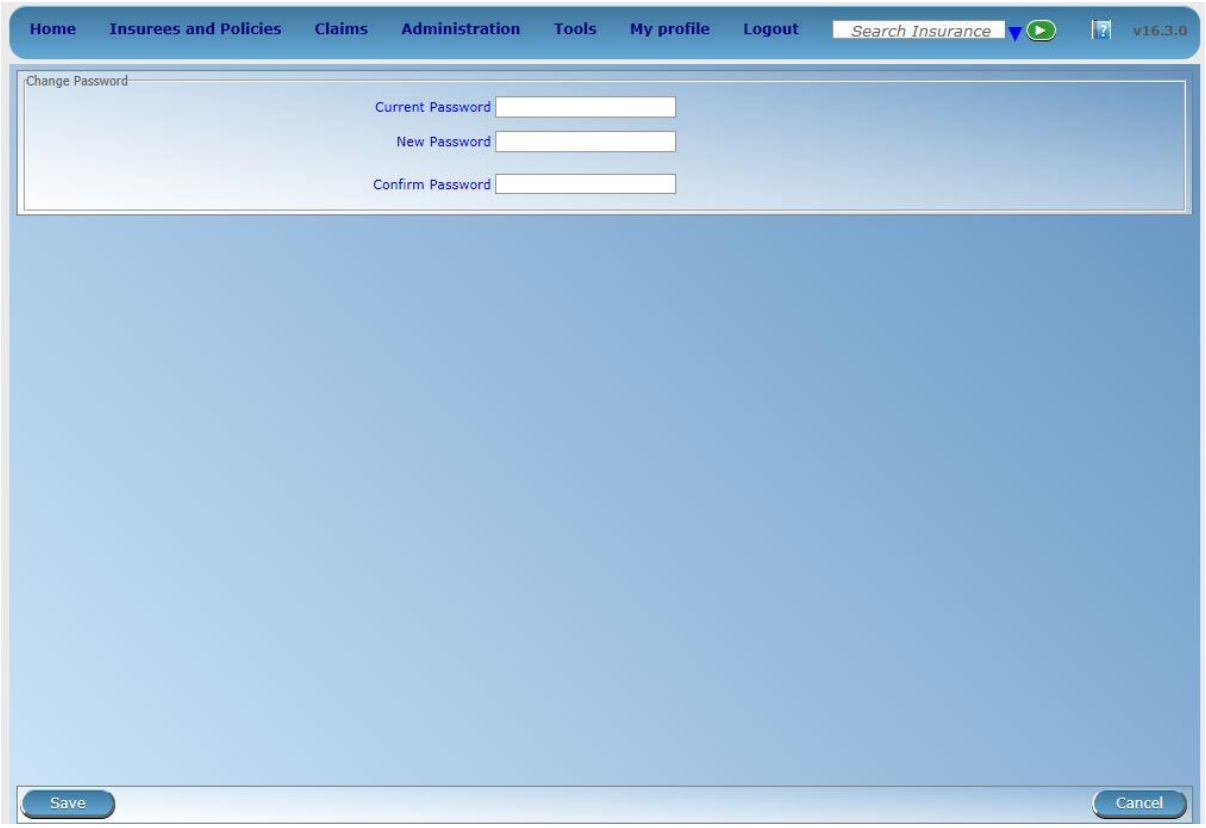
5.2.2.ForgotPassword.aspx



Interface object name:	*ForgotPassword.aspx*
Menu path:	*N/A*
Hyperlinks/Redirections:	*Open after click Forgot password*
Action Button:	*Submit Email address*
Use case Reference	*N/A*
Class Diagram	

Brief description:
This interface opens up when the user click Forgot password button in default.aspx page. The interface used to recover user login password, user has to submit the email address then the system will verify the email and send the correct password.

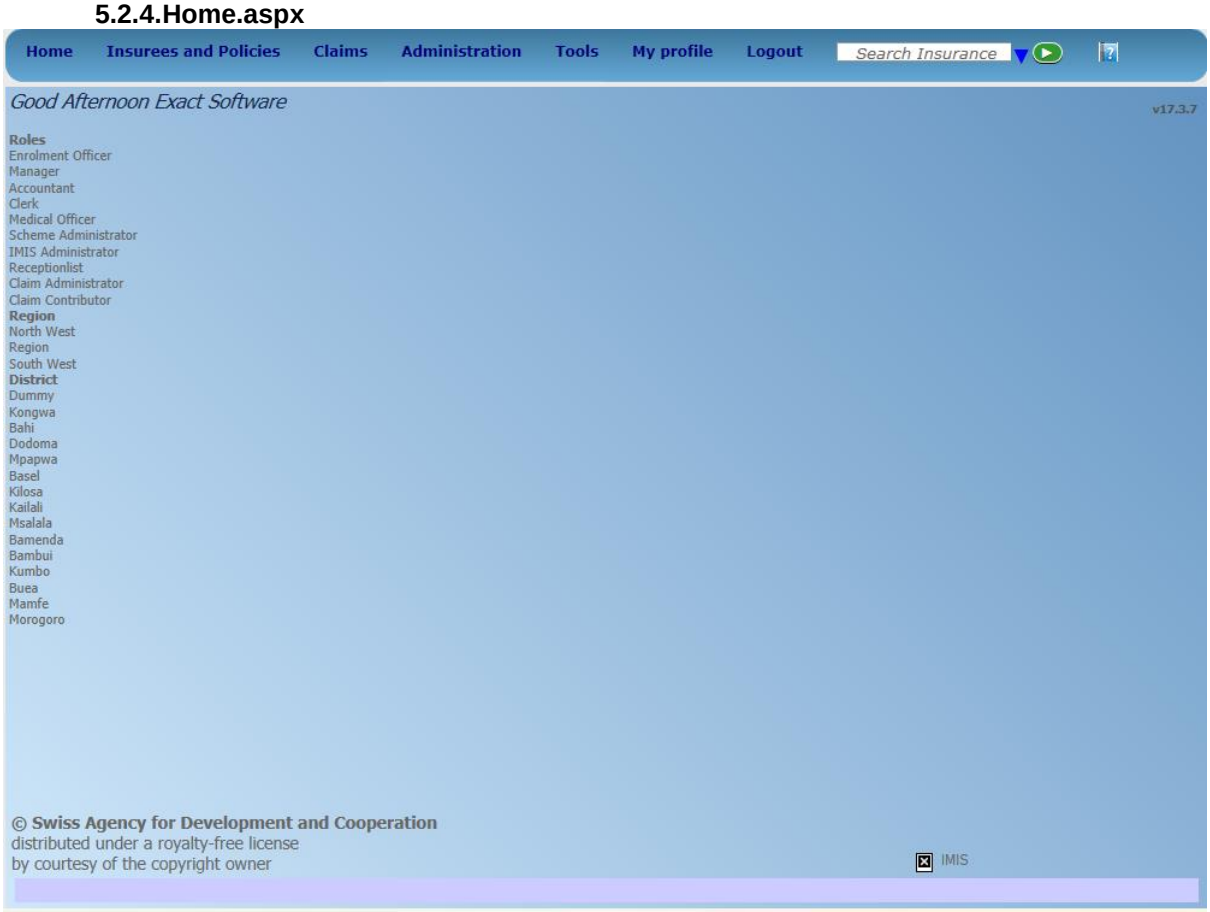
5.2.3.ChangePassword.aspx



Interface object name:	*ChangePasswors.aspx*
Menu path:	*Change Password (Under My Profile Menu)*
Hyperlinks/Redirections:	*N/A*
Action Button:	*N/A*
Use case Reference	*N/A*
Class Diagram	

Brief description:
This interface opens up when the user click Change Password. The interface used to change user password by enter the current password and the new password.

5.2.4.Home.aspx



Interface object name:	*Home.aspx*
Menu path:	*Home*
Hyperlinks/Redirections:	*Opens after logging in successfully*
Action Button:	*Initiated via all cancel buttons on find screens(called via top menu calls)*
Use case Reference	*N/A*
Class Diagram	

Brief description:

This interface opens up when the user has successfully logged in. The menu will show in the language of the user.

5.2.5.Logout.aspx

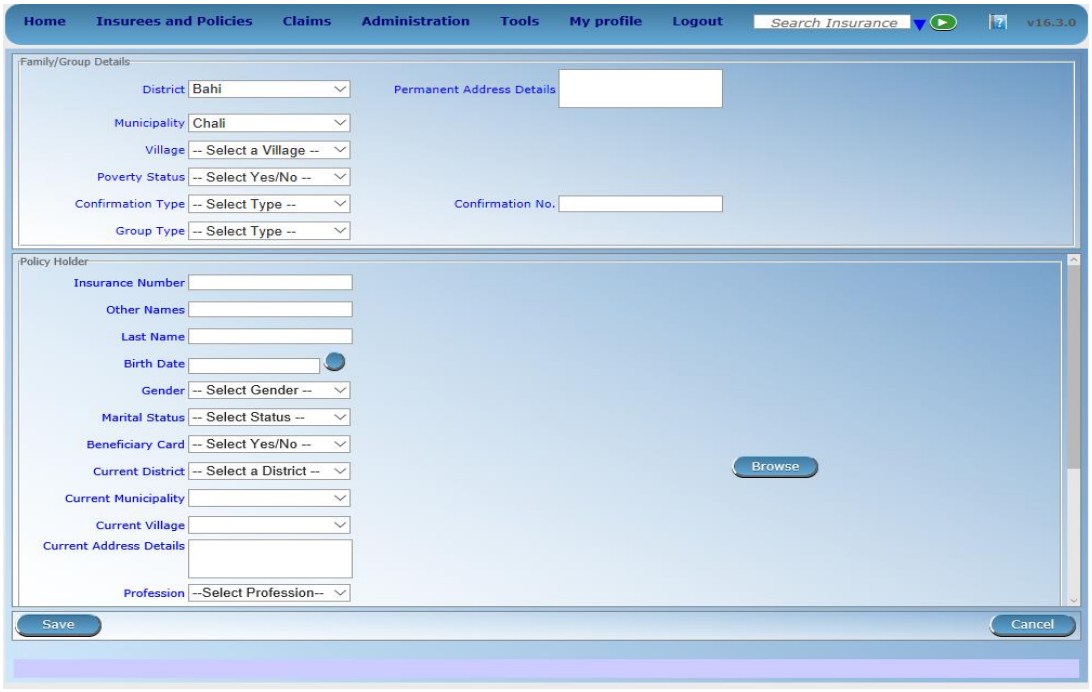
NO DESIGN APPLICABLE

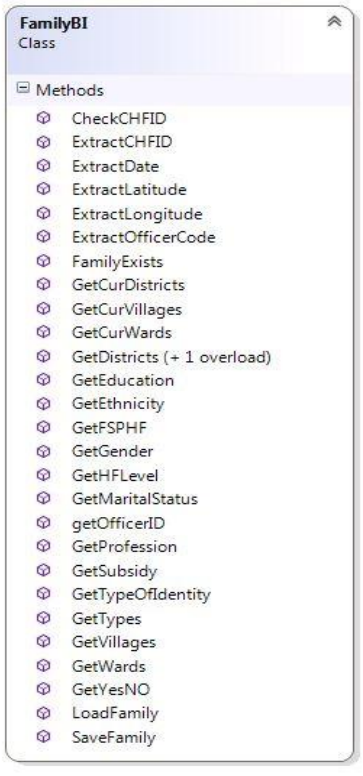
Interface object name:	*Logout.aspx*
Menu path:	*Logout*
Hyperlinks/Redirections:	
Action Button:	
Use case Reference	*N/A*
Class Diagram	*N/A*

Brief description:

This interface opens up when the user has clicked the Logout option on the top menu. The open sessions will be terminated and this page redirects automatically to the Login page: Defaults.aspx

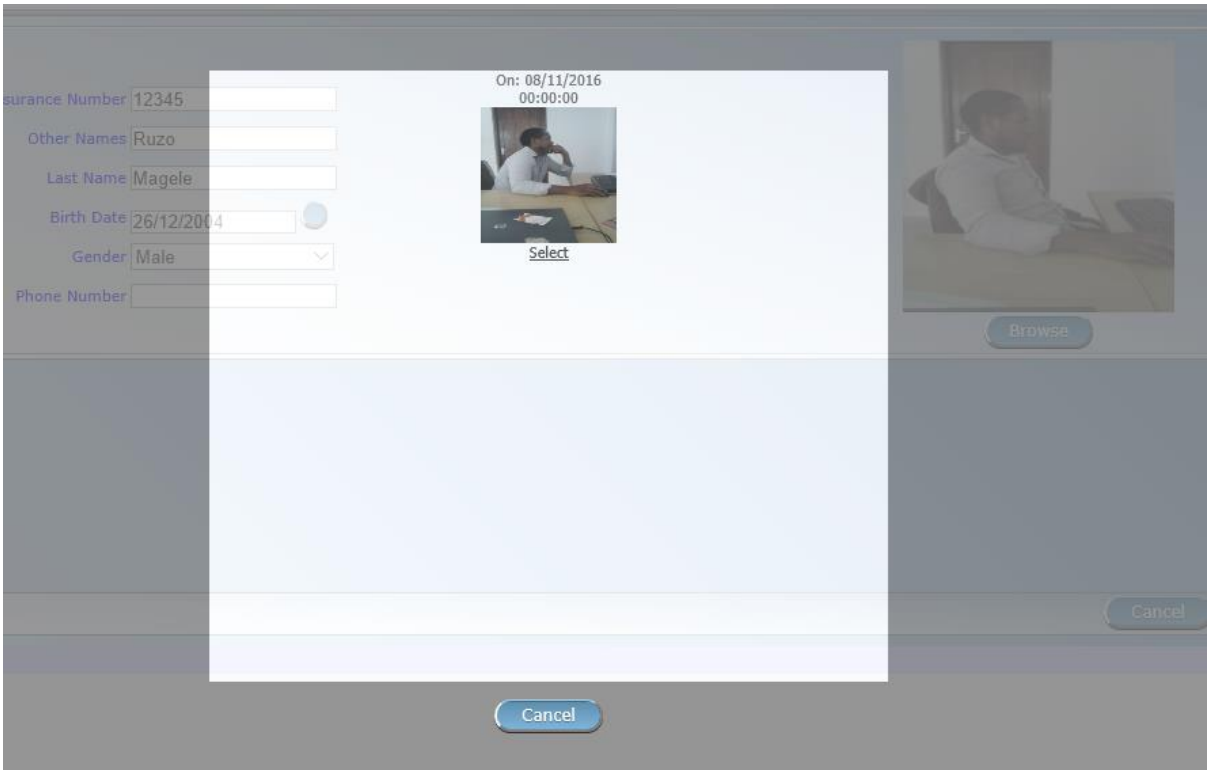
5.2.6.Family.aspx



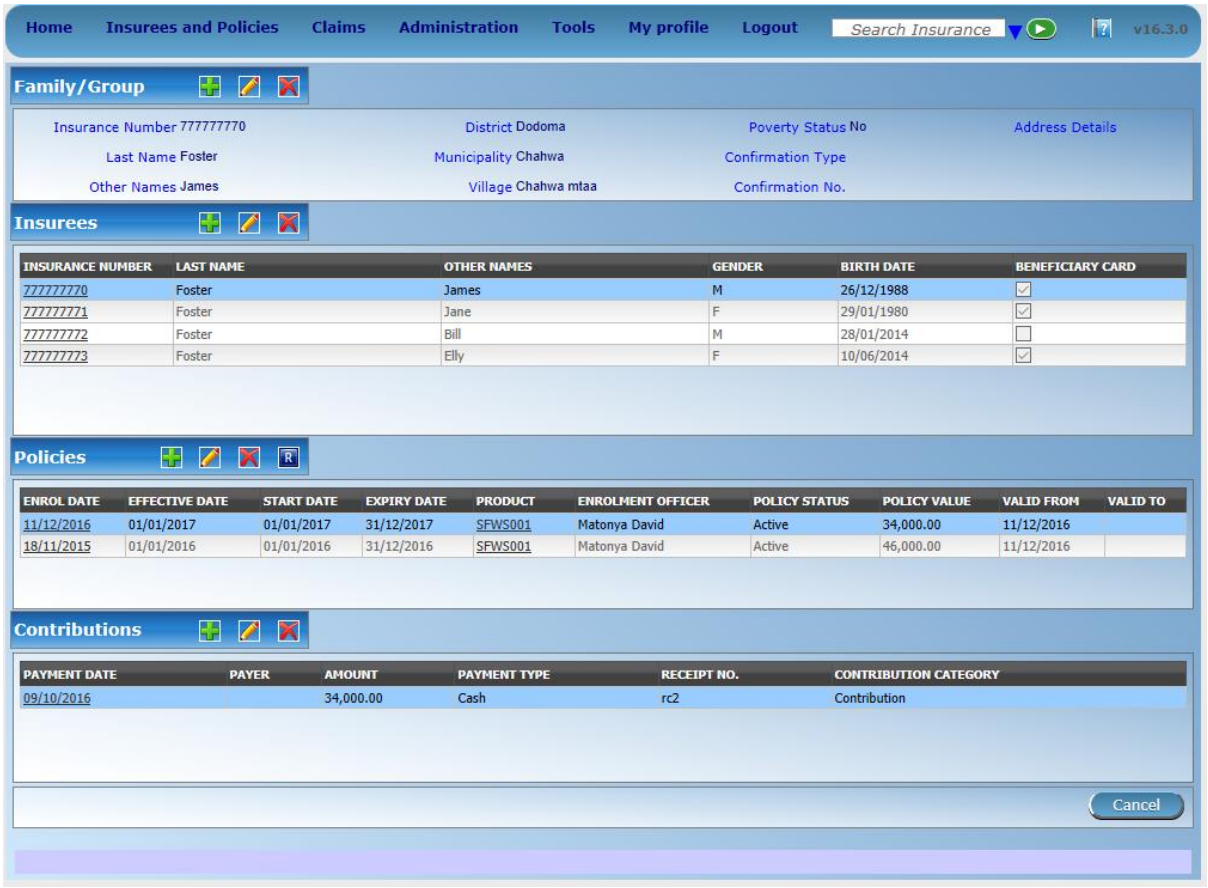
Interface object name:	*Family.aspx*
Menu path:	*Add Family/Group*
Hyperlinks/Redirections:	
Action Button:	*OverviewFamily.aspx (Add button in family/group section)*
Use case Reference	*5.2.1*
Class Diagram	

Brief description:

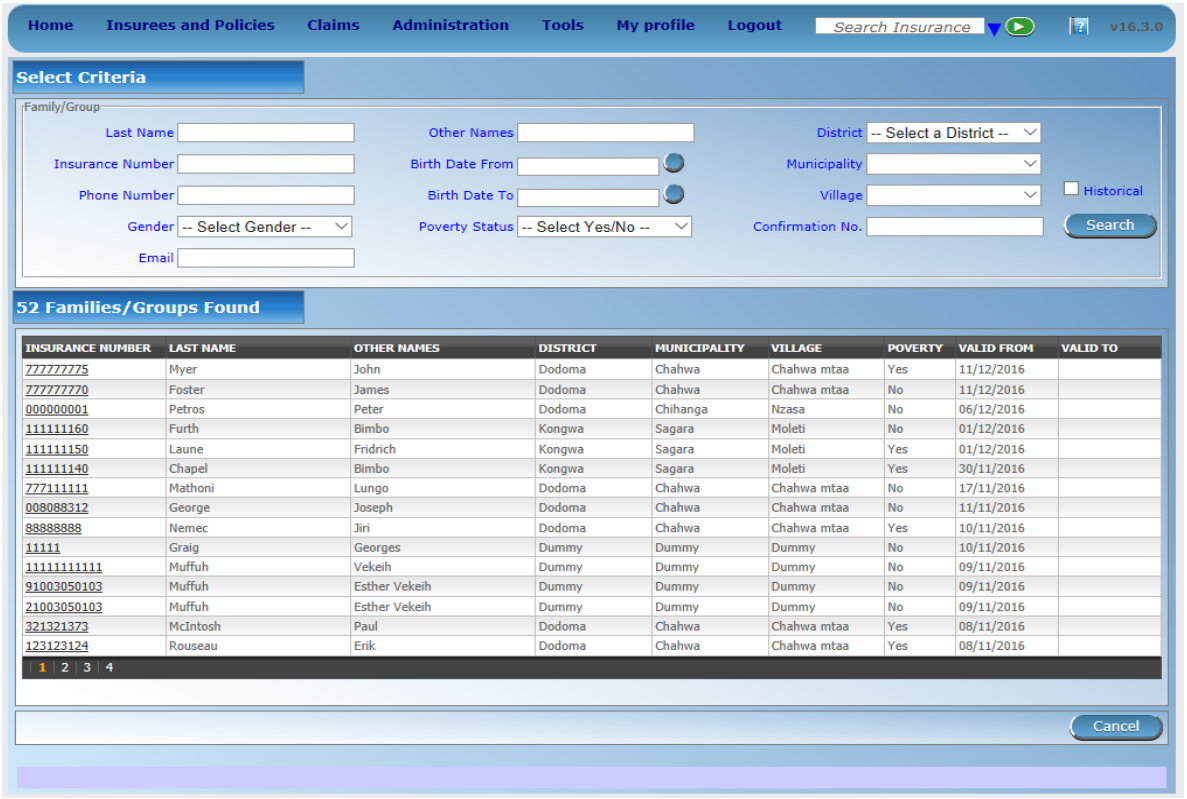
This interface opens up when the user has clicked the Add family/group button option on the top menu (Insurees and Policies). This interface is used to add new families into IMIS with the Head of the family/group. When entering the Insurance Number automatically the system queries the folder for submitted photos and will try to associate the photo by using the photo naming convention (discussed later).
In case one photo is available the photo will be automatically presented as shown in the interface above. In case there would be more than one photograph, a browser window will open (shown below) and the operator could choose the correct photo.

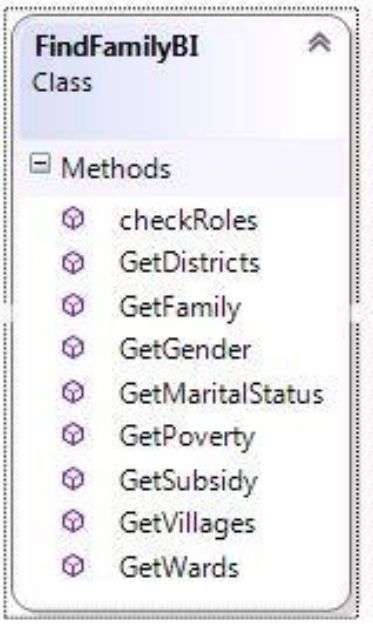


When clicking the 'save' button' first the family/group record is inserted and subsequently the insuree record (as head of family/group). Furthermore, the photograph will be transferred to a new folder for uploaded photos and a record will be inserted in the photo table (if association was successful).
After all saving processes are completed, the user will be directed to the page for the family/group overview as shown below for further adding dependants and/or policies/contributions:



5.2.7.FindFamily.aspx

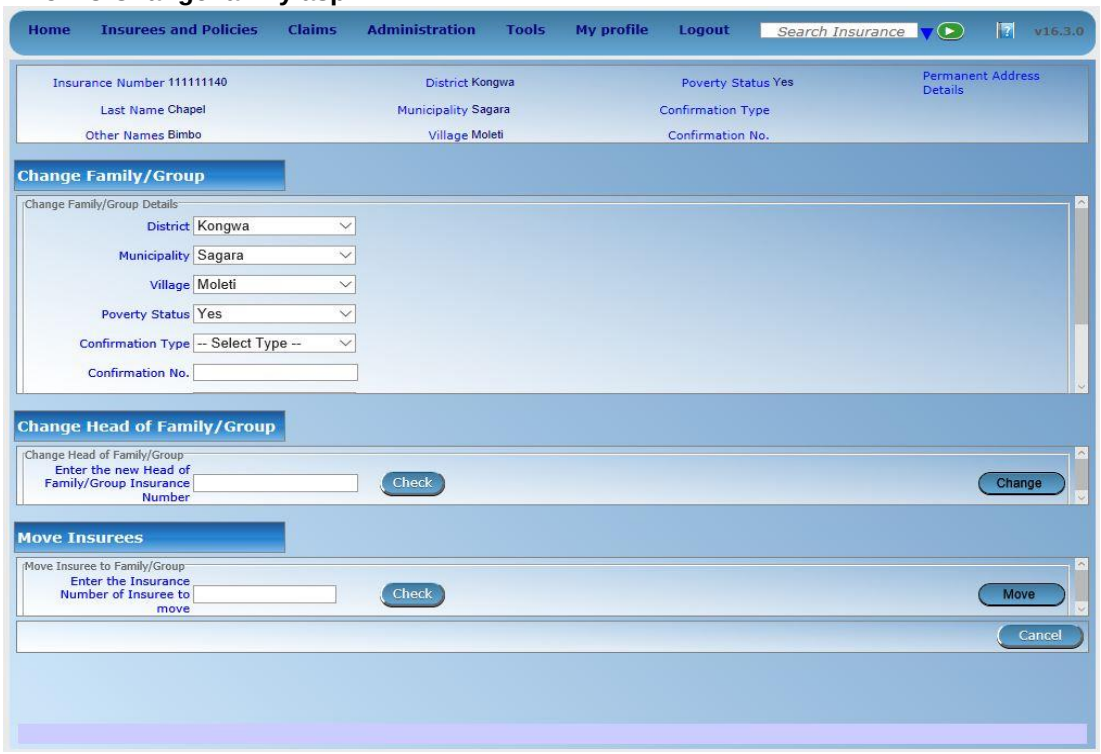


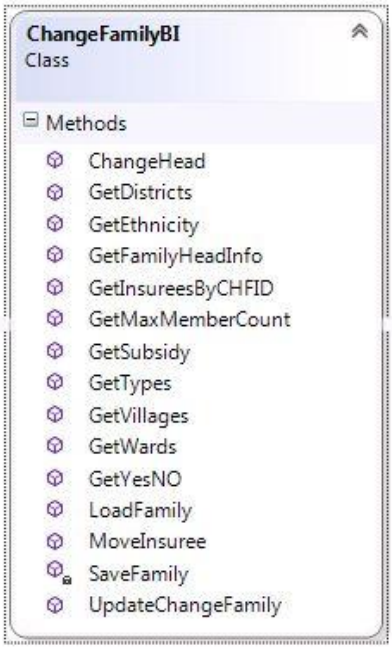
Interface object name:	*FindFamily.aspx*
Menu path:	*Families*
Hyperlinks/Redirections:	
Action Button:	*OverviewFamily.aspx (Cancel button),*
Use case Reference	*5.2.5*
Class Diagram	

Brief description:

This interface opens up when the user has clicked the Families button option on the top menu (Insurees and Policies). This interface is used to search for families. The operator can enter the search criteria and click the search button. All records satisfying the criteria will appear. The Insurance Number hyperlink (Head of Family/Group) in the grid will open up the family/group overview page. Further operations will take action from this overview page.

5.2.8.ChangeFamily.aspx



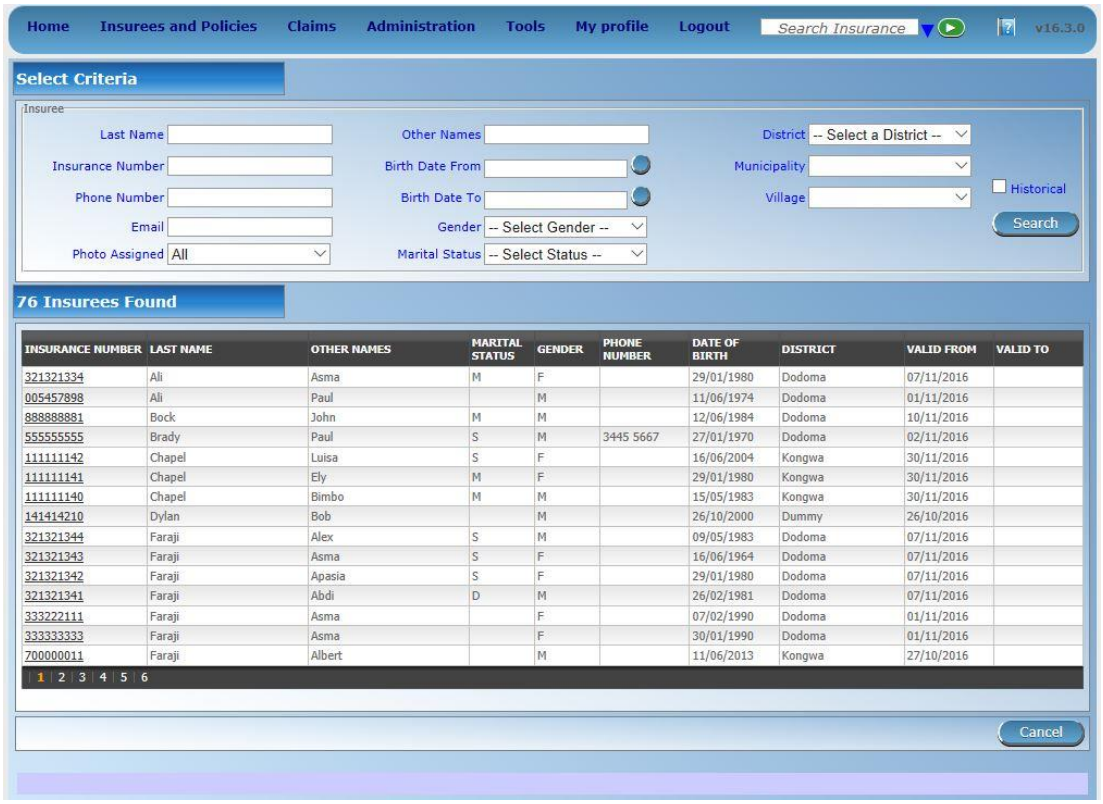
Interface object name:	*ChangeFamily.aspx*
Menu path:	
Hyperlinks/Redirections:	
Action Button:	*OverviewFamily.aspx (Edit button in family/group section)*
Use case Reference	*5.2.9 (Modify Family/Group)* *5.2.13 (Change Head of Family/Group)* *5.2.14 (Move Insuree)*
Class Diagram	

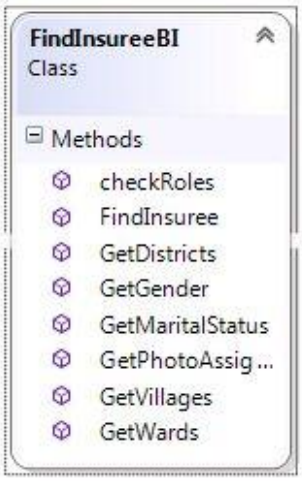
Brief description:

This interface opens up when the user has clicked the Edit button (family/group section) in the family/group overview page.
The operator can use this interface for updating the Family/Group information, changing the head of Family/Group and moving an Insuree toward the current family/group selected.

The Insurance Number to be set as the new head of family/group or to be moved toward the current selected family/group will be entered into the Insurance Number fields. By clicking the 'Change' or 'Move' button, the insuree will subsequently, after validating as per 'use-cases', be inserted in the insuree table (archiving) then updated (current record). Also the family table (change head of family/group) will be updated. (For audit record purposes only).

5.2.9.FindInsuree.aspx

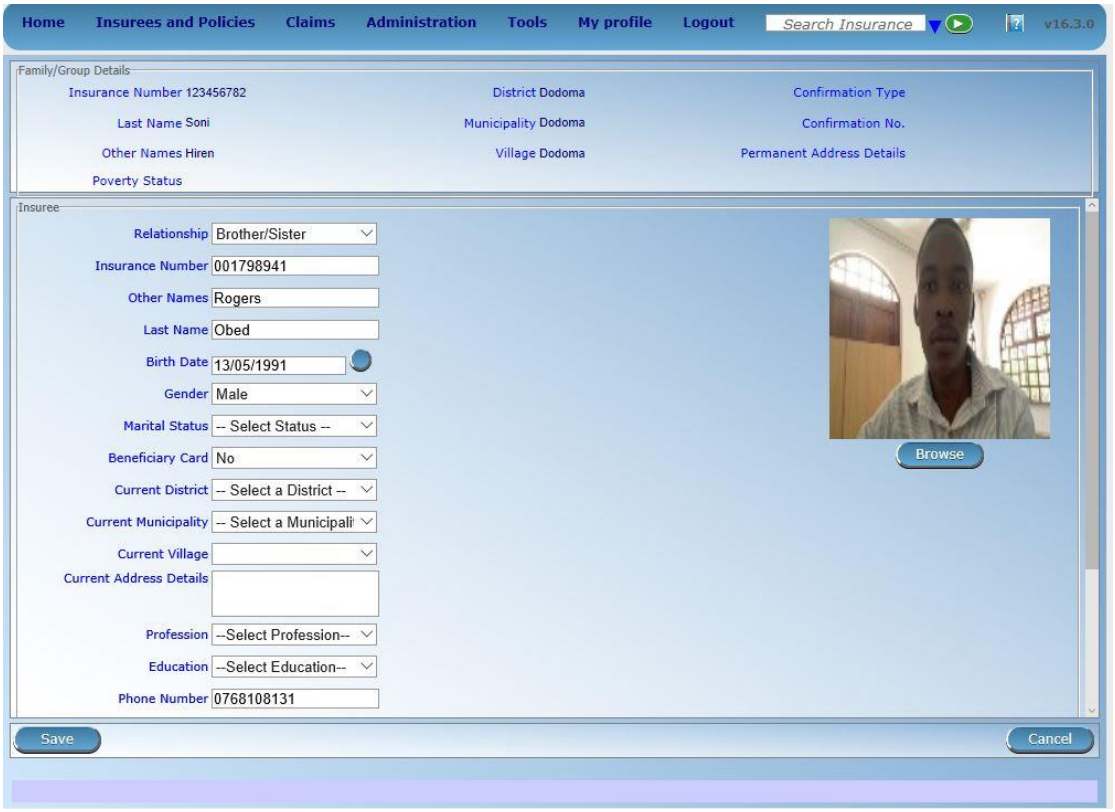


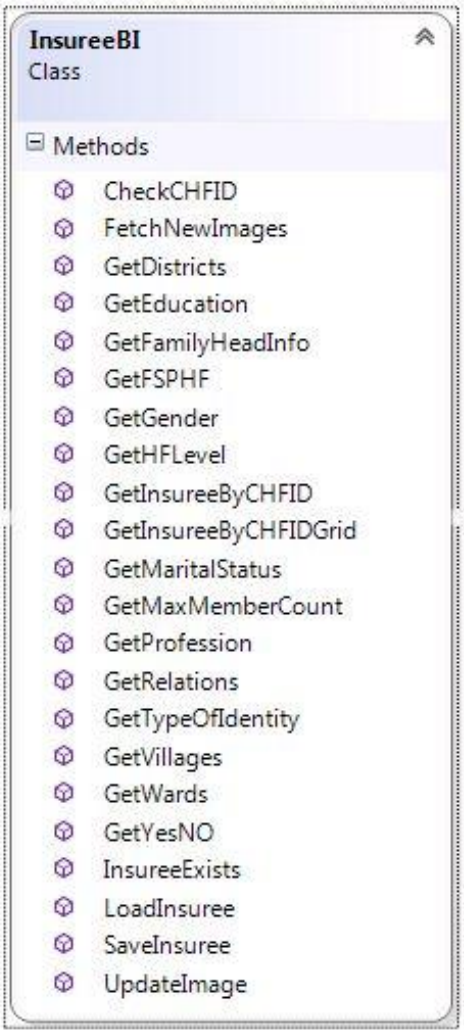
Interface object name:	*FindInsuree.aspx*
Menu path:	*Insurees*
Hyperlinks/Redirections:	
Action Button:	*OverviewFamily.aspx (Cancel button)*
Use case Reference	*5.2.6*
Class Diagram	

Brief description:

This interface opens up when the user has clicked the Insurees button option on the top menu (Insurees and Policies). This interface is used to search for Insurees. The operator can enter the search criteria and click the search button. All records satisfying the criteria will appear. The Insurance Number hyperlink (Insuree) in the grid will open up the family/group overview page. Further operations on the insuree will take action from this overview page.

5.2.10. Insuree.aspx

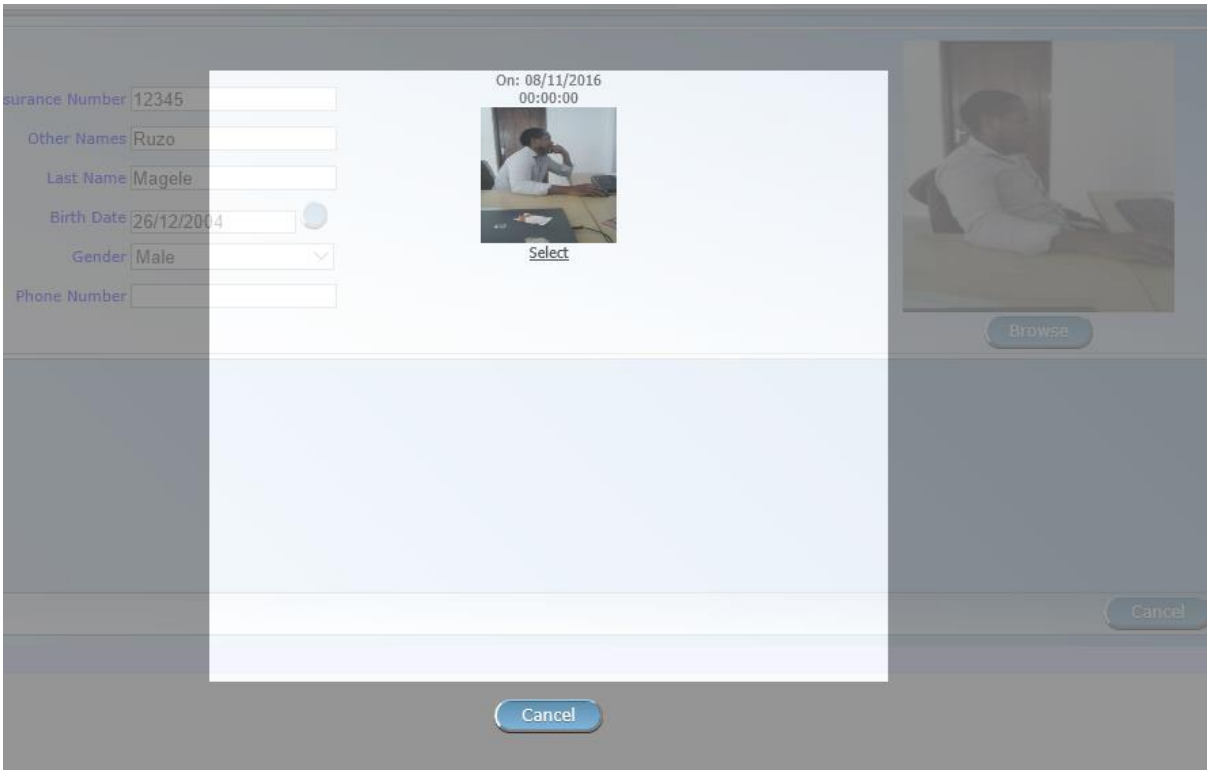


Interface object name:	*Insuree.aspx*
Menu path:	
Hyperlinks/Redirections:	*Insurance Number column in Insurees section in OverviewFamily.aspx*
Action Button:	*OverviewFamily.aspx (Add and Edit button in Insuree section)*
Use case Reference	*5.2.2 (Add insure)* *5.2.10 (Modify Insuree)*
Class Diagram	

Brief description:

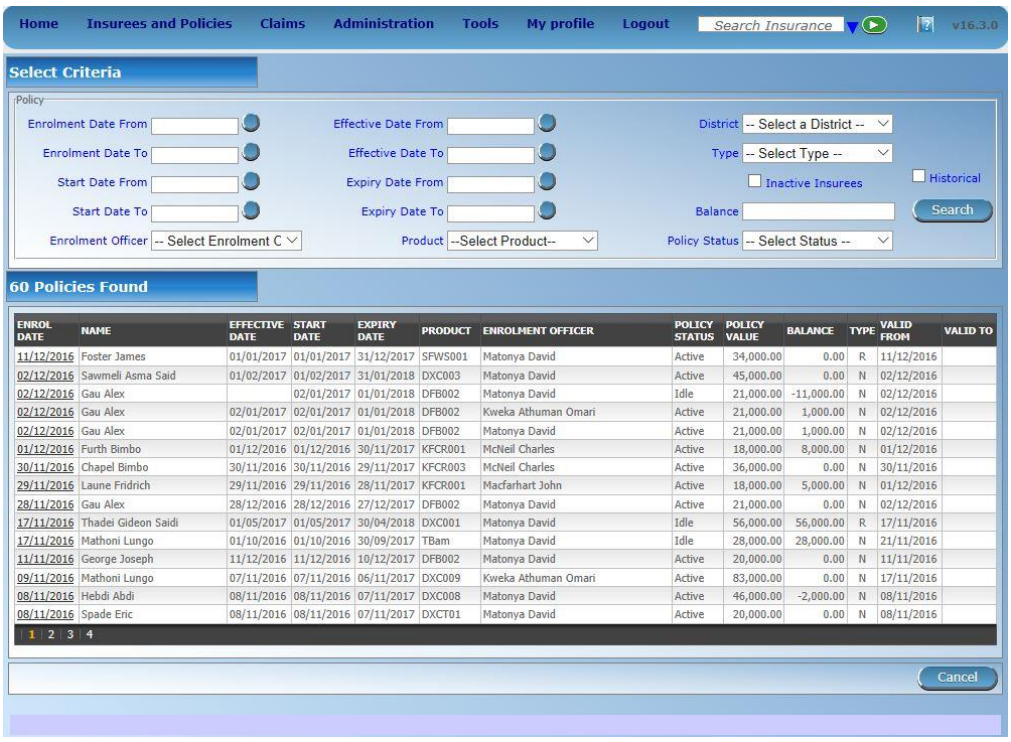
This interface opens up when the user has clicked the Add or Edit button on the family/group overview (Insuree section) . The insuree information can be changed as per defined 'use-cases' for adding and editing an Insuree. When adding a new Insuree, after entering Insurance Number automatically the system queries the folder for submitted photos and will try to associate the photo by using the photo naming convention (discussed later).
In case one photo is available the photo will be automatically presented as shown in the interface above. In case there would be more than one photograph, a browser window will open (shown below) and the operator could choose the correct photo.

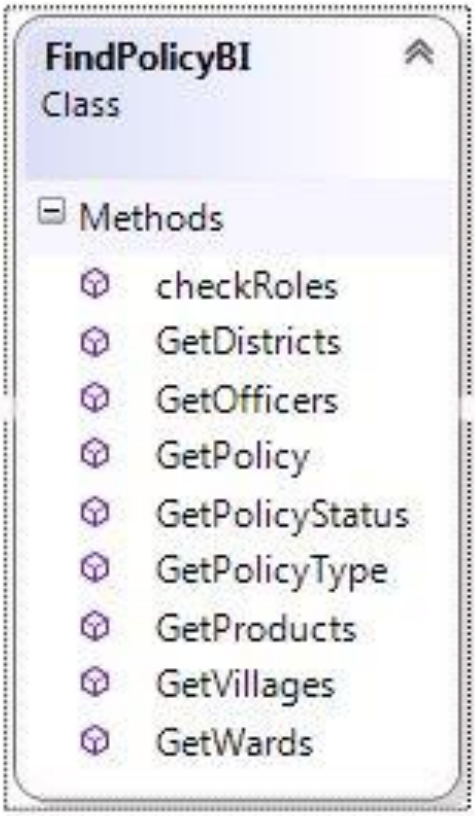
The browse button allows viewing all submitted photographs. The following screen will appear:



One could select a new Photo by clicking the 'select' button under the photograph. After selecting the desired photo, it will be reloaded in the page.
On saving, the photo will be associated with the selected Insuree. The cancel button will bring you back without any changes to the photograph holder.
On saving an insuree, he/she will be added /refreshed on the current insurees section within the family/group overview page.

5.2.11. FindPolicy.aspx

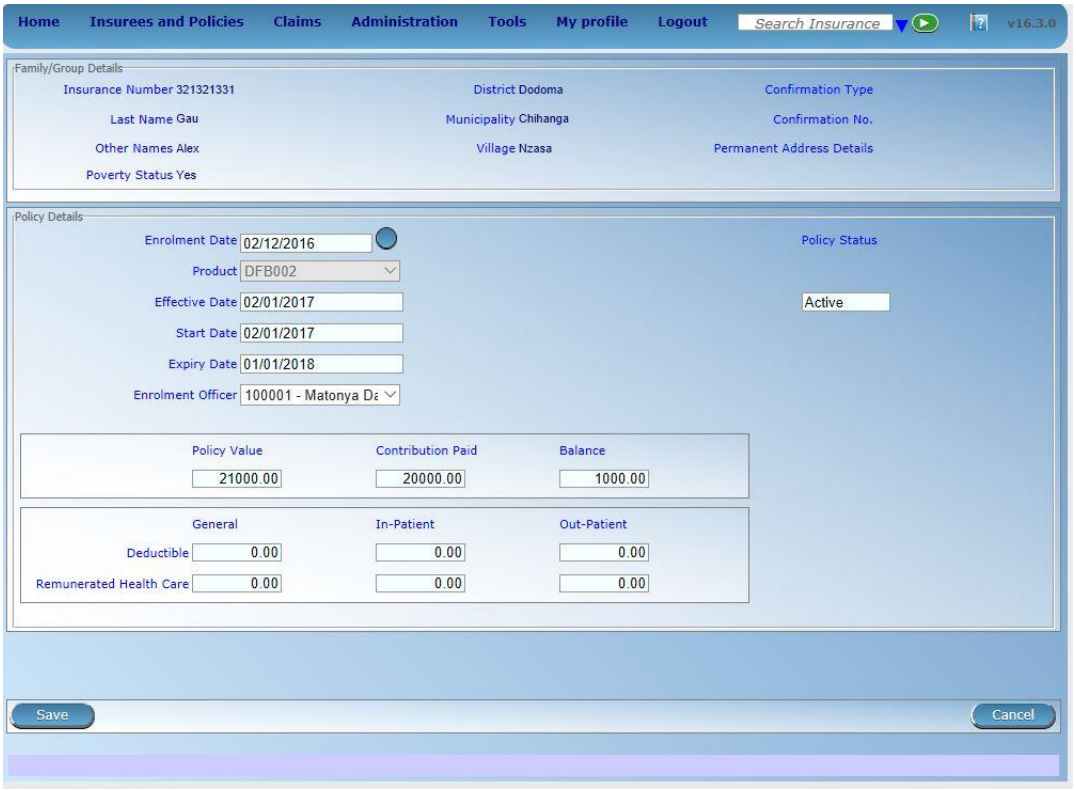


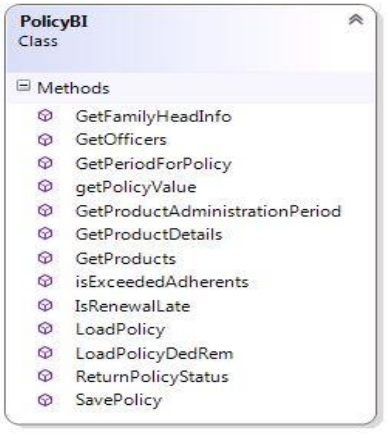
Interface object name:	*FindPolicy.aspx*
Menu path:	*Policies*
Hyperlinks/Redirections:	
Action Button:	*OverviewFamily.aspx (Cancel button)*
Use case Reference	*5.2.7*
Class Diagram	

Brief description:

This interface opens up when the user has clicked the Policies button option on the top menu (Insurees and Policies). This interface is used to search for Policies. The operator can enter the search criteria and click the search button. All records satisfying the criteria will appear. A click on the hyperlink on the first column (Enrollment Date) in the grid will open up the family/group overview page. Further operations on the policy will take action from this overview page.

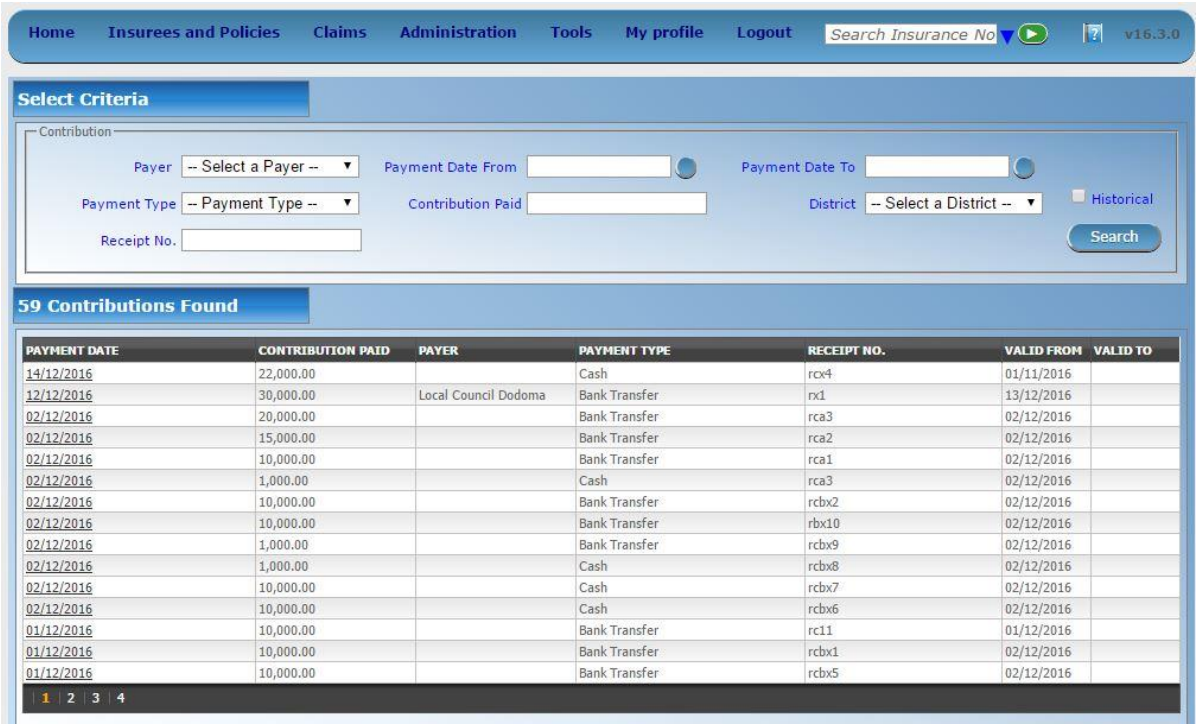
5.2.12. Policy.aspx

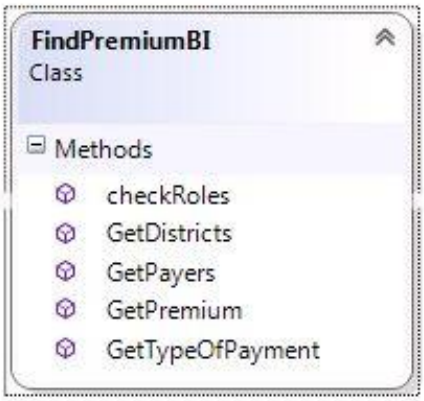


Interface object name:	*Policy.aspx*
Menu path:	
Hyperlinks/Redirections:	*Enroll date column in Policy section in OverviewFamily.aspx*
Action Button:	*OverviewFamily.aspx (Add and Edit button in Policy section)*
Use case Reference	*5.2.3 (Add Policy)* *5.2.11 (Modify Policy)*
Class Diagram	

Brief description:
This interface opens up when the user has clicked the Add or Edit button on the family/group overview (Policy section). The policy information can be changed as per defined 'use-cases' for adding and editing a Policy.
On saving a Policy it will be added /refreshed on the current policies section within the family/group overview page.

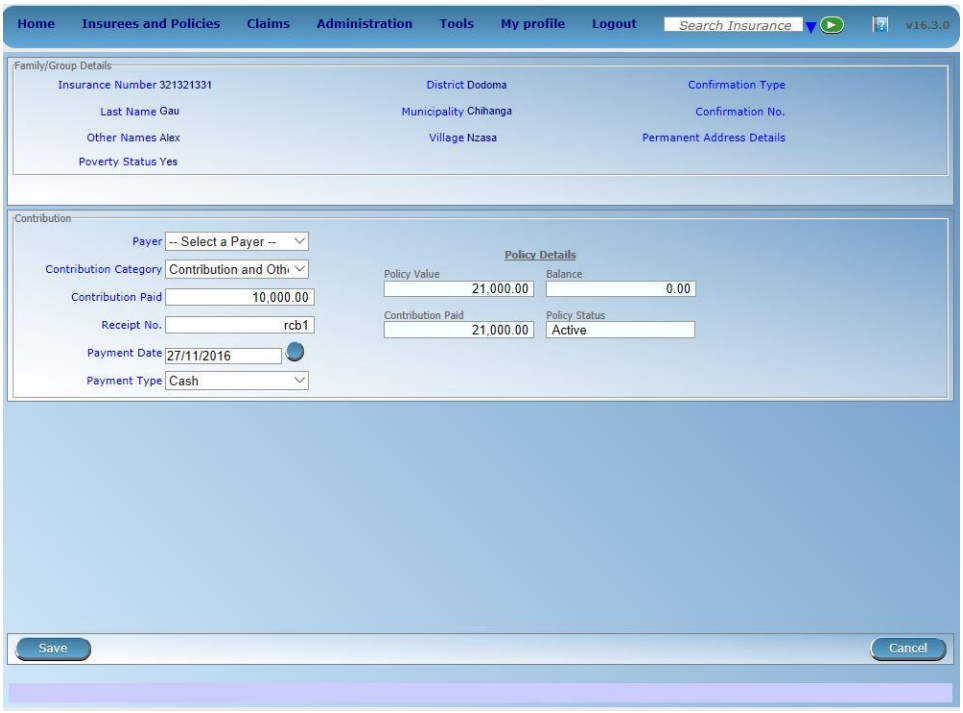
5.2.13. FindPremium.aspx

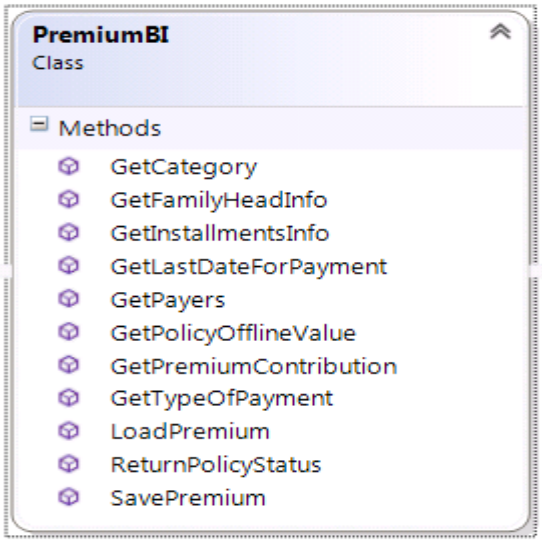


Interface object name:	*FindPremium.aspx*
Menu path:	*Contributions*
Hyperlinks/Redirections:	
Action Button:	*OverviewFamily.aspx (Cancel button)*
Use case Reference	*5.2.8*
Class Diagram	

Brief description:
This interface opens up when the user has clicked the Contributions button option on the top menu (Insurees and Policies). This interface is used to search for Contributions. The operator can enter the search criteria and click the search button. All records satisfying the criteria will appear. A click on the hyperlink on the first column (Payer) in the grid will open up the family/group overview page. Further operations on the Contribution will take action from this overview page.

5.2.14. Premium.aspx

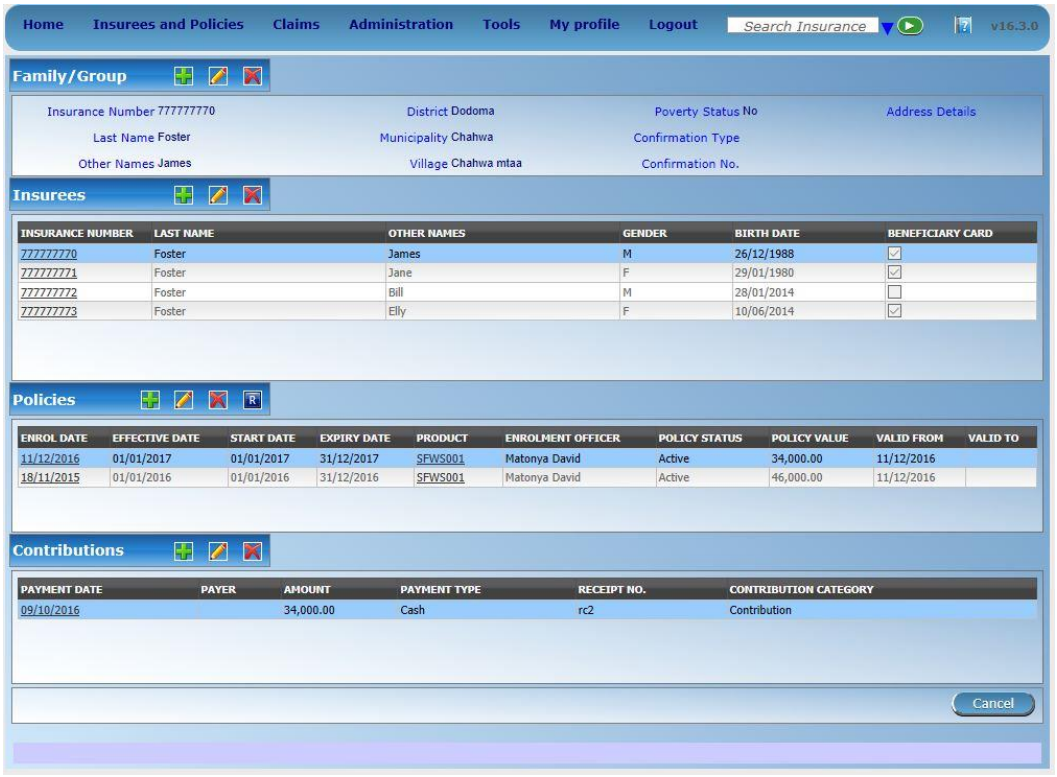


Interface object name:	*Premium.aspx*
Menu path:	
Hyperlinks/Redirections:	*Pay Date column in Contributions section in OverviewFamily.aspx*
Action Button:	*OverviewFamily.aspx (Add and Edit button in Policy section)*
Use case Reference	*5.2.4 (Add Contributions)* *5.2.12 (Modify Contributions)*
Class Diagram	

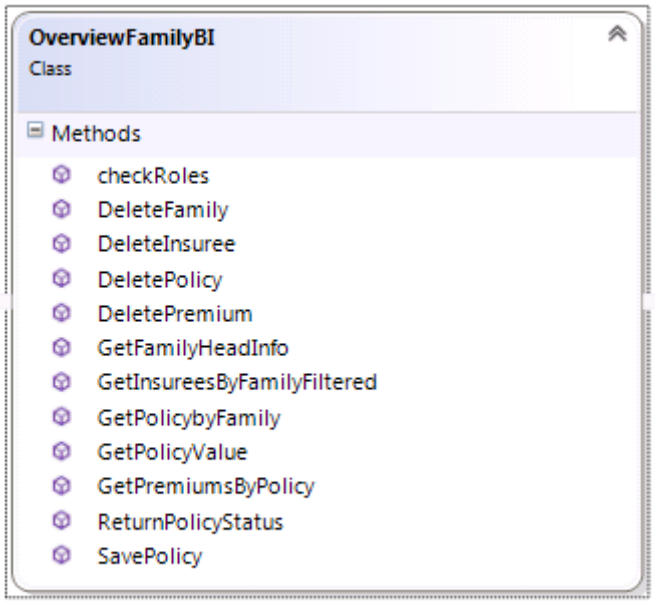
Brief description:
This interface opens up when the user has clicked the Add or Edit button on the family/group overview (Contributions section). The Contribution information can be changed as per defined 'use-cases' for adding and editing a Contributions.

On saving a Contribution it will be added /refreshed on the current Contributions section within the family/group overview page.
The Load Contribution call will also fetch the amount to be paid according the product(s) selected and member counts (member count, quantity of adults and children) of the family/group (refer to use cases). In case a Contribution paid is lower than the amount to be paid, a message will appear alerting the operator of such case. In the latter case the operator has the option to activate the policy regardless the covered amount. Each product has a grace period. In case of 'under' payments, the system will automatically 'suspend' the policy after expiration of the grace period. This action is performed by a daily service that runs in the background.
A policy will become active only when the sum of Contributions paid is equal or higher than the amount to be paid.

5.2.15. OverviewFamily.aspx



Interface object name:	*OverviewFamily.aspx*
Menu path:	
Hyperlinks/Redirections:	*FindFamily.aspx (Insurance Number)* *FindInsuree.aspx (Insurance Number)* *FindPolicy.aspx (Enrollment date)* *FindPremium.aspx (Payer)* *Product.aspx (product.aspx)*
Action Button:	*Premium.aspx (cancel button)* *Policy.aspx (cancel button)* *Insuree.aspx (cancel button)* *ChangeFamily.aspx (cancel button)*
Use case Reference	*5.2.15 (Delete Family/Group)* *5.2.16 (Delete Insuree)* *N/A (Delete Policy)* *N/A (delete Contribution)* *Basically this interface is the result of several searches and will be used to activate several other use cases (adding, editing, deleting)*

Class Diagram	

Brief description:

This page will be the main page for adding, editing and deleting of the following entities related to a family/group:

- Insurees
- Policies
- Contributions

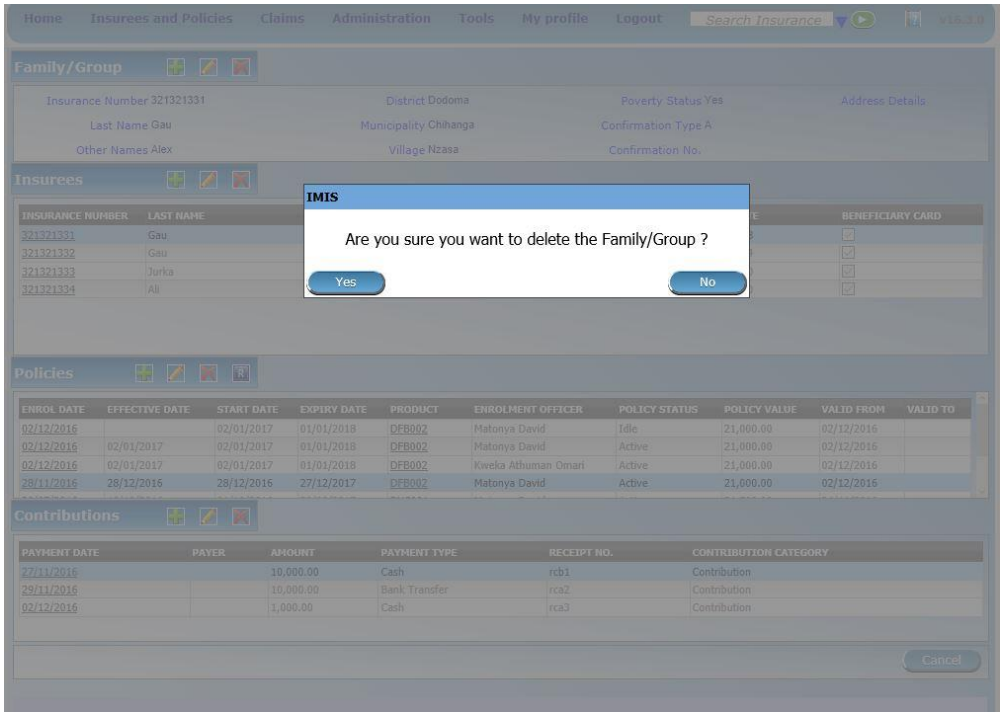
All insures related to the family/group (active only) will be shown in the section for Insurees. Adding of new Insurees under the same family/group will be done by clicking the add button in the Insuree section. Similar wise we can add policies and Contributions that way.
Editing of any of the three entities will be performed by clicking the edit button or by clicking the hyperlink (first column) in the sections. The functionality of 'Modifying a family/group', 'change of head of family/group' or 'move insure' is found under the edit function in the family/group section. The actual functionality has been described earlier in the chapter as for several other functionalities activated in this user interface.

By selecting the Policy, the Contribution section will be refreshed, showing the Contributions paid on the selected policy. By default the first policy found is selected automatically.

A separate hyperlink will be positioned on the 'Product' column in the policies section to show the actual product in view mode via a popup window.
Similar wise a hyperlink will be available on the 'Payer' column in the Contributions section to provide, via a read-only popup, additional information on the payer.

By clicking the delete button in the sections, a validity check will take place on the deletions. In case a record should not be deleted the user will be informed. If the validation has passed, IMIS produces a confirmation box where the operator has to confirm the operation.

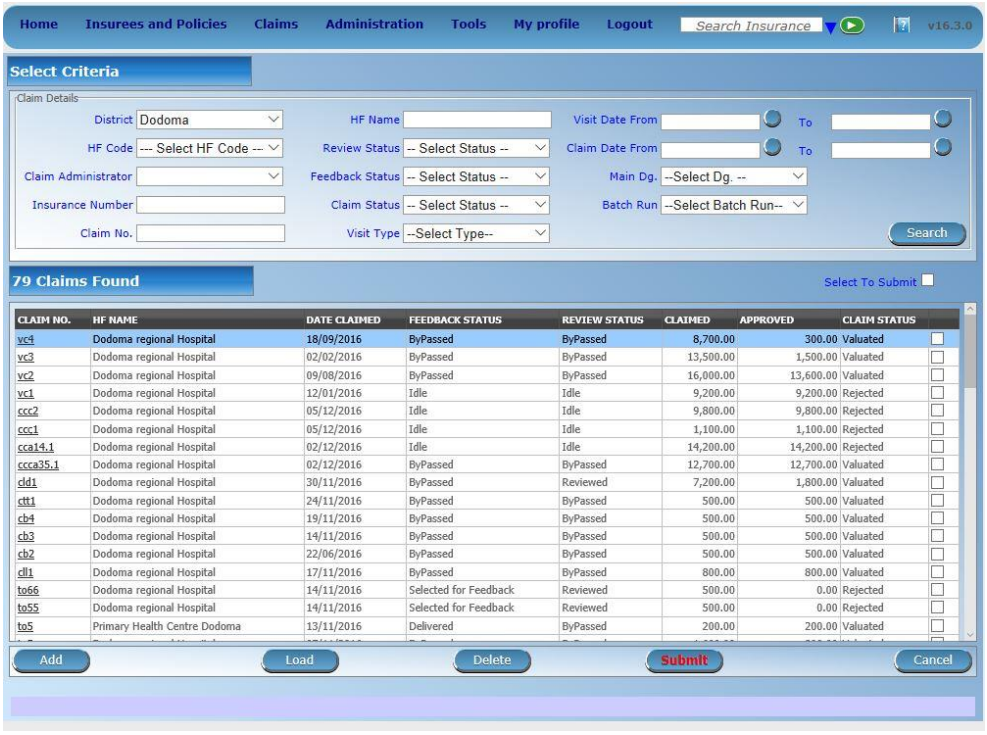
Please find below the confirmation message for this deletion.

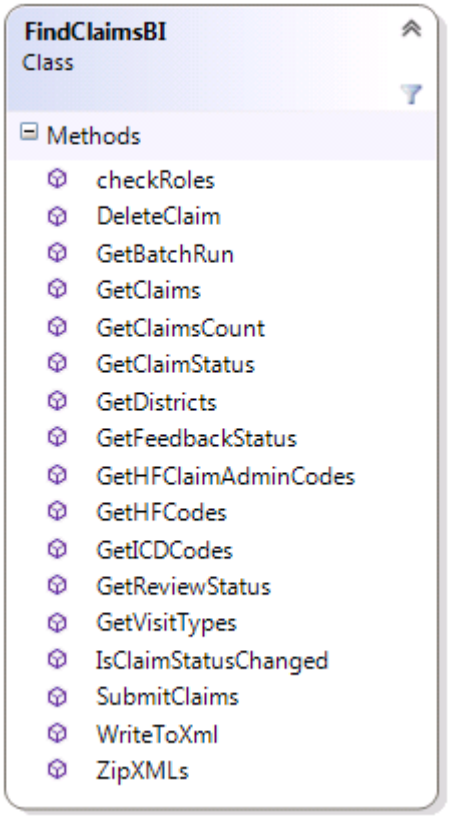


After confirmation, the entity will be deleted by flagging the ValidityTo field in the record. No additional (audit) record needs to be inserted as the record will be logically deleted from the system. Special developed 'nested' SQL Triggers will update all related records by setting the ValidityTo field.

The design has also added 'deletion' possibilities for Products and Contributions as there might be situations in which the operator mistakenly entered policies or Contributions. These mistakes can in this case be rectified by deletions. No use cases support this scenario.

5.2.16. FindClaims.aspx



Interface object name:	*FindClaims.aspx*
Menu path:	*Health Facility Claims(under Claims menu)*
Hyperlinks/Redirections:	
Action Button:	*Claim.aspx (Save and Cancel buttons)*
Use case Reference	*3.2.8(Finding Claim Direct)*
Class Diagram	

Brief description:

This interface opens up when the user has clicked the 'Claims' option under the top menu Claims.
In case an operator is 'linked' to a specific Health facility, the District, Health Facility Code and Health Facility Name boxes are already entered and locked. Furthermore, in that case, all claims with the status 'Entered' are automatically displayed. Alternatively, the operator is able to search for the desired claims by using the various filters as seen in the screenshot. By clicking the search button, all records satisfying the criteria will appear. A click on the hyperlink on the first column (Claim Code) in the grid will open up the Claim page to show all related information to the claim.

The following buttons are available on the button bar (depending on statuses and operators role):

Button	Role	Claim Status
Add	Claim Administrator, Clerk	'Entered', 'Checked', 'Processed', 'Valuated'
Delete	Claim Administrator, Clerk	'Entered'
View	Medical Officer	'Entered', 'Checked', 'Processed', 'Valuated'
Load	Claim Administrator, Clerk	'Checked', 'Processed', 'Valuated'
Submit	Claim Administrator, Clerk	'Entered', 'Checked', 'Processed', 'Valuated'

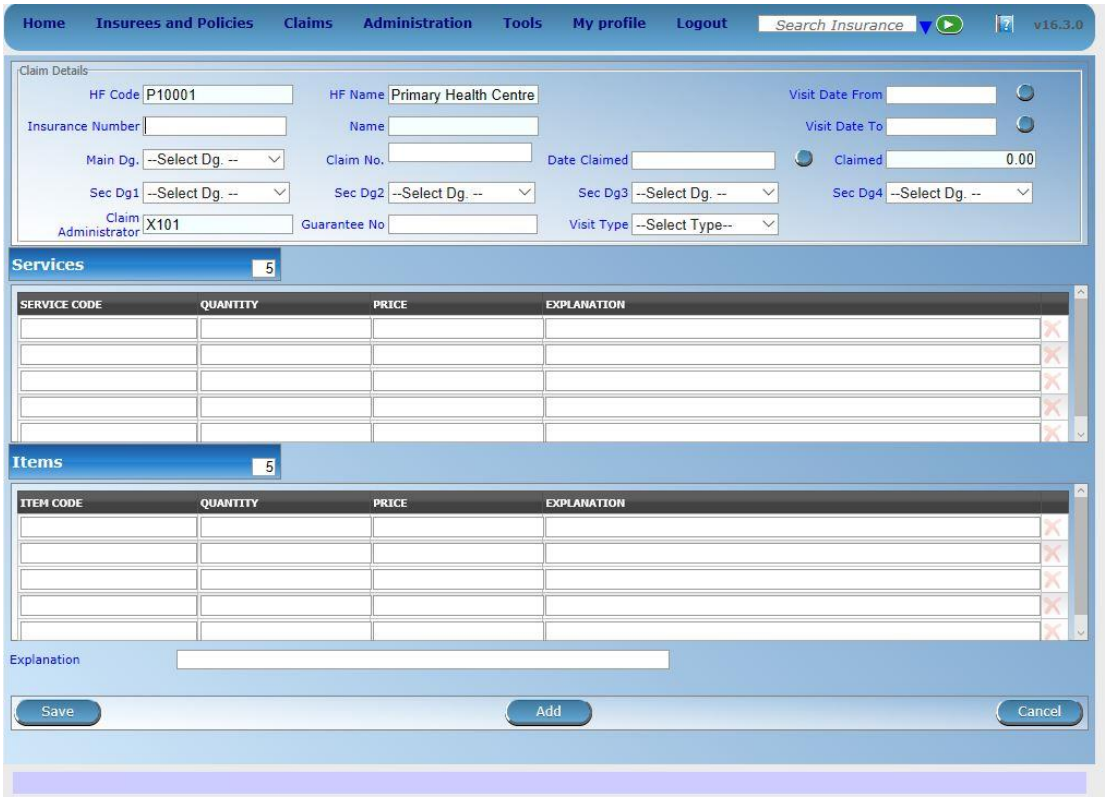
Depending on the role of the operator and the status of the claim (see above), by clicking the 'Add', or 'View' buttons, the Claim.aspx interface will be opened.

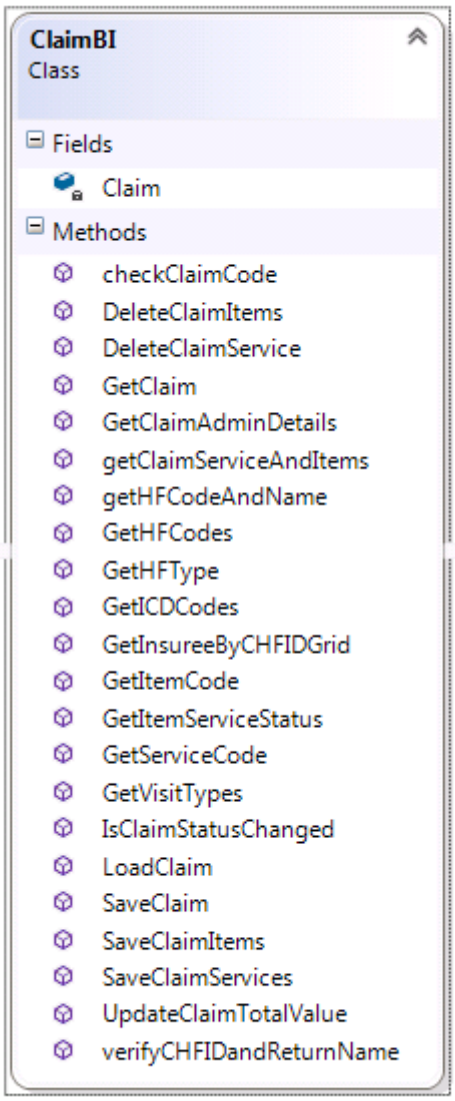
The Claim interface will be used to add, edit or view a claim.
By clicking the hyperlink of the claim (first column), the claim interface will be opened in read-only mode or edit mode depending on the role of the user and the status of the claim.
As long as claims are in the status 'entered' they could be modified or deleted. To finally submit a claim to IMIS Administrator, Admin would need to select the claim on the right side of the data grid and click the 'Submit' button. The claim status will now automatically be changed to 'Checked' after the system has performed the standard verification.
One could select one or multiple claims for submission. By clicking the 'All' button above the column 'Submit' in the grid, all claims with status 'Entered' will be candidate for submission by having its Submit field checked. By clicking the 'Submit' button with having multiple claims selected, all claims will be verified and set to the status 'Checked'.

In case we are operating in an 'Offline' mode, on clicking the 'Submit' button, the system will generate an XML file containing the claims. At the same time, the claims are updated to 'Checked'. The 'Checked' status in the offline mode obviously should be interpreted differently as if we would have the checked status on Central level. In case we are Offline, 'Checked' means that the HF has submitted the claim.

The XML file needs to be provided (uploaded) to the server in order to be inserted in the Claims table. This process of inserting the XML based claims is done automatically by a windows service running in the background at the server. The upload procedure of the XML data from mobile phones and offline IMIS clients is exactly the same.

5.2.17. Claim.aspx



Interface object name:	*Claim.aspx*
Menu path:	
Hyperlinks/Redirections:	*FindClaims.aspx (Claim code column)*
Action Button:	*FindClaims.aspx (Add and Edit button)*
Use case Reference	*3.2.2*
Class Diagram	

Brief description:

This interface opens up when the user has clicked the hyperlink on the Find Claim interface or the 'Add', or 'View' button on the Find Claim interface.

This interface is used to Add, Edit and View the selected claim.

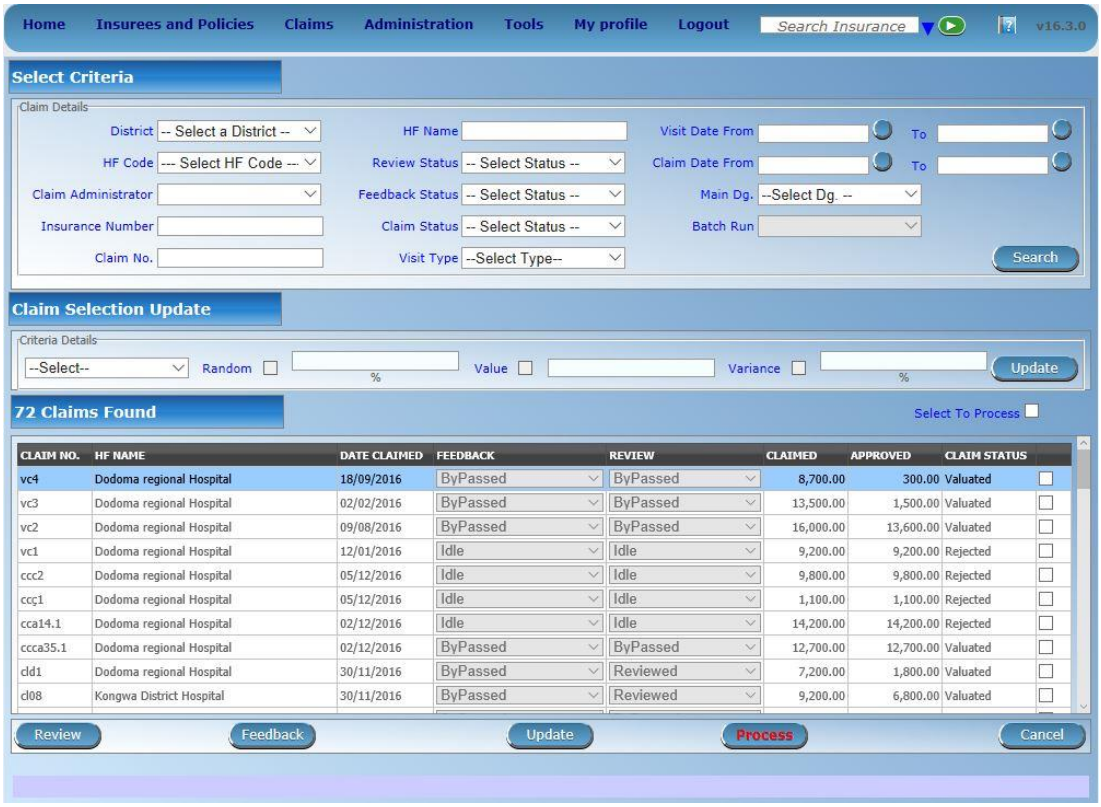
The HF Code is mandatory and needs to be selected in case the operator has no HF code linked to its user profile.

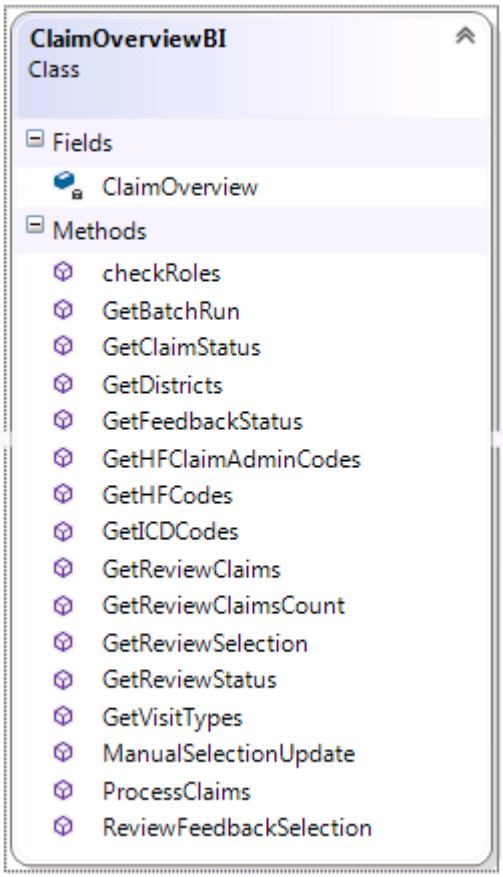
The claim interface holds in the top section of the screen the header information on the claim. The operator has to enter all relevant information on the claim as per use cases. By clicking the save button, data will be validated

and stored in the database. Data is editable only in case the claim is in the status ‘entered’ and the operator role is ‘Clerk’ or ‘Claim Administrator’.
In all other cases, data is only available in read-only mode.

The claim total is dynamically calculated (client side) at loading of the claim and each time an item or service is added, changed or deleted.
The operator is able to enter optionally any explanation to elaborate on the claim contents or value.
Further to this, the operator is also able to enter an explanation on claim item or service level.

5.2.18. ClaimOverview.aspx



Interface object name:	*ClaimOverview.aspx*
Menu path:	*Review (under Claims main menu)*
Hyperlinks/Redirections:	
Action Button:	*ClaimFeedBack.aspx (Save & Cancel buttons)* *ClaimReview.aspx (Save & Cancel buttons)*
Use case Reference	*3.2.9(Review selection and Review batch of Claims)* *3.2.10(Feedback selection and prompting for feedback)*
Class Diagram	

Brief description:

This interface opens up when the user has clicked the 'Review' menu option under the Claims menu. This interface is used to get information about the claim(s) and to select the claim(s) for further review or feedback.

If the claim has the status 'Processed' (=8) or Valuated (=16) the interface will only allow viewing of information.

The operator is able to filter the claims by using the various filters as seen in the screenshot. The following filters are available for filtering the claims:

- District
- HF Code
- HF Name
- Claim Diagnosis
- Insurance Number
- Claim From Date

- Claim To Date
- Visit From Date
- Visit To Date
- Review Status
- Feedback status
- Claim Status
- Claim Code
- Claim Administrator
- Visit Type
- Batch Run

The HF Code combo box will be only containing the HFs of the district (s) available for the current operator.
The default Claim status will be set to 'Checked'.
The system will automatically show all claims with status 'Checked' And (Review Status OR Feedback Status) = 'Idle'. These claims will be the claims that need to be selected for Review or Feedback.
The 'claim date from' and 'claim date to' will be blank initially.
The operator can, after the automatic data retrieval, further filter the data by using the optional filters and clicking the 'Search' button.
Each Claim within the grid has a combo box that shows the current Review status of the claim. The following statuses could appear:

- I → Idle (=0)
- N → Not selected for review (=1)
- S → Selected for review (=2)
- R → Reviewed (=4)
- P → (By-) Passed (=8)

The operator can use 4 methods for the 'review' selection process

- Automatic: randomly select claims based on a certain percentage (default = 5%)
- Automatic: Select claims that are having a claimtotal over a certain value (default = 40,000)
- Automatic: Select claims that have a variance of over a certain percentage compared with the average of claims related to a similar ICD code over 1 years information (currently suggested) within the same district. (or country wide?) (Default =50%)
- Manually: Operator judges and changes the review status of claims in the grid manually and clicks the update button in the bottom bar of the interface.

The automatic selection can only be applied to the claims with a review status 'Idle'.
In case the operator chooses one of the automatic selection methods, all claims satisfying the criteria will be changed from 'I' →'N' or 'S'. It is possible to manually 'select 'or 'un-select' a claim for review. In case the review status of a claim has been changed to 'R' (via interface ClaimReview.aspx), no further changes are allowed to the claim review status.

A click on the 'Review' button will open up the Claim Review page for the selected claim with the status 'S' or 'R' only.

Besides the review status, each claim within the grid has a combo box that shows the current Feedback status of the claim. The following statuses could appear:

- I → Idle (=0)
- N → Not selected for feedback (=1)
- S → Selected for feedback (=2)
- D → Delivered (=4)
- P → (By-) Passed (=8)

The operator can use 4 methods for the 'feedback' selection process:

- Automatic: randomly select claims based on a certain percentage (default = 5%)
- Automatic: Select claims that are having a claim total over a certain value (default = 40,000)

- Automatic: Select claims that have a variance of over a certain percentage compared with the average of claims related to a similar ICD code over 1 years information (currently suggested) within the same district. (Or country wide?) (Default =50%)
- Manually: Operator judges and changes the feedback status of claims in the grid manually and clicks the 'update' button in the bottom bar of the interface..

In case the operator chooses one of the automatic selection methods, all claims satisfying the criteria will be changed from 'I' →'N' to 'S'. It is possible to manually 'select ' or 'un-select' a claim for feedback. In case the feedback status of a claim has been changed to 'D' via interface ClaimFeedback.aspx or the automatic upload of Feedback records via the XML upload procedure, no further changes are allowed to the claim feedback status.

A click on the 'Feedback' button will open up the Claim Feedback page for claims with the status 'S' or 'D' only.

The claims selected for feedback will be handles by a separate process that runs each day automatically (via a separate windows service) from the server. This process will loop through all claims that were selected for feedback and will register in a separate table 'TblFeedbackPrompts' all needed details to be sent via the SMS Message.

A separate interface will be available (like the policy Renewals in Phase 1) to allow reporting on the feedback SMS journal and running a report separate from the SMS service, in case the VEOs do not have mobile connectivity or the phone number is missing. The report will contain all information as mentioned in use case 3.2.10.

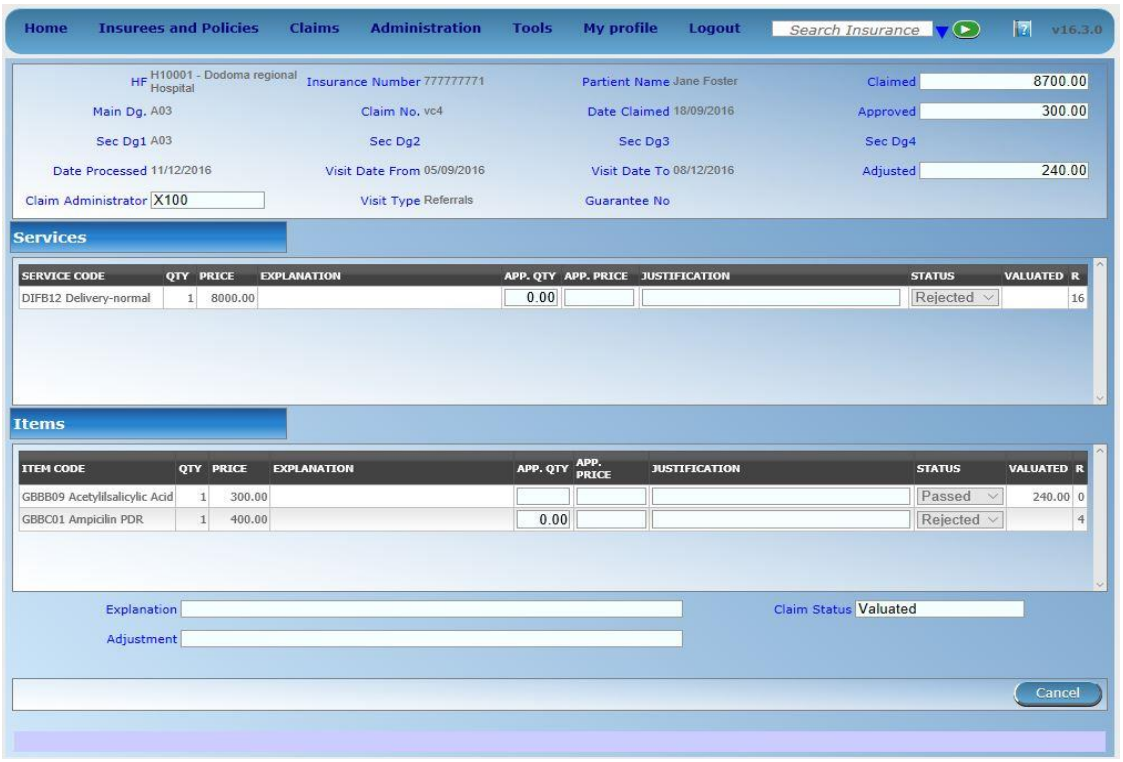
In case the medical officer considersall currently shown claims in the grid as 'reviewed' and would like to process these claims, he/she could change the status of these to 'Processed' (=8) by clicking the 'Process' button on the right side of the screen.

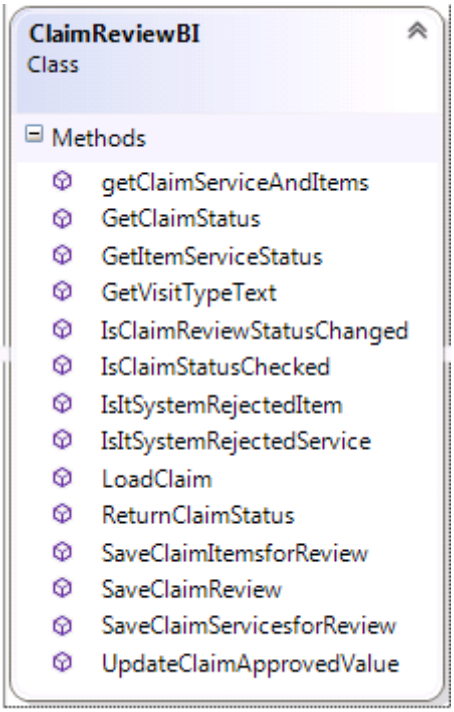
On clicking the 'process' button, the system will only consider claims in the grid with claim status 'Checked' and Review Status <> 'Idle' And Feedback Status <> 'Idle'.

In case one or more claims considered in the process, were having the status 'selected for review' at the moment of processing, a warning message will be displayed alerting the operator that some claims have not been reviewed yet. The same procedure will be applied to the feedback status. Any claims that are in still in the 'Checked' status and are selected for review will be set to review status 'ByPassed' (=8). The same applies to claims that were selected for feedback but feedback was not delivered yet. In the latter case, the feedback status will be set to 'ByPassed' (=8).

After the operator confirms the update, all considered claims are set to 'processed' and the 'processed date' is set to 'now'.
The system will now attempt to 'valuate' the 'processed claims'. The'valuation' process, is described later in paragraph 3.3.

5.2.19. ClaimReview.aspx



Interface object name:	*ClaimReview.aspx*
Menu path:	
Hyperlinks/Redirections:	
Action Button:	*ClaimOverview.aspx (Review button)*
Use case Reference	*3.2.9 (review a claim)*
Class Diagram	

Brief description:

This interface opens up when the user has clicked the 'Review' button on the Review selection interface (claim review status 'S' or 'R'). This interface is used for displaying claim information and entering review data by the medical officer.

The medical officer can adjust the following statuses manually:

- Claim status ([Checked] ,Processed, Rejected)
- Claim Item Status (Passed , Rejected)
- Claim Service Status (Passed, Rejected)

If a claim has not yet been set as processed, the operator can still change rejected items and services as passed as long as the rejection did not come from the automated check at the moment of submitting the batch. A separate field will be kept internally for the rejection reasons (automatic or manual)

A claim could also be manually adjusted by entering the Qty Approved and Value approved fields in the sections for services and items. An optional justification comment could be provided as well. Further to this, the medical officer could simply reject the claim item or service.

Additional justification text can be amended to each claim item and service and also on general claim level.
The date released will be set by the system as soon as the claim has been set to the status valuated (automated via a system procedure).

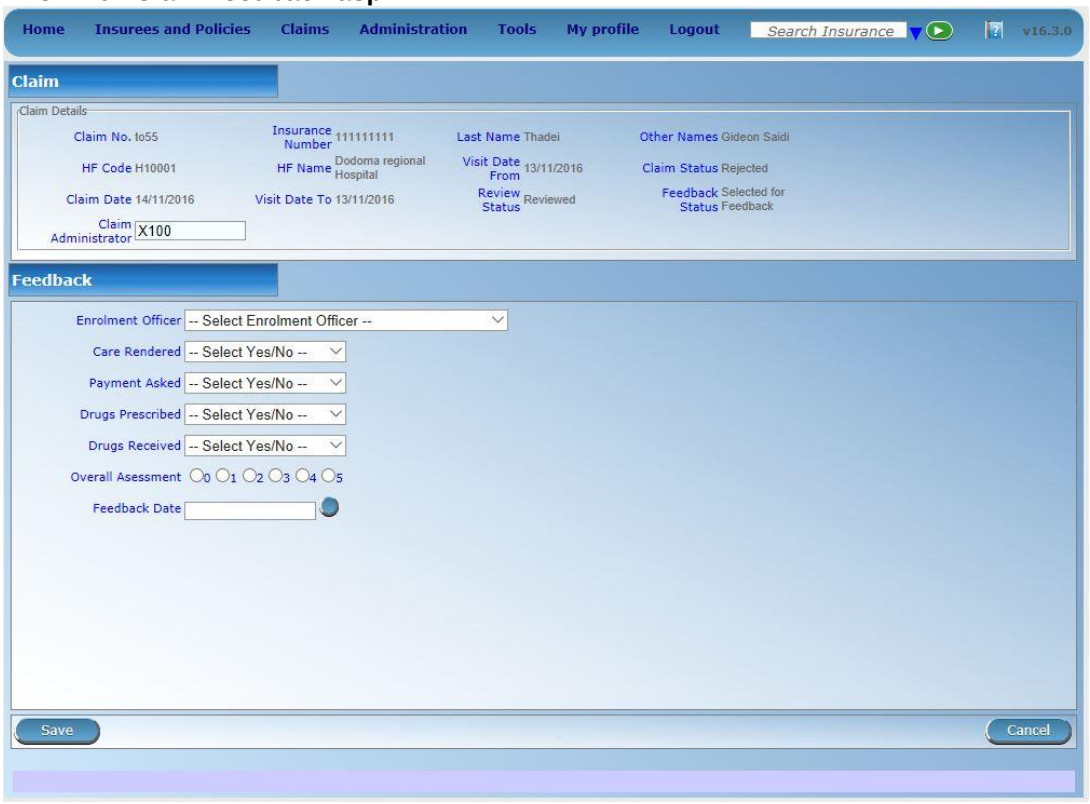
This interface is only available to medical officers.

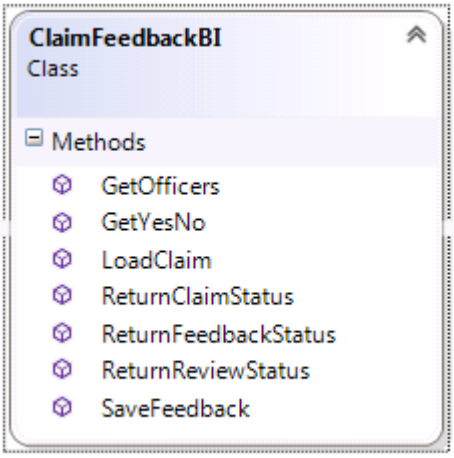
The interface will show three dynamically calculated totals for values (locked):

- Original Claim value
- Adjusted Claim value
- Valuated Claim value

A claim with status 'Checked' and feedback status <>'Idle' and review status <> 'Idle' could be set to processed by changing the Claim status and clicking the 'Save' button.
At the moment of saving a Claim with the new status 'Processed' , the status for Review will automatically change to 'Reviewed' (=4). In case the Feedback Status is 'S' but feedback was not delivered yet, the Feedback status will be set to 'ByPassed' (=8). Furthermore, the date processed will be set to 'now'.

5.2.20. ClaimFeedback.aspx



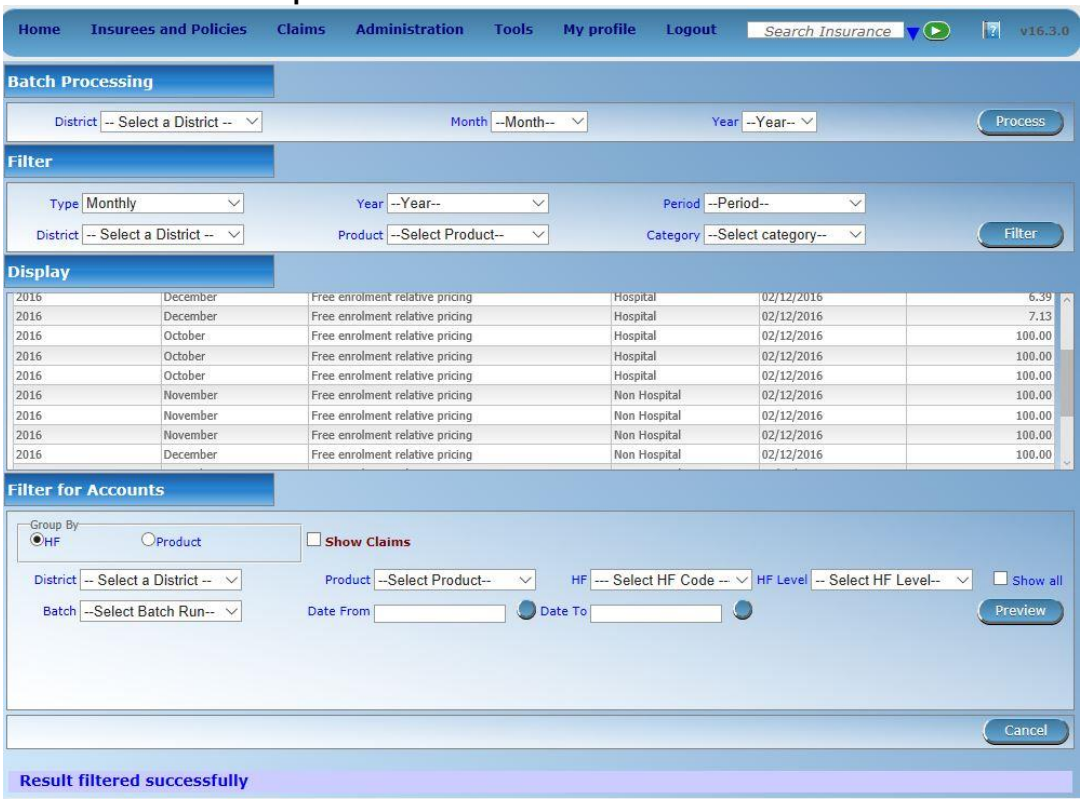
Interface object name:	*ClaimFeedback.aspx*
Menu path:	
Hyperlinks/Redirections:	
Action Button:	*ClaimOverview.aspx (Feedback button)*
Use case Reference	*3.2.11 (Feedback of a claim)*
Class Diagram	

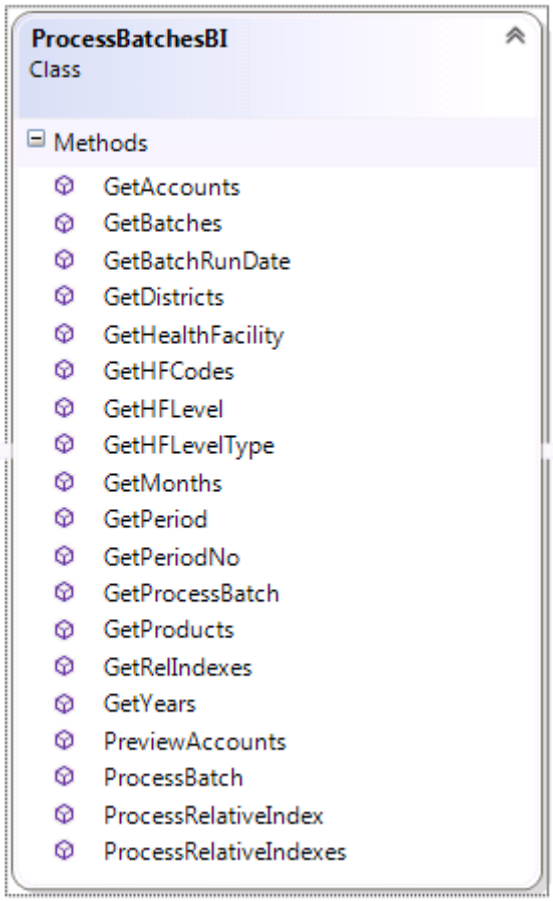
Brief description:

This interface opens up when the user has clicked the 'Feedback' button on Claim Review interface (claim feedback status 'S' or 'D') this interface is used for displaying feedback information on a claim for the medical officer and entering feedback data by the clerk.

The clerk will enter the required feedback and will click the save button to insert the information in the database. At the time of saving, the Feedback status of the claim changes to 'D' (delivered = 4). Feedback records could be manually entered or will be uploaded automatically by a service running in the backgrounds that uploads XML files originating from the mobile phones (via mobile feedback application).

5.2.21. BatchRun.aspx



Interface object name:	*ProcessBatches.aspx*
Menu path:	*Batch Run (under Claims menu)*
Hyperlinks/Redirections:	
Action Button:	
Use case Reference	*3.2.15 (Calculating the relative indexes) and 'Export to Epicor'*
Class Diagram	

Brief description:

This interface opens up when the user has clicked the 'Batch Run' menu option in the Claims menu. This function is used to generate the monthly, quarterly and yearly indexes used for the calculation of relative prices for items and services that have been defined under the product with Price origin 'R' (Relative). This function should be executed each month. In normal cases this process is executed one month after the calendar month has been passed. E.g. we would execute this process for the month of June by the end of July to allow health facilities and IMIS Scheme administrator to(enter, submit, review etc..) all batches for the period of June.

By clicking the process button, the system will calculate the index for the month selected. Furthermore, in case we are processing month 3, 6, 9 or 12, we will also run the calculation process for the quarterly index. In case we are in month 12 we will besides the monthly and quarterly index also calculate and insert a yearly index record in table tblRelIndex. The actual process that will run is elaborated in a separate flowchart. The system will not allow duplicate indexes for periods. In case an operator attempts to insert an already existing index, a user warning will be displayed and the process will be cancelled.

After running the calculation of relative indexes as described above, the system will go through all claims that are in the status 'Processed' and will attempt to set the claims to the status 'Valuated' by going through all claims which are in the status 'processed' and applying the relative index were needed. If all items and services in a claim are valuated, the claim is valuated.

After all claims are being handled and possibly be set to ‘valuated’, we will run through all claims in the state ‘valuated’ that have not yet been assigned a so called ‘Run ID’.
This ID will ‘bind’ all claims belonging to the same run together for reporting to the accounting system.
All unassigned ‘valuated’ claims are now updated with the ‘RunID’.

All batches that could not be ‘valuated’ are kept in the status ‘Processed’ and might be included in the next run.
Yearly, Quarterly and Monthly runs are kept separate.

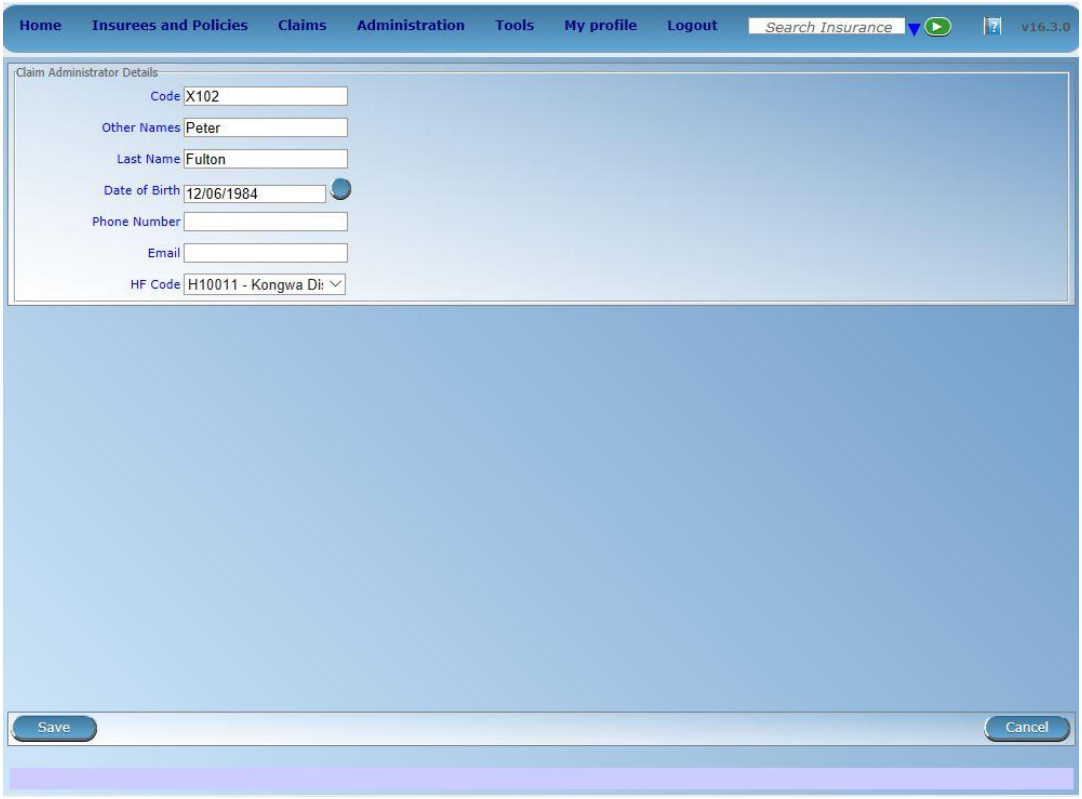
The search button will allow the operator to filter existing indexes for products etc. The calculated indexes that satisfy the criteria will be displayed in the grid.

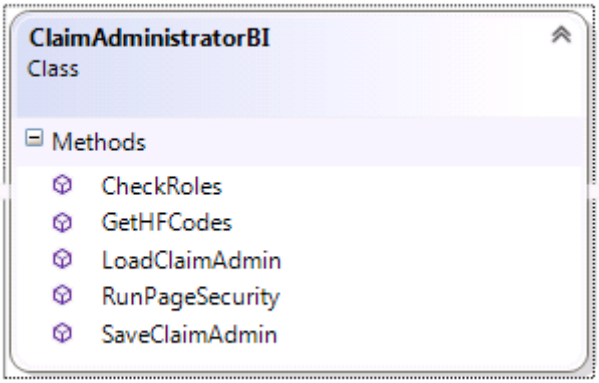
The lower section will allow filtering records from the BatchRun table. The following filters will be available:

- Period from (will be filtering Batch run date)
- Period to (will be filtering Batch run date)
- Month of run
- Year of run
- District
- Product
- Health facility
- Care type (Inpatient , Outpatient, both)

After setting the filters the system will prevail which batch RUNS (full or partial contents) satisfy the criteria.
The preview button will generate the report based on the criteria. This report could be a subset of what was sent as an instruction to the accounting system or the full accounting instruction. The report will be grouped by District, Health facility and Product.

5.2.22. ClaimAdministrator.aspx



Interface object name:	*ClaimAdministrator.aspx*
Menu path:	*Claim Administrators (under Administration menu)*
Hyperlinks/Redirections:	*FindClaimAdministrator.aspx (Claim Administrator Code)*
Action Button:	*Add, Edit Button*
Use case Reference	*(Adding, Editing, Deletion of Claim Administrators)*
Class Diagram	

Brief description:

This interface opens up when the user has clicked the Add or Edit button on the claim administrator search page (or clicked the claim administrator code hyperlink). The claim administrator information can be changed as per defined 'use-cases' for adding and editing claim administrator.

On saving, claim administrator will be added /refreshed on the data grid within the claim administrator search page as described before.

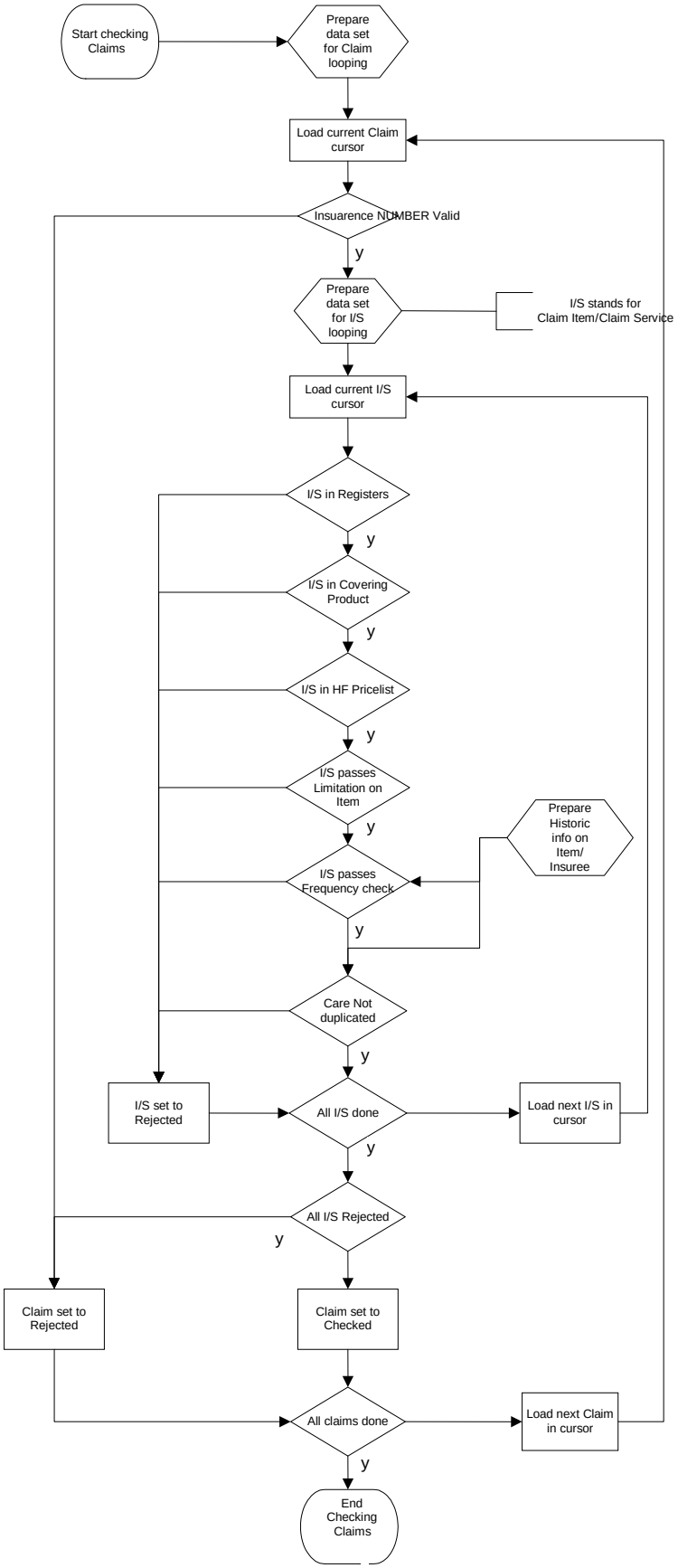
5.3. Checking of claims process

The checking of claims process will be automatically executed at the moment the Claim administrator or Clerk submits the claims (via the Submit button). The actual checking procedure will run purely on T-SQL level at the server side. The procedure will loop through all claims one by one. Each claim will have again a looping mechanism for each item and service found under the claim. The items and claims are checked on validity and could have either the status 'Passed' or 'Rejected'.

Claims will be updated by the process from the status 'entered' (default) towards status 'checked' or 'rejected'.

The batch will be updated from 'Open' towards 'Checked' or 'Rejected'.

The following flowchart will graphically present the checking process of submitted Claims


Start checking Claims
Prepare data set for Claim looping
Load current Claim cursor
Insuarence NUMBER Valid
y
Prepare data set for I/S looping
I/S stands for Claim Item/Claim Service
Load current I/S cursor
I/S in Registers
y
I/S in Covering Product
y
I/S in HF Pricelist
y
I/S passes Limitation on Item
y
Prepare Historic info on Item/ Insuree
I/S passes Frequency check
y
Care Not duplicated
y
I/S set to Rejected
All I/S done
Load next I/S in cursor
y
All I/S Rejected
y
Claim set to Rejected
Claim set to Checked
All claims done
Load next Claim in cursor
y
End Checking Claims

5.4. Batch Valuation process

The claim valuation process will start at the moment of clicking the Process button on the ClaimOverview page (possibly multiple claims) or on the individual save action on the Claim review interface e (ClaimReview.aspx)

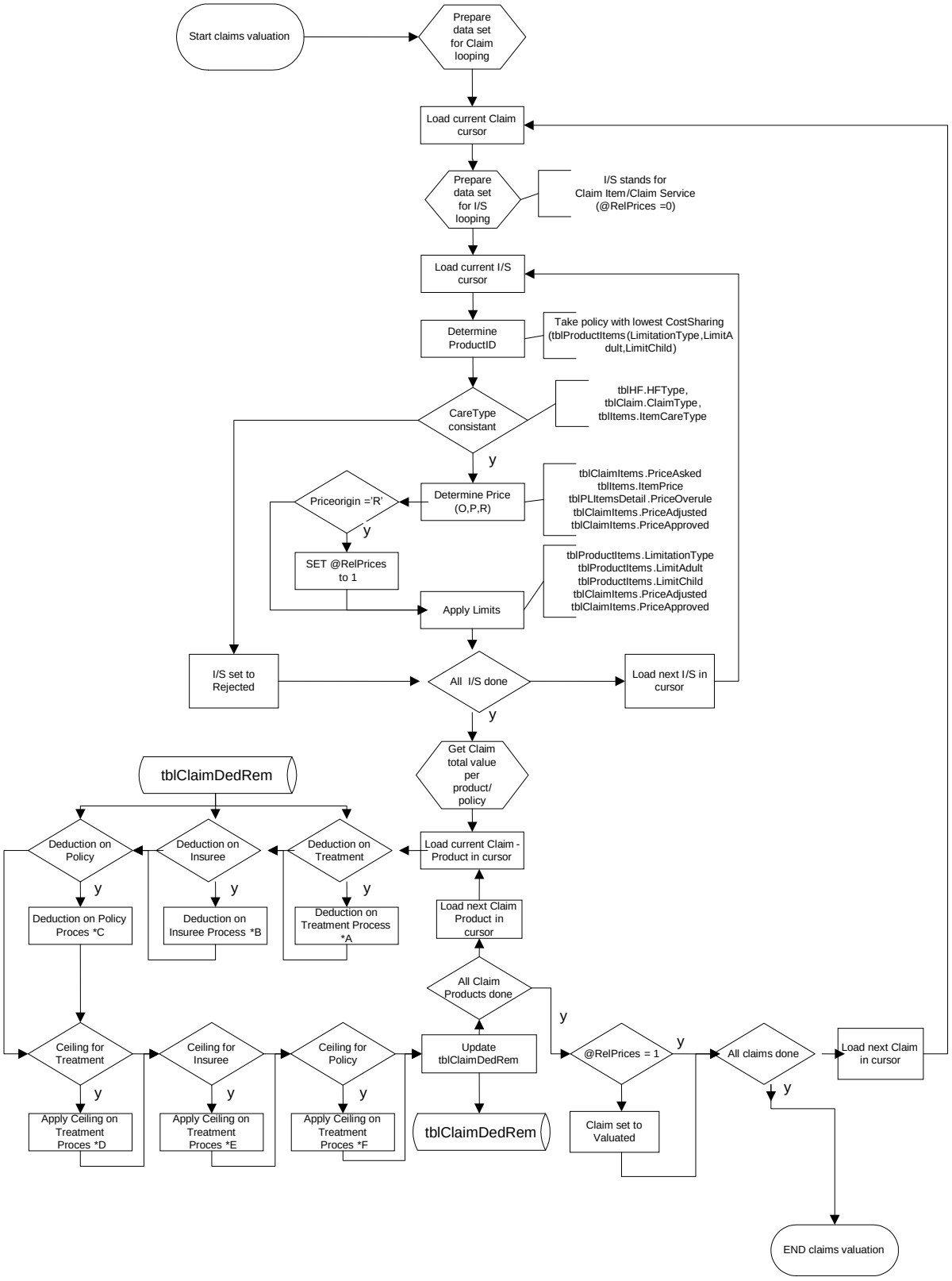
Automatically the system will run through a process that will loop through all claims and will update the claims with the amount to be remunerated towards the health facilities.

Most of the validations are done on item and service level lined up with the actual covered product /policy for the insuree. First we will validate and provide information on prices of services and items. Thereafter we will loop through all products and adjust (if needed) the amounts to be renumerated depending on deductions (Treatment,Insuree,Policy) and Ceilings (Treatment,Insuree,Policy).

In case a product item or service involved in the claim is having the 'price origin' set to 'R' (relative prices), then the final valuation of the claim can only be done after the relative prices index has been calculated for the period in question.

The process of relative price calculation will be performed manually on monthly basis. In case we are generating the monthly index for March we will, at the same time calculate the relative price index for the first quarter of the year. In case we calculate the relative price index for the month of December, we will at the same time also calculate the index for the last quarter of the year and the yearly index. The process of index calculation is covered in a separate paragraph.

The claim valuation process will also insert records in table 'tblClaimDedRem'. This table will hold all transactions on deductibles and remunerations per policy and per insuree. This table will be used for output and input in the valuation process.


Start claims valuation
Prepare data set for Claim looping
Load current Claim cursor
Prepare data set for I/S looping
I/S stands for Claim Item/Claim Service (@RelPrices =0)
Load current I/S cursor
Determine ProductID
Take policy with lowest CostSharing (tblProductItems(LimitationType,LimitAdult,LimitChild)
CareType consistant
tblHF.HFType, tblClaim.ClaimType, tblItems.ItemCareType
y
Priceorigin ='R'
Determine Price (O,P,R)
tblClaimItems.PriceAsked tblItems.ItemPrice tblPLItemsDetail.PriceOverule tblClaimItems.PriceAdjusted tblClaimItems.PriceApproved
y
SET @RelPrices to 1
tblProductItems.LimitationType tblProductItems.LimitAdult tblProductItems.LimitChild tblClaimItems.PriceAdjusted tblClaimItems.PriceApproved
Apply Limits
I/S set to Rejected
All I/S done
Load next I/S in cursor
y
Get Claim total value per product/ policy
tblClaimDedRem
Load current Claim - Product in cursor
Deduction on Policy
Deduction on Insuree
Deduction on Treatment
y
Deduction on Policy Proces *C
Deduction on Insuree Process *B
Deduction on Treatment Process *A
Load next Claim Product in cursor
All Claim Products done
y
Ceiling for Treatment
Ceiling for Insuree
Ceiling for Policy
Update tblClaimDedRem
@RelPrices = 1
y
All claims done
Load next Claim in cursor
y
Apply Ceiling on Treatment Proces *D
Apply Ceiling on Treatment Proces *E
Apply Ceiling on Treatment Proces *F
tblClaimDedRem
Claim set to Valuated
END claims valuation

5.5. Calculation of 'relative prices index' process

The process of calculation of the index of relative prices is needed to determine the actual price to be paid to the health facility for items and services that were covered with an insurance product with the price origin 'R' (relative prices) for its items and services.

The calculation of relative prices is done once a month on manual basis for the menu function 'Batch Run' under the claims menu. This function is available for restricted use, e.g. only the IMIS Scheme administrator.

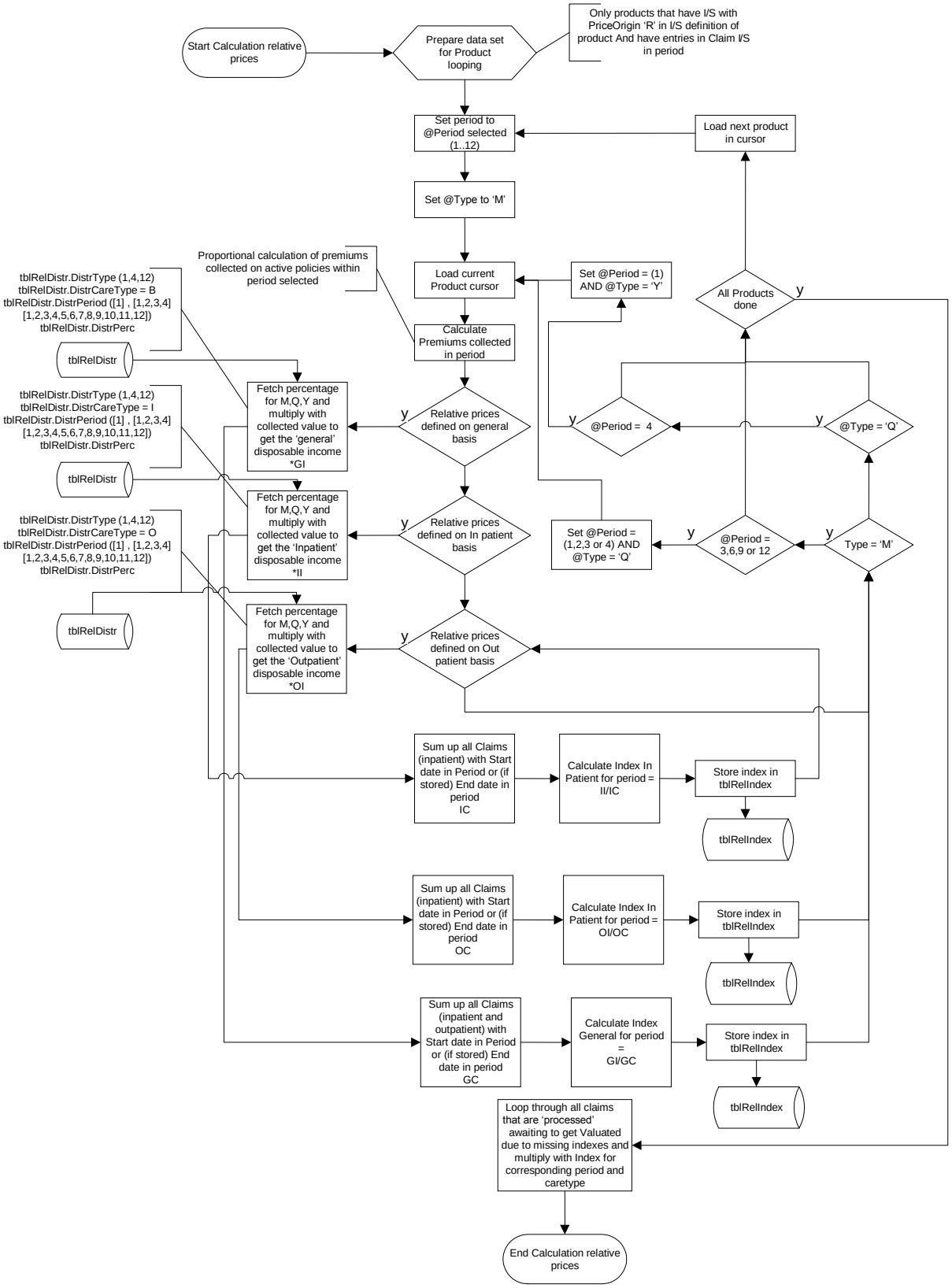
The function will first gather all Contributions collected for the period selected, e.g. March 2012. This value will be proportionally calculated as Contributions are normally prepaid for a year (or other period defined). In this calculation we will ignore the 'grace' period and late payments on Contributions. We will only use the policy value spread out over the period (default = 12) in question and use the 'slice' that will be valid for March 2012.

Now we will need to calculate the total amount of claims over the calculation period (e.g March or Quarter 1 2012). This is done by the use of the table tblClaimItems and tblClaimServices. We will check how much is claimed within the period in question by looking at the start date of health care provided or the end date of the health care in case it was inpatient care.

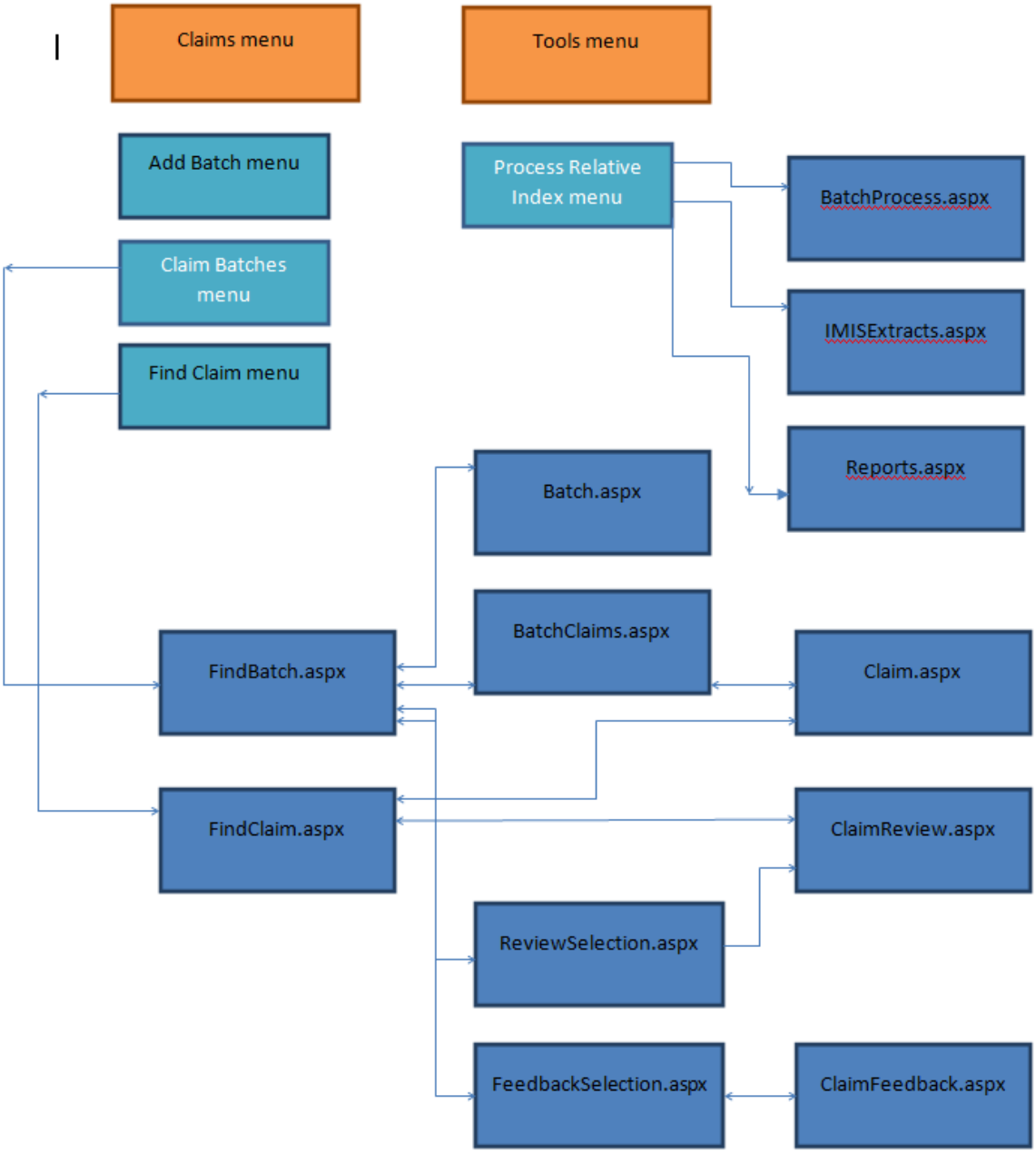
Separate indexes are calculated:

- General (all claims are used in the calculation)
- In patient care (Only claims from health facilities of HFLevel 'H' are considered)
- Out-patient care (Only claims from health facilities with HFLevel 'C' and 'D' are considered)

Please find below the flowchart of this process.


Start Calculation relative prices
Prepare data set for Product looping
Only products that have I/S with PriceOrigin 'R' in I/S definition of product And have entries in Claim I/S in period
Set period to @Period selected (1..12)
Load next product in cursor
Set @Type to 'M'
Proportional calculation of premiums collected on active policies within period selected
Load current Product cursor
Set @Period = (1) AND @Type = 'Y'
All Products done
y
tblRelDistr.DistrType (1,4,12)
tblRelDistr.DistrCareType = B
tblRelDistr.DistrPeriod ([1] , [1,2,3,4] [1,2,3,4,5,6,7,8,9,10,11,12])
tblRelDistr.DistrPerc
tblRelDistr
Calculate Premiums collected in period
Fetch percentage for M,Q,Y and multiply with collected value to get the 'general' disposable income *GI
Relative prices defined on general basis
@Period = 4
@Type = 'Q'
tblRelDistr.DistrType (1,4,12)
tblRelDistr.DistrCareType = I
tblRelDistr.DistrPeriod ([1] , [1,2,3,4] [1,2,3,4,5,6,7,8,9,10,11,12])
tblRelDistr.DistrPerc
tblRelDistr
Fetch percentage for M,Q,Y and multiply with collected value to get the 'Inpatient' disposable income *II
Relative prices defined on In patient basis
Set @Period = (1,2,3 or 4) AND @Type = 'Q'
@Period = 3,6,9 or 12
Type = 'M'
tblRelDistr.DistrType (1,4,12)
tblRelDistr.DistrCareType = O
tblRelDistr.DistrPeriod ([1] , [1,2,3,4] [1,2,3,4,5,6,7,8,9,10,11,12])
tblRelDistr.DistrPerc
tblRelDistr
Fetch percentage for M,Q,Y and multiply with collected value to get the 'Outpatient' disposable income *OI
Relative prices defined on Out patient basis
Sum up all Claims (inpatient) with Start date in Period or (if stored) End date in period IC
Calculate Index In Patient for period = II/IC
Store index in tblRelIndex
tblRelIndex
Sum up all Claims (inpatient) with Start date in Period or (if stored) End date in period OC
Calculate Index In Patient for period = OI/OC
Store index in tblRelIndex
tblRelIndex
Sum up all Claims (inpatient and outpatient) with Start date in Period or (if stored) End date in period GC
Calculate Index General for period = GI/GC
Store index in tblRelIndex
tblRelIndex
Loop through all claims that are 'processed' awaiting to get Valuated due to missing indexes and multiply with Index for corresponding period and caretype
End Calculation relative prices

5.6. Graphical page routing overview Claim interfaces

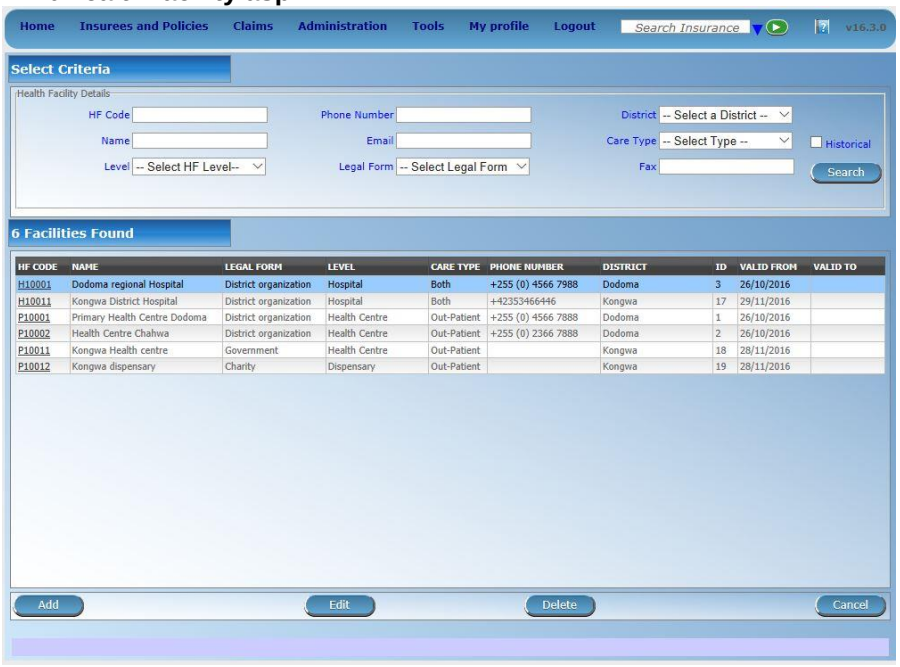


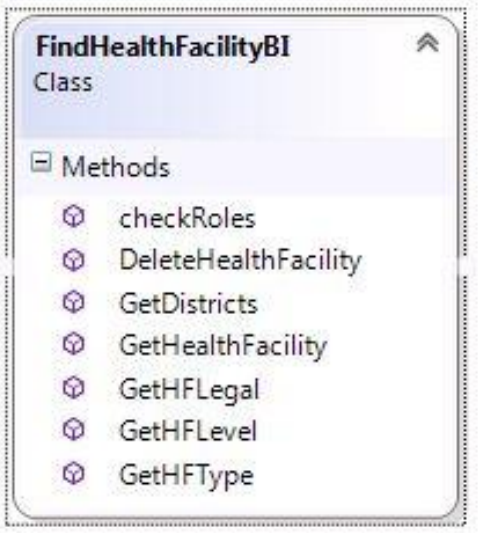
5.7. Overview of status fields

Please find below an overview of the different statuses found in several tables and the actual interface/action that would set the status in question.

Table	*Field*	*Status*	*Description*
TblClaim	**ClaimStatus**	1	Rejected
		2	Entered
		4	Checked
		8	Processed
		16	Valuated
	ReviewStatus	0	Idle
		1	Not selected
		2	Selected for Review
		4	Reviewed
		8	ByPassed
	FeedbackStatus	0	Idle
		1	Not selected
		2	Selected for Feedback
		4	Delivered
		8	ByPassed
	ApprovalStatus	1	N/U
		2	N/U
		4	N/U
TblClaimItem	**ClaimItemStatus**	1	Passed
		2	Rejected
TblClaimService	**ClaimServiceStatus**	1	Passed
		2	Rejected

5.7.1.FindHealthFacility.aspx



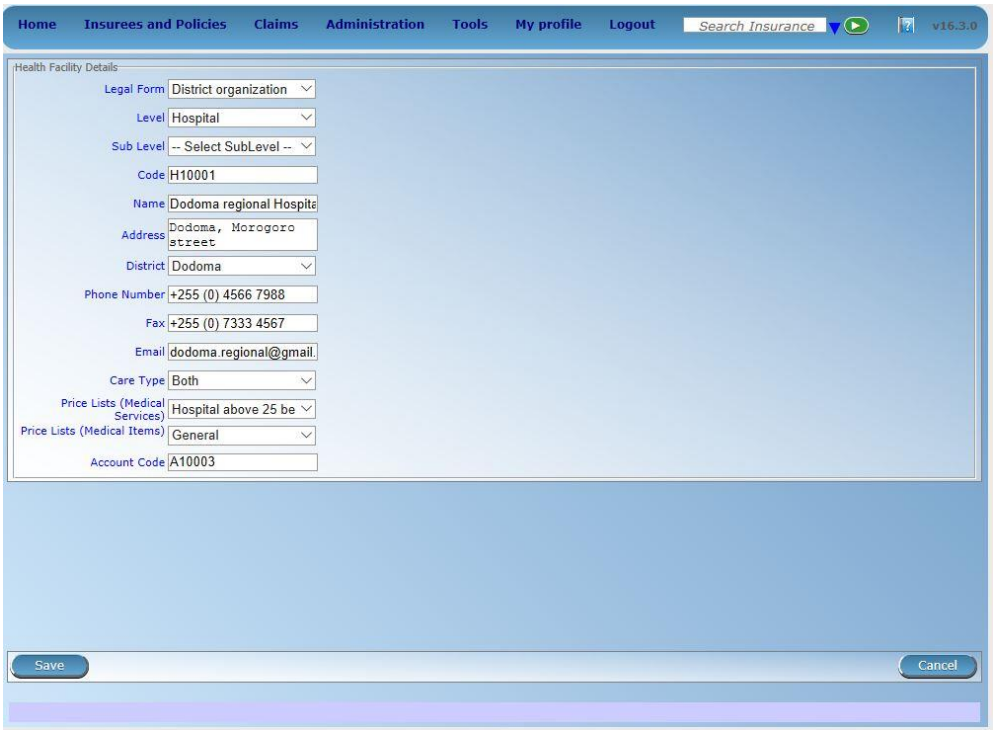
Interface object name:	*FindHealthFacility.aspx*
Menu path:	*Health Facilities*
Hyperlinks/Redirections:	
Action Button:	*HealthFacility.aspx (Save and Cancel button)*
Use case Reference	*6.2.17(Finding Health Facility)* *6.2.19 (Deleting Health Facility)*
Class Diagram	

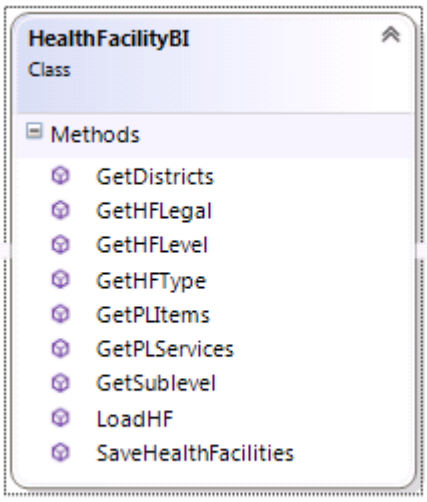
Brief description:

This interface opens up when the user has clicked the 'Health Facilities' button option on the top menu (Administration). This interface is used to search for Health Facilities and acts as the selector for additions, modifications and deletions on the entity Health facility.

The operator can enter the search criteria and click the search button. All records satisfying the criteria will appear. A click on the hyperlink on the first column (Name) in the grid will open up the health facility page. Any further operations on Health Facilities will take action from this page.

5.7.2.HealthFacility.aspx

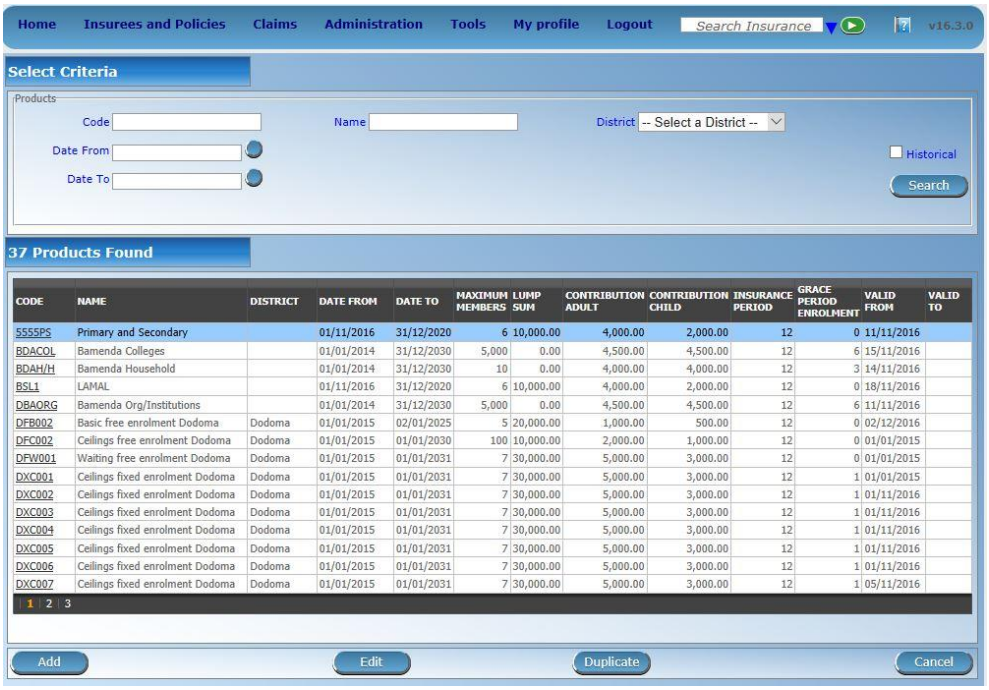


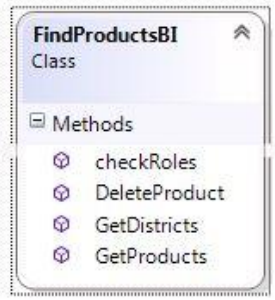
Interface object name:	*HealthFacility.aspx*
Menu path:	
Hyperlinks/Redirections:	*FindHealthFacility.aspx (name column)*
Action Button:	*FindHealthFacility.aspx (Add and Edit button)*
Use case Reference	*6.2.16 (Add Health Facility)* *6.2.18 (Modify health Facility)*
Class Diagram	

Brief description:
This interface opens up when the user has clicked the Add or Edit button on the health facility search page (or clicked the hyperlink). The health facility information can be changed as per defined 'use-cases' for adding and editing a health facility.

On saving a health facility, it will be added /refreshed on the data grid on the health facilities search page as described before.

5.7.3.FindProduct.aspx



Interface object name:	*FindProduct.aspx*
Menu path:	*Products*
Hyperlinks/Redirections:	
Action Button:	*Product.aspx (Save and Cancel button)*
Use case Reference	*6.2.14(Finding Insurance Product)* *N/A (Deleting Insurance Product)*
Class Diagram	

Brief description:

This interface opens up when the user has clicked the 'Products' button option on the top menu (Administration). This interface is used to search for Products and acts as the selector for additions, modifications and deletions on the entity Product.

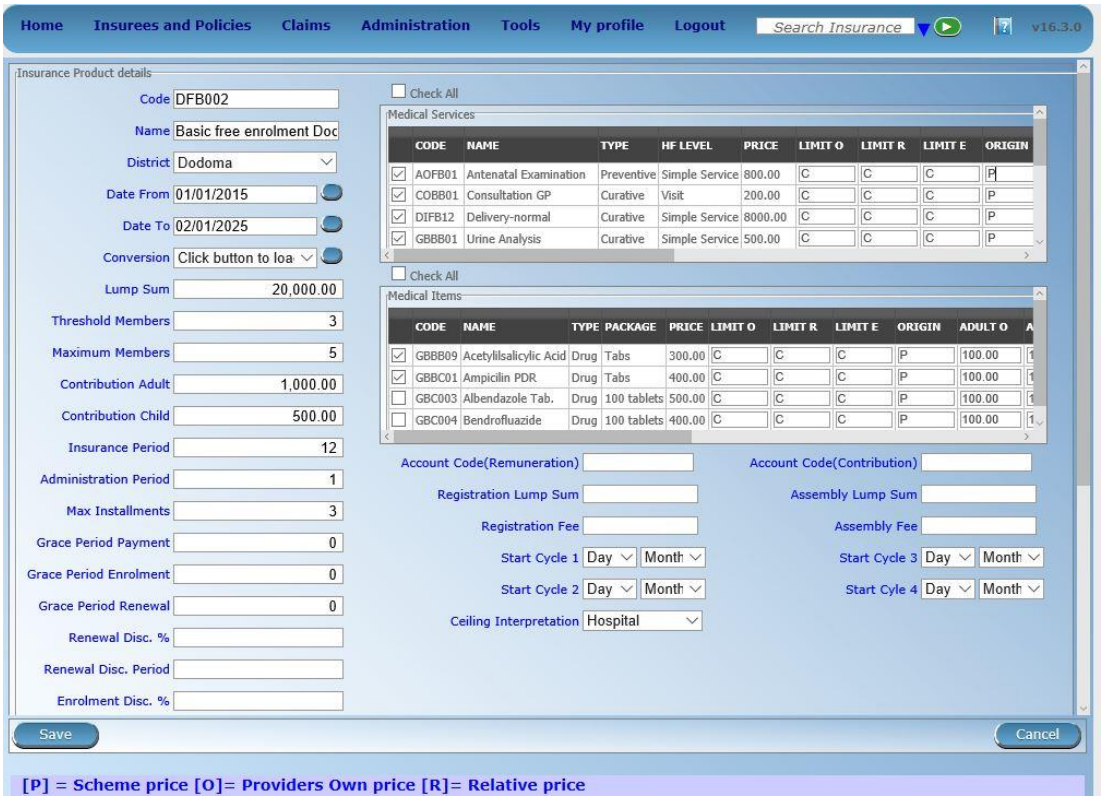
The operator can enter the search criteria and click the search button. All records satisfying the criteria will appear. A click on the hyperlink on the first column (Product Code) in the grid will open up the product page. Any further operations on products will take action from this page. The delete button is not covered by use cases but we foresee data entry errors and the need for deletions on products. Validations will take place before a product can be deleted.

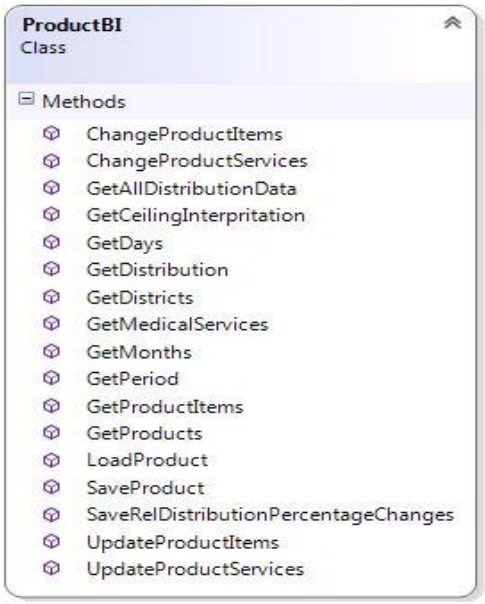
An extra button is added for duplicating a Product into a new Product. This will be handy as perhaps a new product is more or less similar as another product with just a few changes on perhaps the item/services or pricing parameters.

To duplicate a product:

Search and select the product to be duplicated in the grid. Click the 'Duplicate' button. Automatically the system will now will open up the product page and insert (in memory) the new product with its details (including items and services) as a duplicate of the original. The operator can make further modifications to the information and could decide to save the 'duplicated' product. In this case the normal procedure for saving will take place. If the operator cancels the entry, nothing will be saved.

5.7.4.Product.aspx



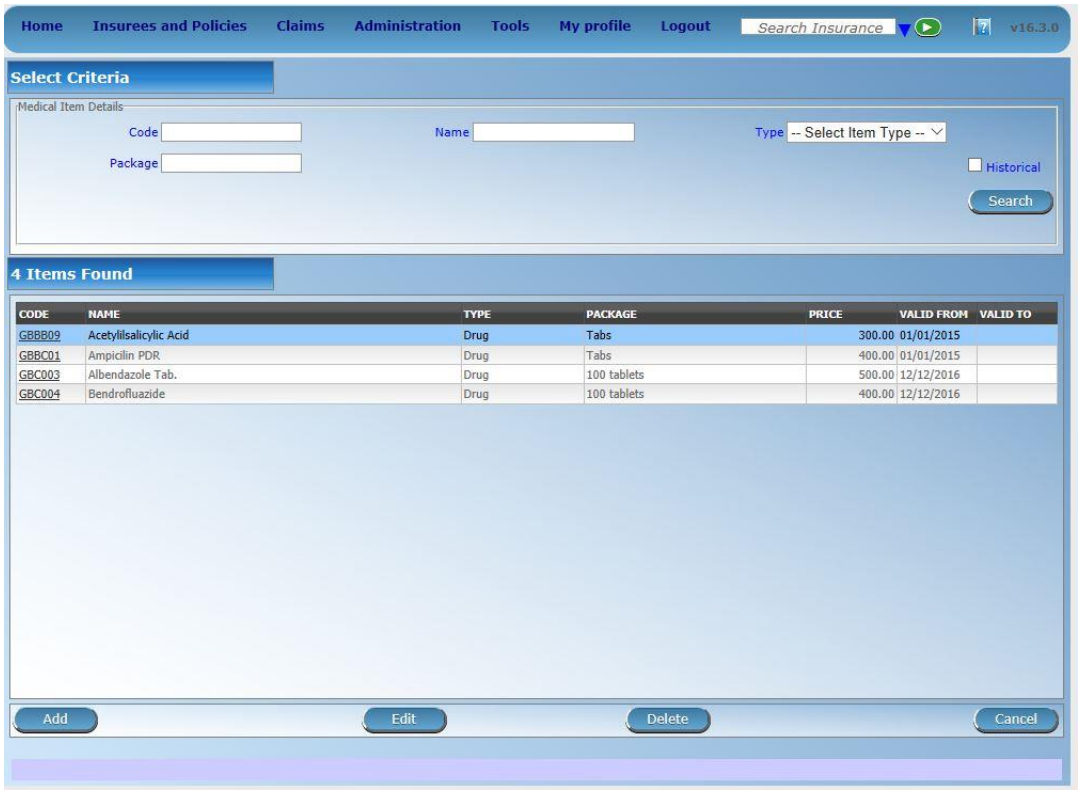
Interface object name:	*Product.aspx*
Menu path:	
Hyperlinks/Redirections:	*FindProduct.aspx (Product Code)*
Action Button:	*FindProduct.aspx (Add and Edit button)*
Use case Reference	*6.2.13 (Add Insurance Product)* *6.2.15 (Modify Insurance Product)*
Class Diagram	

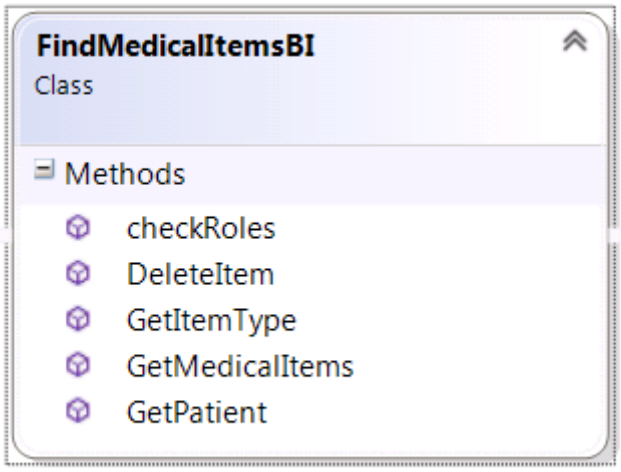
Brief description:
This interface opens up when the user has clicked the Add or Edit button on the product search page (or clicked the hyperlink). The product information can be changed as per defined 'use-cases' for adding and editing a product. The product page will have 2 separate sections for the items and services covered in the product. The check all checkbox will allow quick selection of all items or services.
Product Deductibles and Product Ceilings will be defined as per Treatment or Insuree or Policy. (Not a combination)
Furthermore we could define the ceilings and deductions on general level. In that case we are not able to define these parameters also separately for 'in' and –'out' patient level.

The product distribution information is used for the calculation of relative indexes.
The distribution will be set at Monthly, Quarterly or Yearly level (independent for general, in-patient and out-patient). At the moment of changing the setting, new records will be populated (1 for yearly, 4 for quarterly and 12 for monthly). The operator will now use the separate grids (period-percentage) with scroll bar to view and/or edit the distribution percentages.

On saving a product it will be added /refreshed on the data grid within the product search page as described before.

5.7.5.FindMedicalItem.aspx



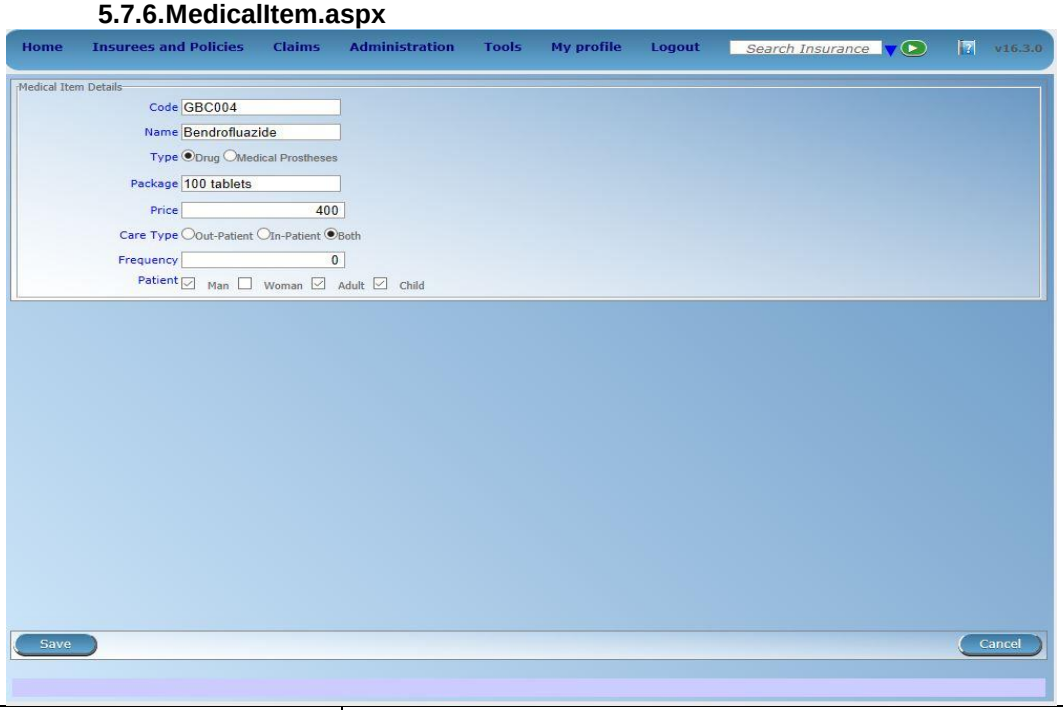
Interface object name:	*FindMedicalItem.aspx*
Menu path:	*Medical Items*
Hyperlinks/Redirections:	
Action Button:	*MedicalItem.aspx (Save and Cancel button)*
Use case Reference	*6.2.2(Finding Medical Item)* *6.2.4 (Deleting Medical Item)*
Class Diagram	

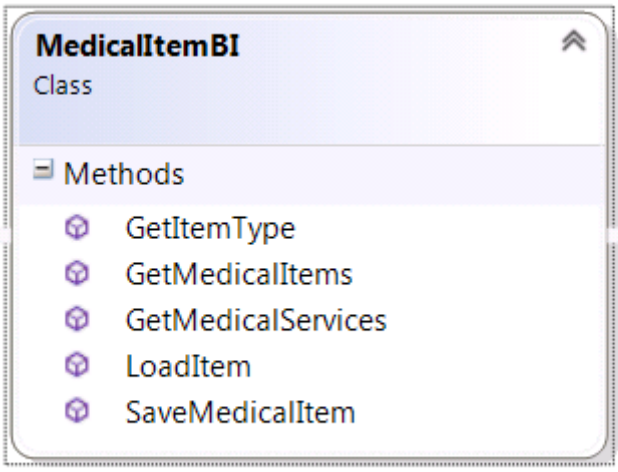
Brief description:

This interface opens up when the user has clicked the 'Medical Items' button option on the top menu (Administration). This interface is used to search for medical items and acts as the selector for additions, modifications and deletions on the entity Medical Item.

The operator can enter the search criteria and click the search button. All records satisfying the criteria will appear. A click on the hyperlink on the first column (Code) in the grid will open up the medical item page. Any further operations on medical items will take action from this page.

5.7.6.MedicalItem.aspx

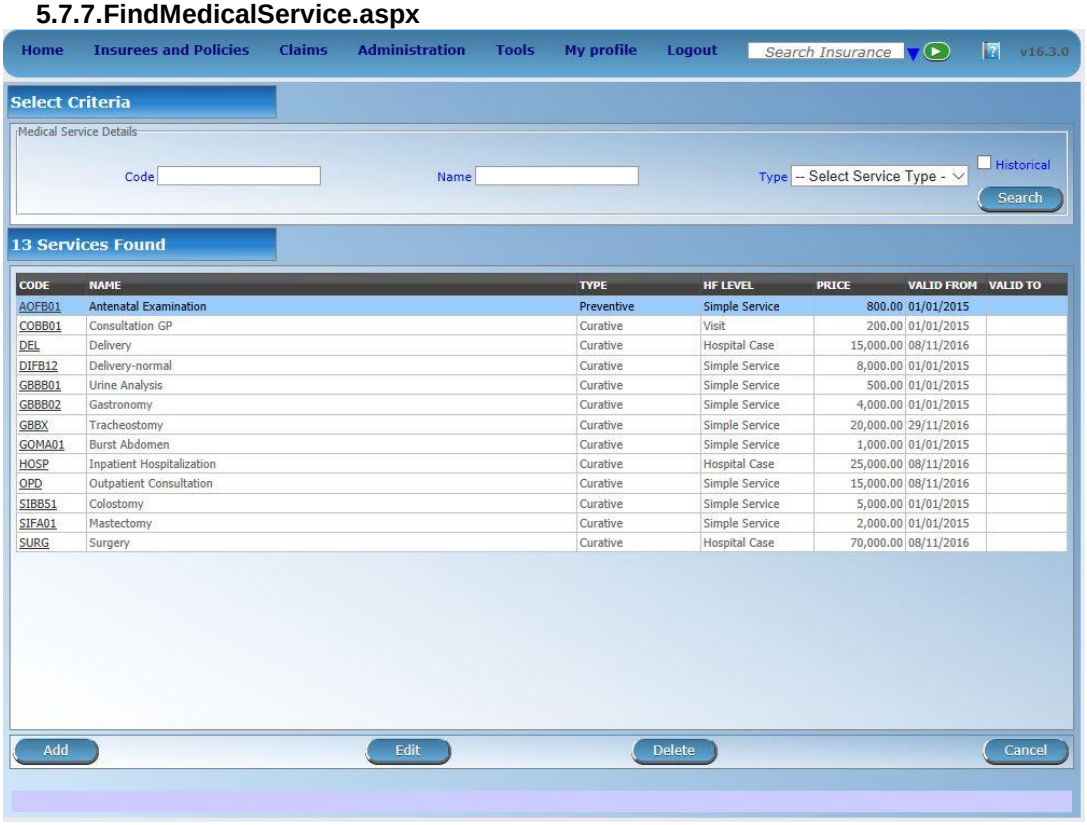


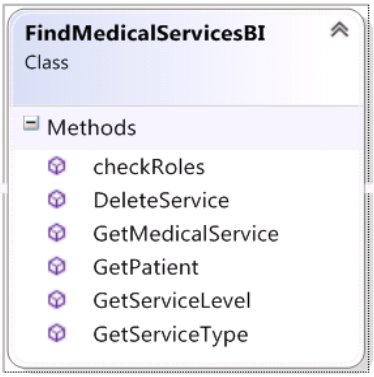
Interface object name:	*MedicalItem.aspx*
Menu path:	
Hyperlinks/Redirections:	*FindMedicalItem.aspx (Code column)*
Action Button:	*FindMedicalItem.aspx (Add and Edit button)*
Use case Reference	*6.2.5 (Add Medical Item)* *6.2.7 (Modify Medical Item)*
Class Diagram	

Brief description:
This interface opens up when the user has clicked the Add or Edit button on the Medical Items search page (or clicked the hyperlink). The medical item information can be changed as per defined 'use-cases' for adding and editing a medical item.

On saving a medical item, it will be added /refreshed on the data grid on the medical items search page as described before.

5.7.7.FindMedicalService.aspx



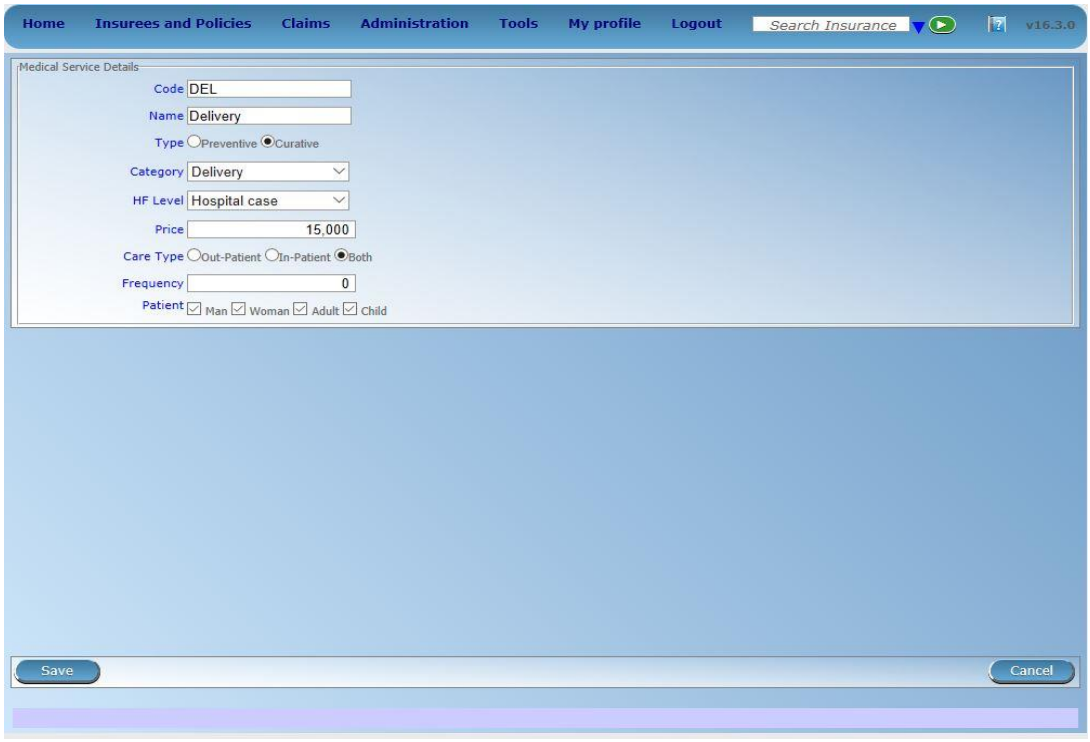
Interface object name:	*FindMedicalService.aspx*
Menu path:	*Medical Services*
Hyperlinks/Redirections:	
Action Button:	*MedicalService.aspx (Save and Cancel button)*
Use case Reference	*6.2.2(Finding Medical Service)* *6.2.4 (Deleting Medical Service)*
Class Diagram	

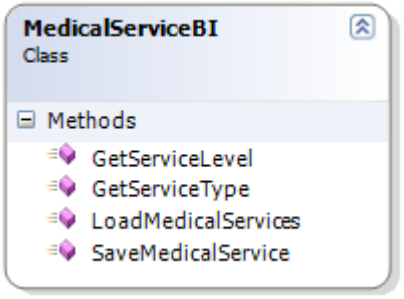
Brief description:

This interface opens up when the user has clicked the 'Medical Services' button option on the top menu (Administration). This interface is used to search for medical services and acts as the selector for additions, modifications and deletions on the entity Medical Service.

The operator can enter the search criteria and click the search button. All records satisfying the criteria will appear. A click on the hyperlink on the first column (Code) in the grid will open up the medical service page. Any further operations on medical services will take action from this page.

5.7.8.MedicalService.aspx

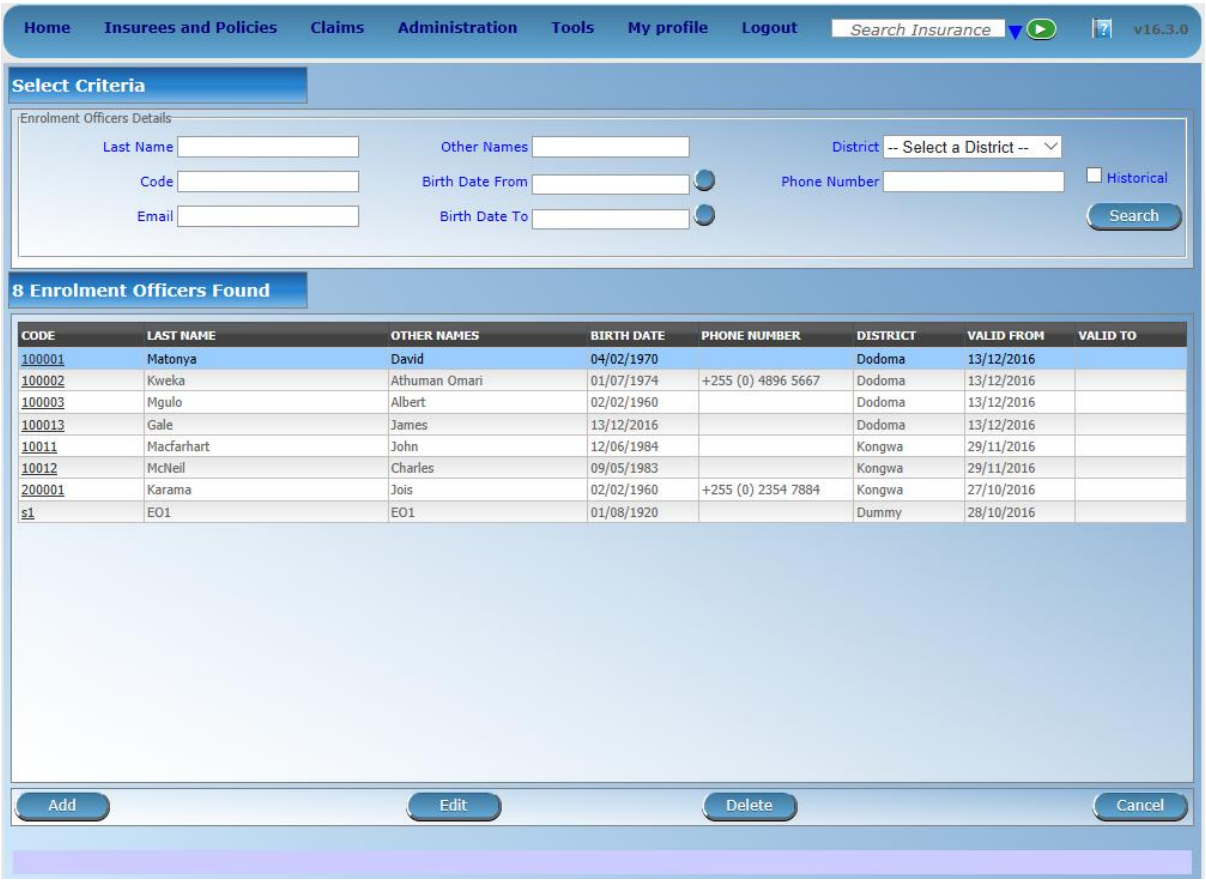


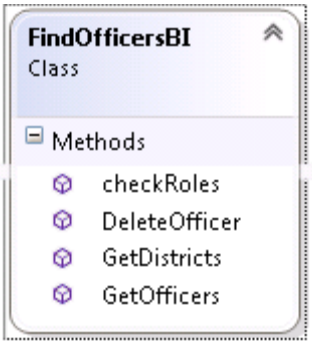
Interface object name:	*MedicalService.aspx*
Menu path:	
Hyperlinks/Redirections:	*FindMedicalService.aspx (Code column)*
Action Button:	*FindMedicalService.aspx (Add and Edit button)*
Use case Reference	*6.2.5 (Add Medical Service)* *6.2.7 (Modify Medical Service)*
Class Diagram	

Brief description:
This interface opens up when the user has clicked the Add or Edit button on the medical services search page (or clicked the hyperlink). The medical service information can be changed as per defined 'use-cases' for adding and editing a medical service.

On saving a medical service it will be added /refreshed on the data grid on the medical services search page as described before.

5.7.9.FindOfficer.asp



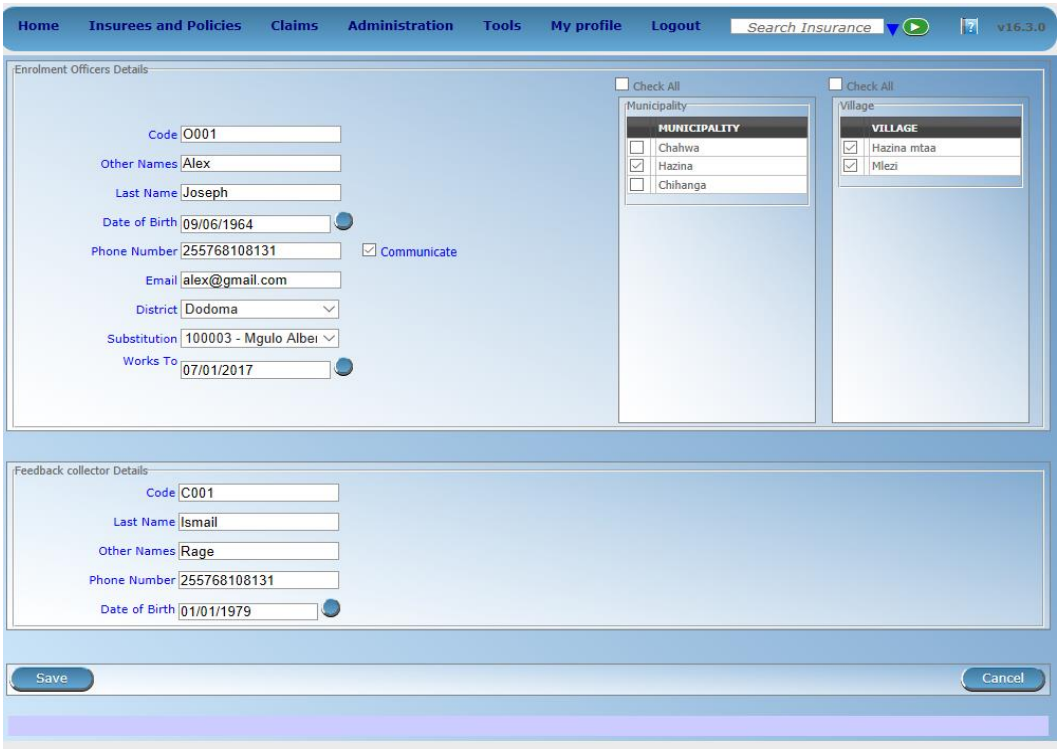
Interface object name:	*FindOfficer.aspx*
Menu path:	*Enrollment Assistant*
Hyperlinks/Redirections:	
Action Button:	*Officer.aspx (Save and Cancel button)*
Use case Reference	*6.2.10(Finding Officers)* *6.2.12 (Deleting Officers)*
Class Diagram	

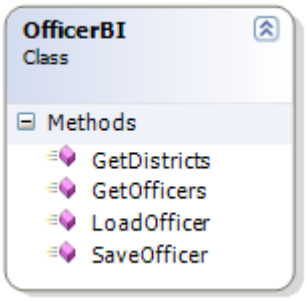
Brief description:

This interface opens up when the user has clicked the 'Officers' button option on the top menu (Administration). This interface is used to search for officers and acts as the selector for additions, modifications and deletions on the entity Enrolment Officer.

The operator can enter the search criteria and click the search button. All records satisfying the criteria will appear. A click on the hyperlink on the first column (Code) in the grid will open up the officer's page. Any further operations on officers will take action from this page.

5.7.10. Officer.aspx

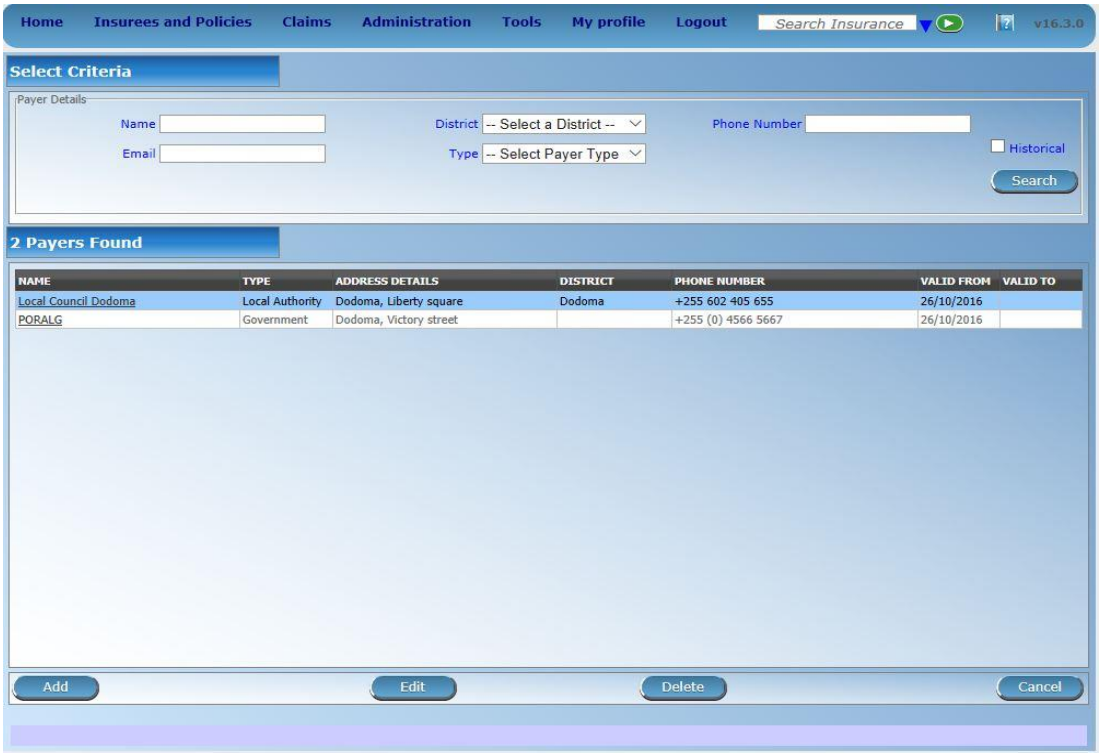


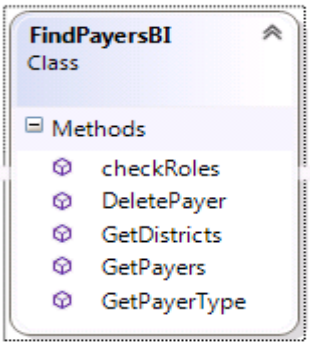
Interface object name:	*Officer.aspx*
Menu path:	
Hyperlinks/Redirections:	*FindOfficer.aspx (Code column)*
Action Button:	*FindOfficer.aspx (Add and Edit button)*
Use case Reference	*6.2.9 (Add Officer)* *6.2.11 (Modify officer)*
Class Diagram	

Brief description:
This interface opens up when the user has clicked the Add or Edit button on the officer search page (or clicked the hyperlink). The officer information can be changed as per defined 'use-cases' for adding and editing an officer.

On saving an officer, he/she will be added /refreshed on the data grid on the officer search page as described before.

5.7.11. FindPayer.aspx

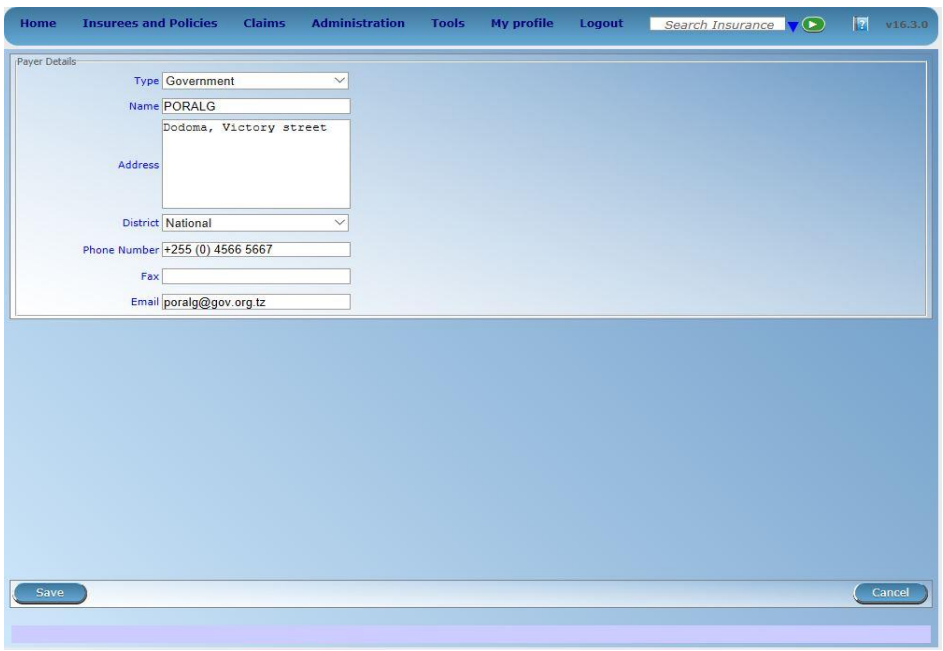


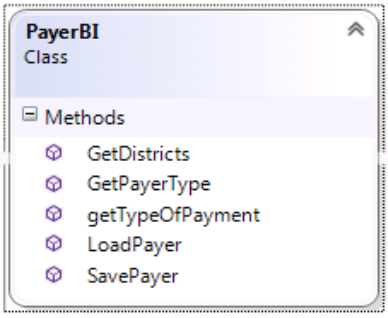
Interface object name:	*FindPayer.aspx*
Menu path:	*Payers*
Hyperlinks/Redirections:	
Action Button:	*Payer.aspx (Save and Cancel button)*
Use case Reference	*6.2.6(Finding Payers)* *6.2.8 (Deleting Payers)*
Class Diagram	

Brief description:

This interface opens up when the user has clicked the 'Payers' button option on the top menu (Administration). This interface is used to search for payers and acts as the selector for additions, modifications and deletions on the entity Payer.
The operator can enter the search criteria and click the search button. All records satisfying the criteria will appear. A click on the hyperlink on the first column (Name… **Note**. to be changed in the field order) in the grid will open up the payers page. Any further operations on payers will take action from this page.

5.7.12. Payer.aspx

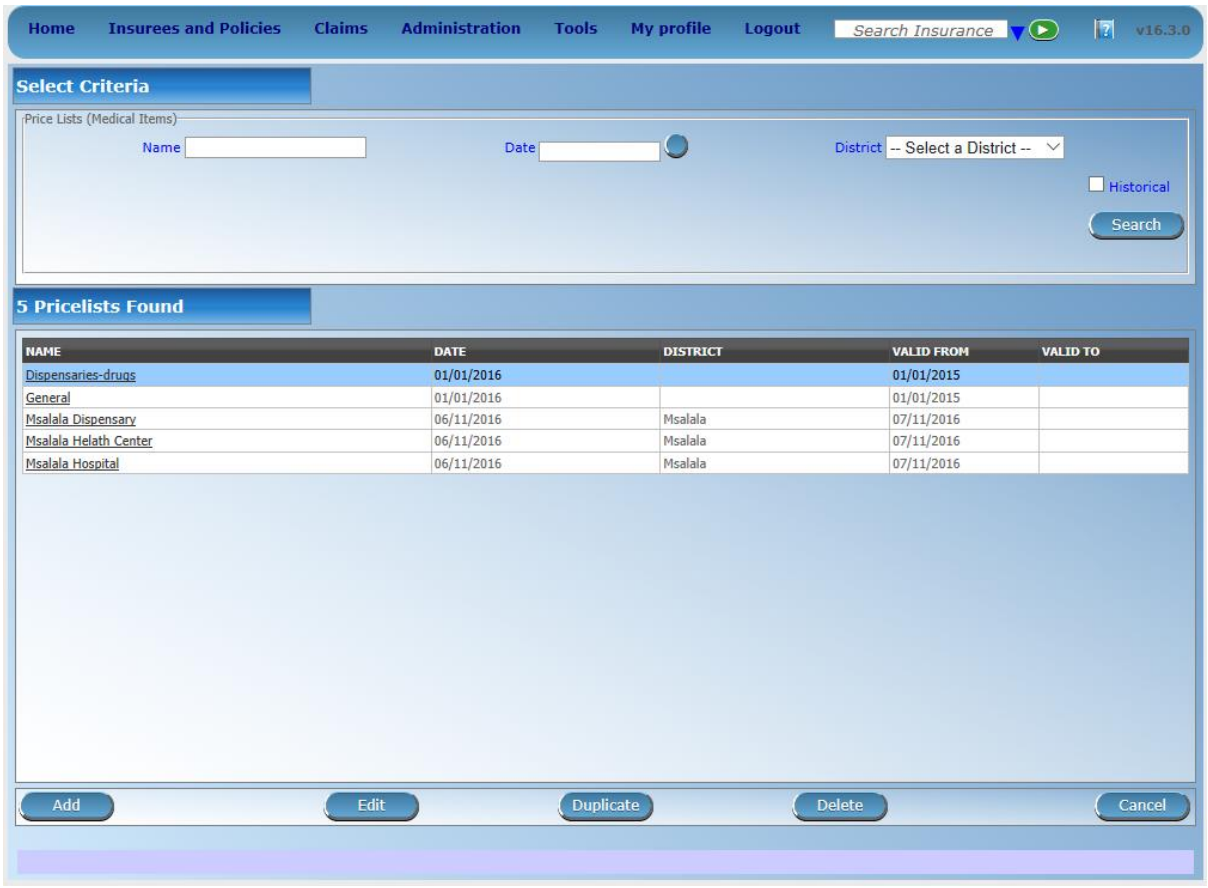


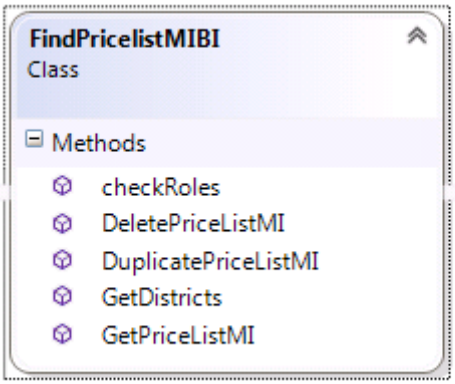
Interface object name:	*Payer.aspx*
Menu path:	
Hyperlinks/Redirections:	*FindPayer.aspx (Name column)*
Action Button:	*Findpayer.aspx (Add and Edit button)*
Use case Reference	*6.2.5 (Add payer)* *6.2.7 (Modify payer)*
Class Diagram	

Brief description:
This interface opens up when the user has clicked the Add or Edit button on the payer search page (or clicked the hyperlink). The payer information can be changed as per defined 'use-cases' for adding and editing a Payer.

On saving a payer, the new entry will be added /refreshed on the data grid on the payer s search page as described before.

5.7.13. FindPriceListMI.aspx



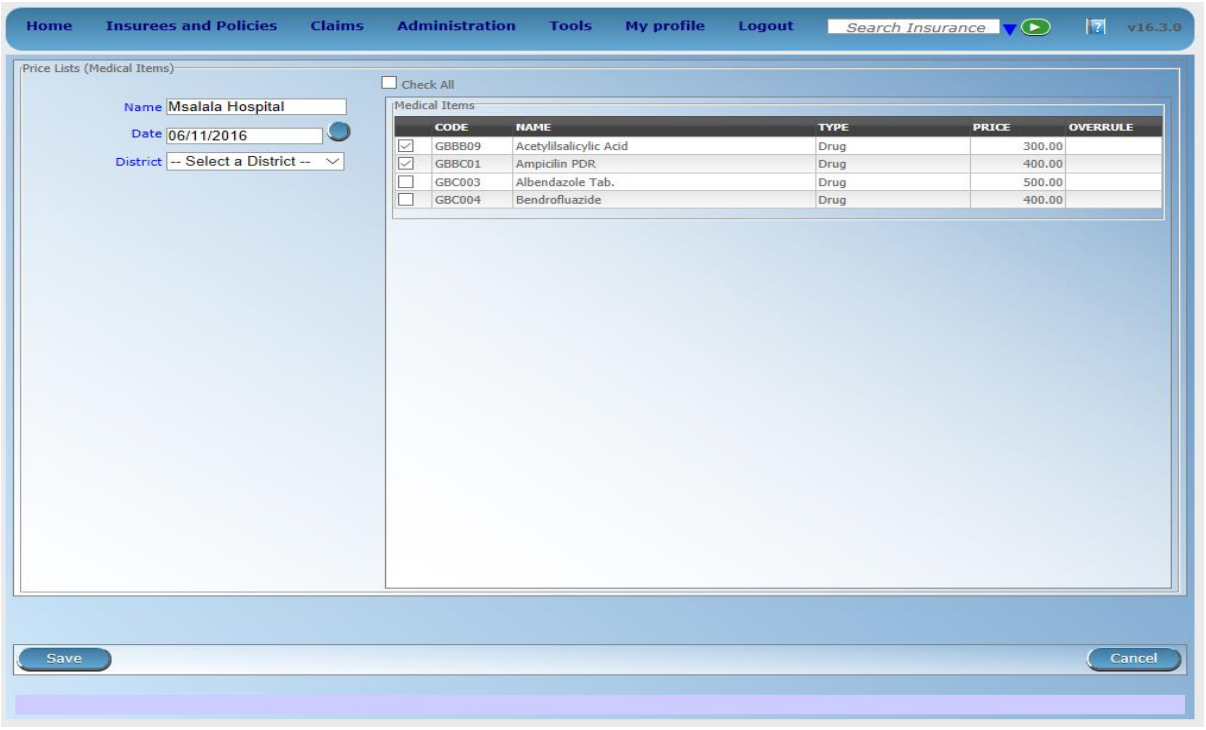
Interface object name:	*FindPricelistMI.aspx*
Menu path:	*Pricelists - Medical Items*
Hyperlinks/Redirections:	
Action Button:	*PriceListMI.aspx (Save and Cancel button)*
Use case Reference	*6.2.21(Finding Pricelist medical Items)* *6.2.23 (Deleting Pricelist medical items)*
Class Diagram	

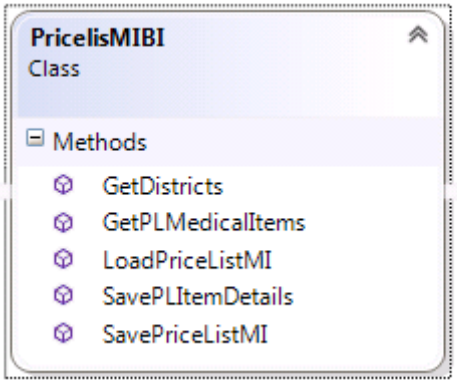
Brief description:

This interface opens up when the user has clicked the 'Medical Items' button option on the top menu Administration, sub menu Pricelists. This interface is used to search for Pricelists of medical items and acts as the selector for additions, modifications and deletions on the entity Pricelist of Items.

The operator can enter the search criteria and click the search button. All records satisfying the criteria will appear. A click on the hyperlink on the first column (Name) in the grid will open up the pricelist medical items page. Any further operations on pricelist of medical items will take action from this page.

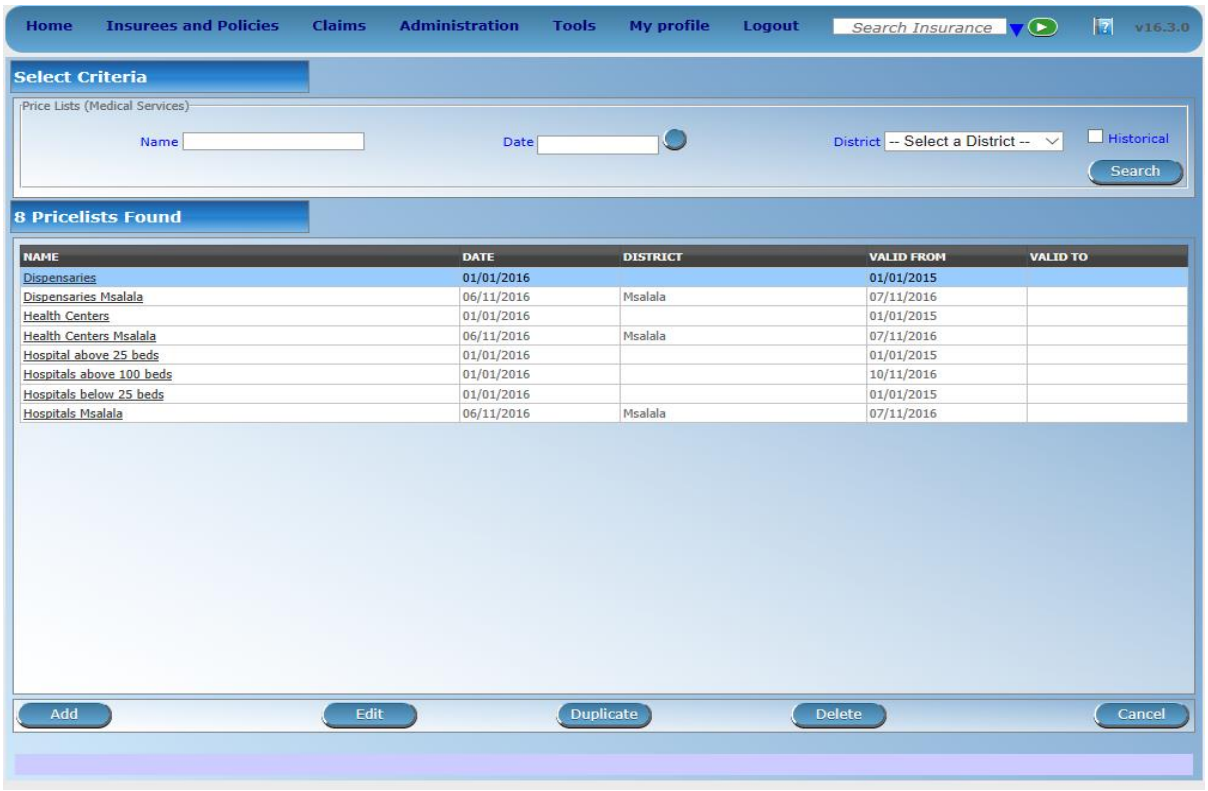
5.7.14. PriceListMI.aspx

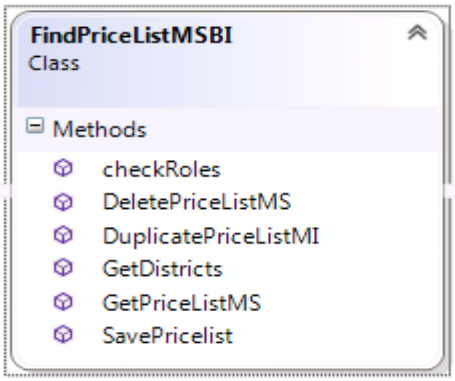


Interface object name:	*PriceListMI.aspx*
Menu path:	
Hyperlinks/Redirections:	*FindPricelistMI.aspx (Name column)*
Action Button:	*FindPriceListMI.aspx (Add and Edit button)*
Use case Reference	*6.2.20 (Add Pricelist Medical Items)* *6.2.22 (Modify Pricelist Medical Items)*
Class Diagram	

Brief description:
This interface opens up when the user has clicked the Add or Edit button on the Pricelist medical items search page (or clicked the hyperlink). The pricelist (of medical items) information can be changed as per defined 'use-cases' for adding and editing a pricelist of medical items.
On saving a pricelist of medical items, the new entry will be added /refreshed on the data grid on the pricelist of medical items search page as described before.

5.7.15. FindPriceListMS.aspx



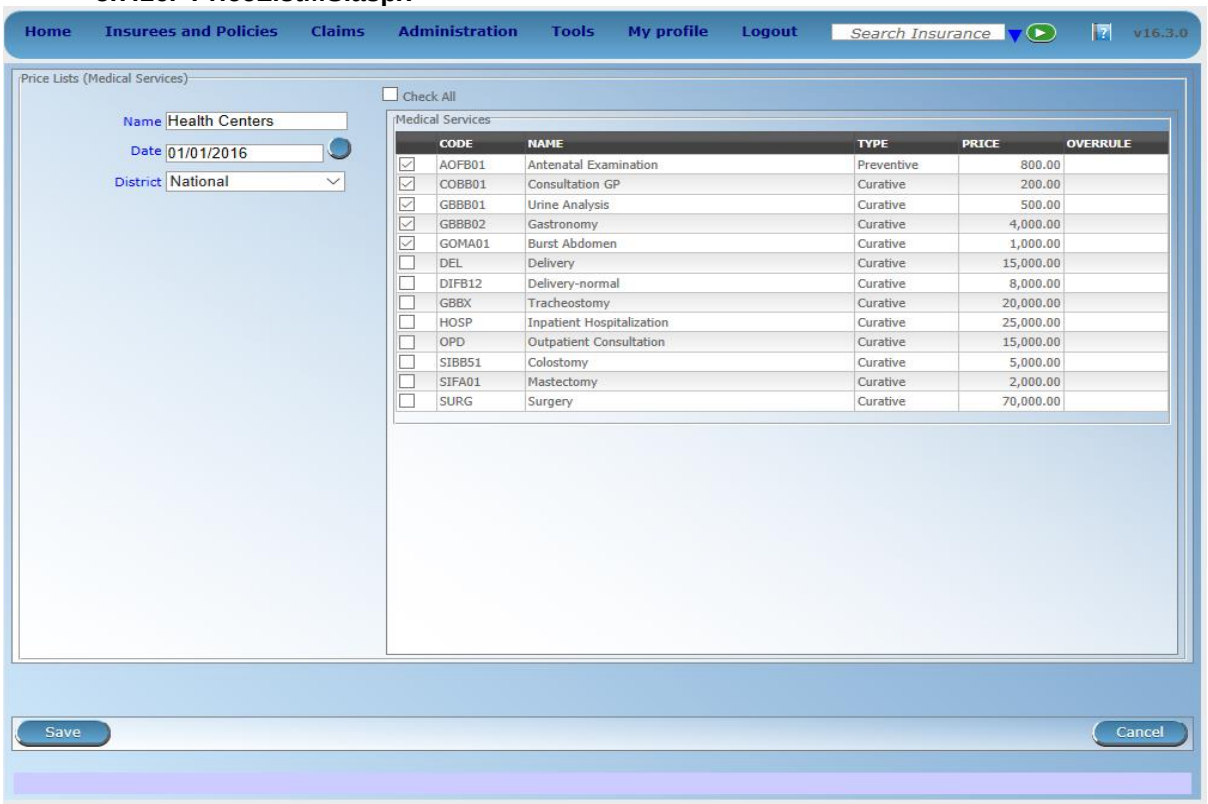
Interface object name:	*FindPricelistMS.aspx*
Menu path:	*Pricelists - Medical Services*
Hyperlinks/Redirections:	
Action Button:	*PriceListMS.aspx (Save and Cancel button)*
Use case Reference	*6.2.21(Finding Pricelist medical services)* *6.2.23 (Deleting Pricelist medical services)*
Class Diagram	

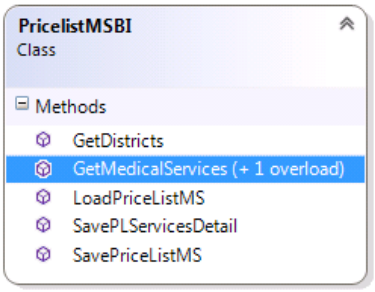
Brief description:

This interface opens up when the user has clicked the 'Medical Services' button option on the top menu Administration, sub menu PriceLists. This interface is used to search for PriceLists of medical services and acts as the selector for additions, modifications and deletions on the entity Pricelist of Services.

The operator can enter the search criteria and click the search button. All records satisfying the criteria will appear. A click on the hyperlink on the first column (Name) in the grid will open up the pricelist medical services page. Any further operations on pricelist of medical services will take action from this page.

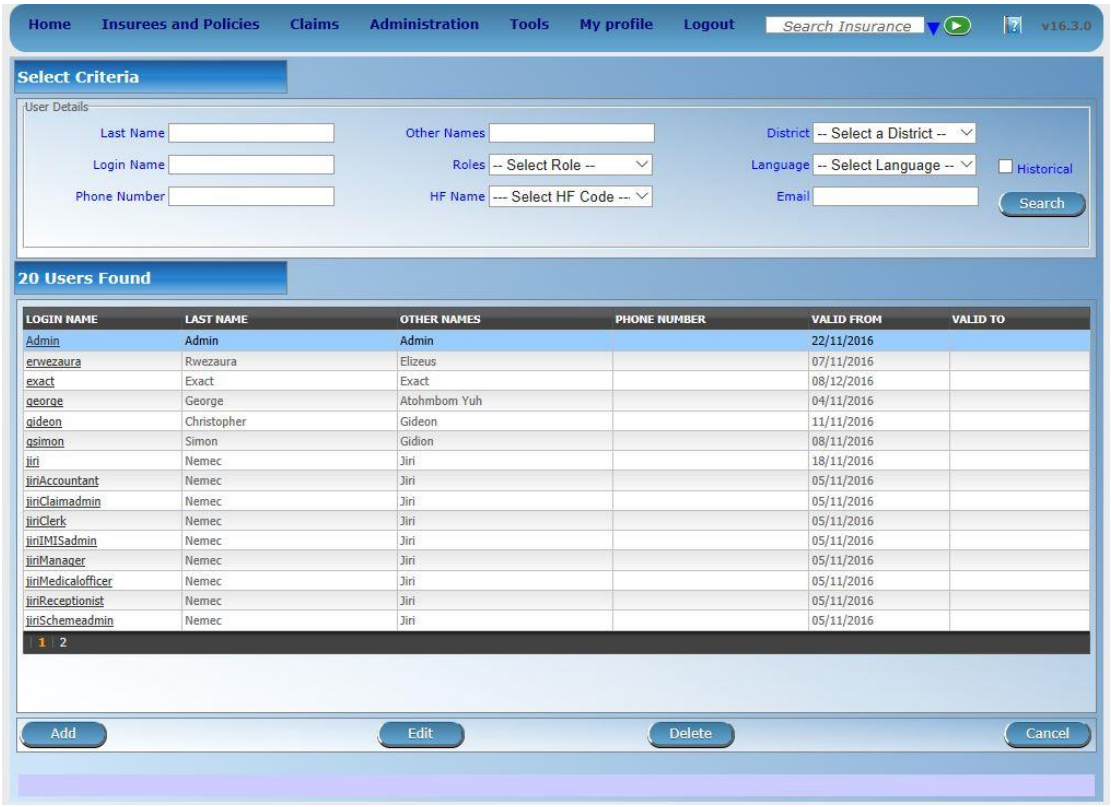
5.7.16. PriceListMS.aspx

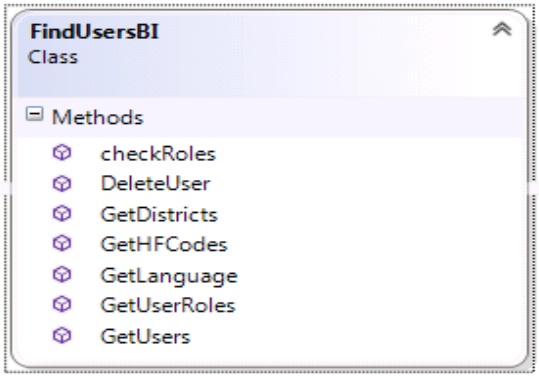


Interface object name:	*PriceListMS.aspx*
Menu path:	
Hyperlinks/Redirections:	*FindPricelistMS.aspx (Name column)*
Action Button:	*FindPriceListMS.aspx (Add and Edit button)*
Use case Reference	*6.2.20 (Add Pricelist Medical Services)* *6.2.22 (Modify Pricelist Medical Services)*
Class Diagram	

Brief description:
This interface opens up when the user has clicked the Add or Edit button on the Pricelist medical services search page (or clicked the hyperlink). The pricelist (of medical services) information can be changed as per defined 'use-cases' for adding and editing a pricelist of medical services.
On saving a pricelist of medical services, the new entry will be added /refreshed on the data grid on the pricelist of medical services search page as described before.

5.7.17. FindUser.aspx



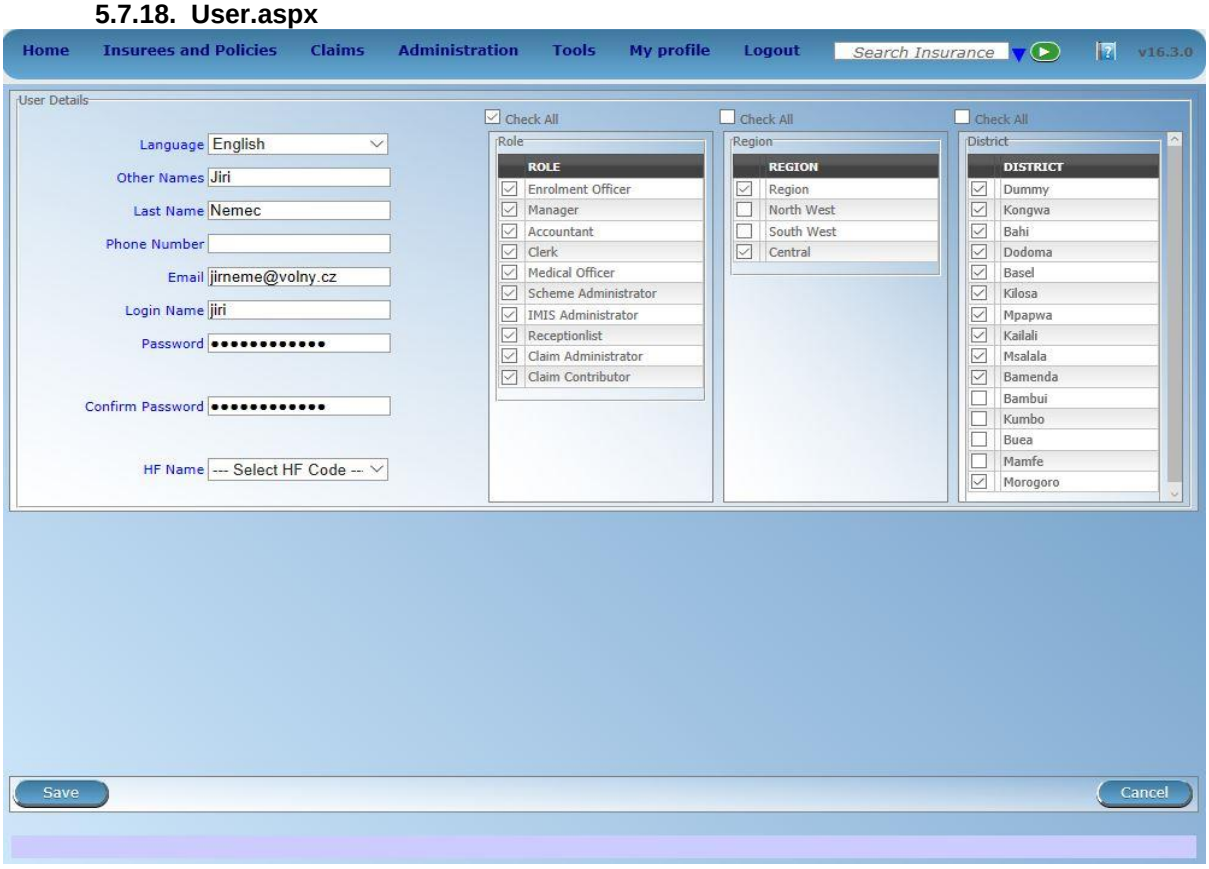
Interface object name:	*FindUser.aspx*
Menu path:	*Users*
Hyperlinks/Redirections:	
Action Button:	*User.aspx (Save and Cancel button)*
Use case Reference	*6.2.2(Finding Users)* *6.2.4 (Deleting Users)*
Class Diagram	

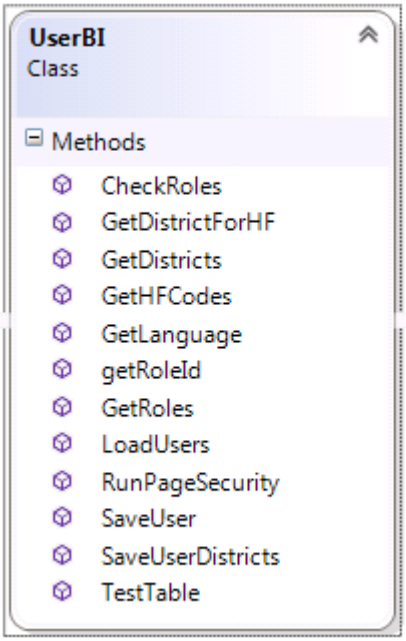
Brief description:

This interface opens up when the user has clicked the 'Users' button option on the top menu Administration. This interface is used to search for users and acts as the selector for additions, modifications and deletions on the entity User.

The operator can enter the search criteria and click the search button. All records satisfying the criteria will appear. A click on the hyperlink on the first column (Login Name) in the grid will open up 'user' page. Any further operations on users will take action from this page.

5.7.18. User.aspx

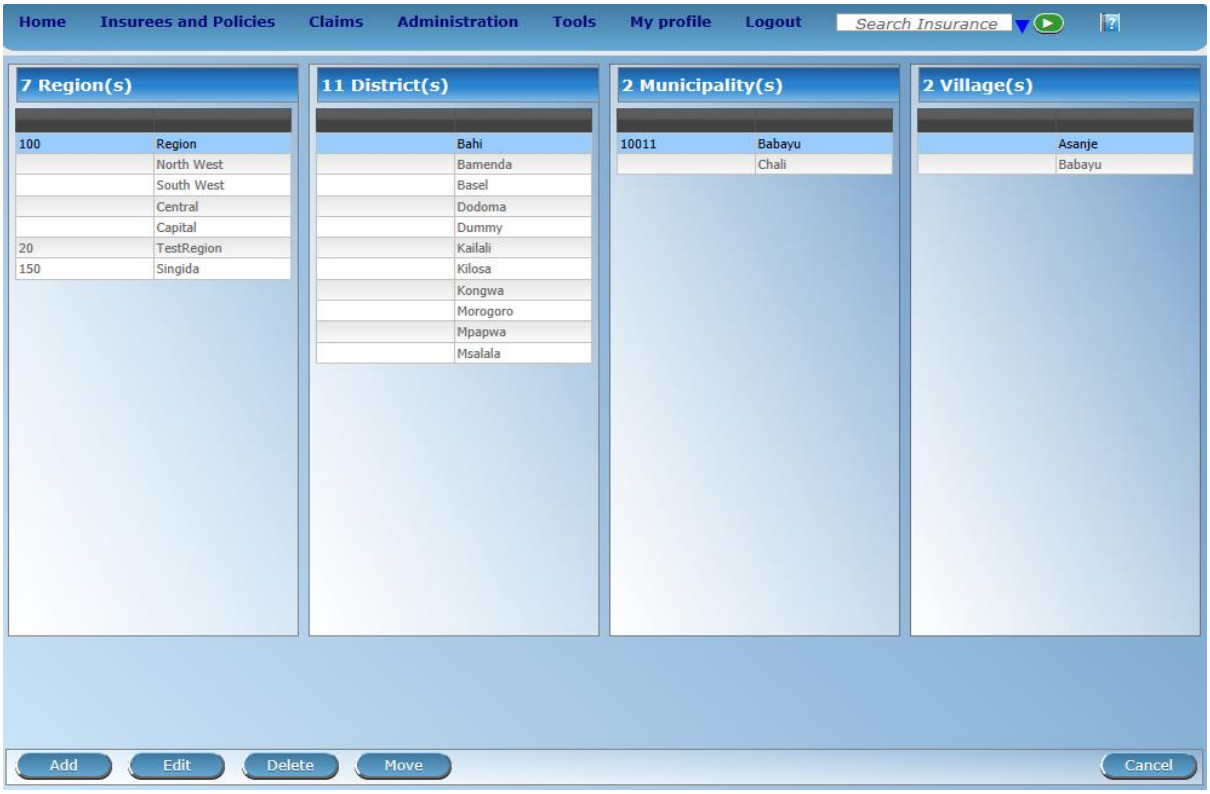


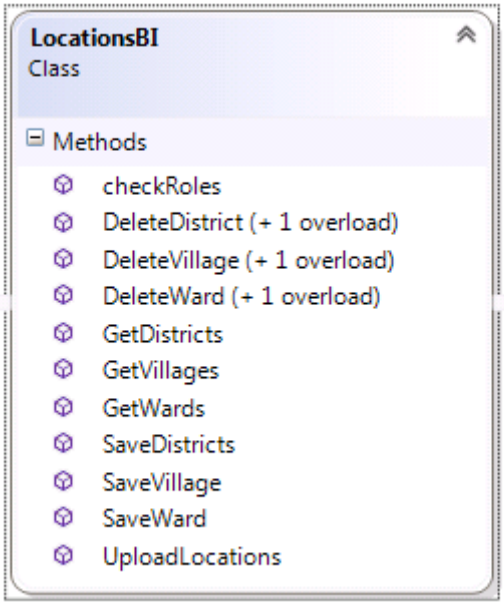
Interface object name:	*User.aspx*
Menu path:	
Hyperlinks/Redirections:	*FindUser.aspx (Name column)*
Action Button:	*FindUser.aspx (Add and Edit button)*
Use case Reference	*6.2.1 (Add User)* *6.2.3 (Modify User)*
Class Diagram	

Brief description:
This interface opens up when the user has clicked the Add or Edit button on the users search page (or clicked the hyperlink). The user information can be changed as per defined 'use-cases' for adding and editing a user.

On saving a user, the new entry will be added /refreshed on the data grid on the users search page as described before.

5.7.19. Locations.aspx

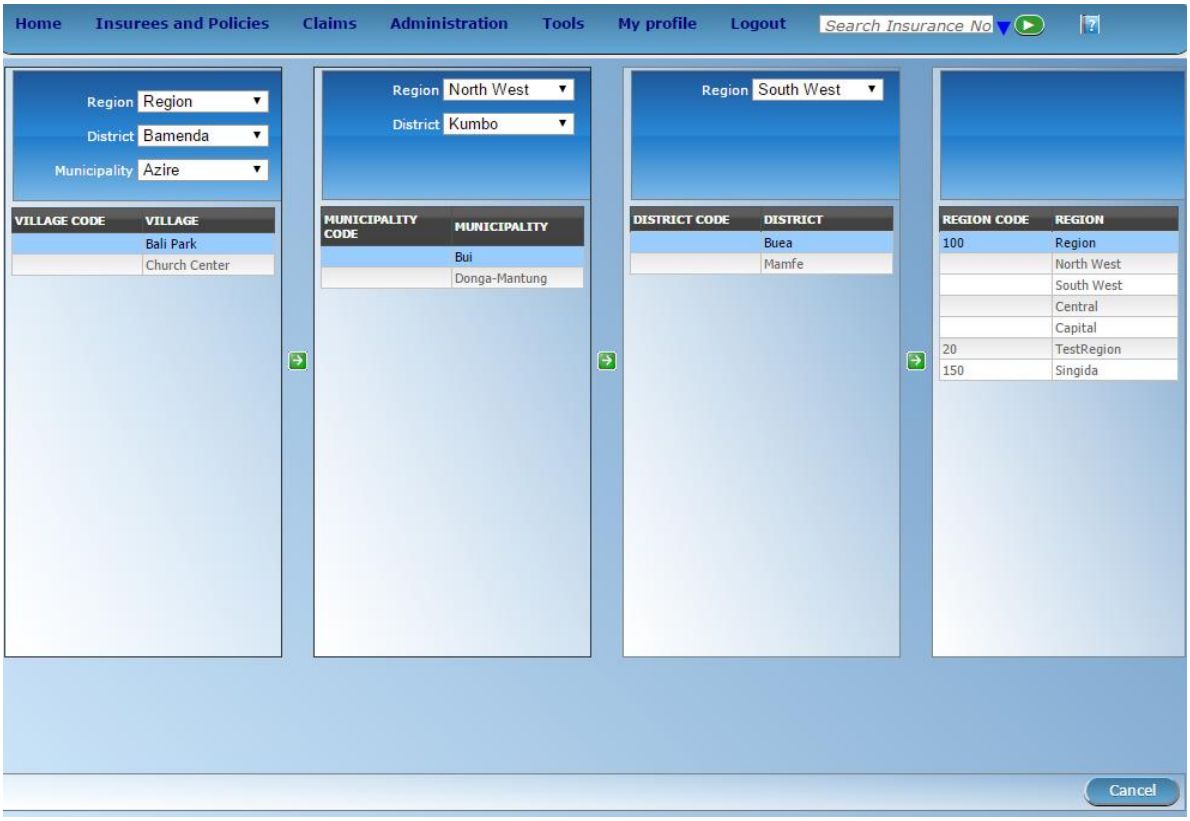


Interface object name:	*Locations.aspx*
Menu path:	*Locations*
Hyperlinks/Redirections:	
Action Button:	
Use case Reference	*6.2.24 (Uploading code lists)*
Class Diagram	

Brief description:
This interface opens up when the user has clicked the menu option 'Locations' in the Administration menu. Although no real use-case has been involved with this screen, we feel it is better to maintain the hierarchy of Districts, Municipality and Villages manually after the first upload. This first upload will be initiated by the developer on the basis of a template provided in excel. Hereafter, any change will be performed by the use of this interface. By selecting the district, the Municipality will automatically refresh and displayed in the Municipality section. The focus will always be set to the first ward. By now selecting any ward, automatically all defined villages will appear in the villages section.
One could add, edit or (logically) delete districts, Municipality and villages. For adding and editing, the user can directly enter codes in the grids. The normal rules for archiving records will apply. Deletion will only be performed by flagging the record with the validity to field (date stamp).

This screen is only available to the IMIS administrator.

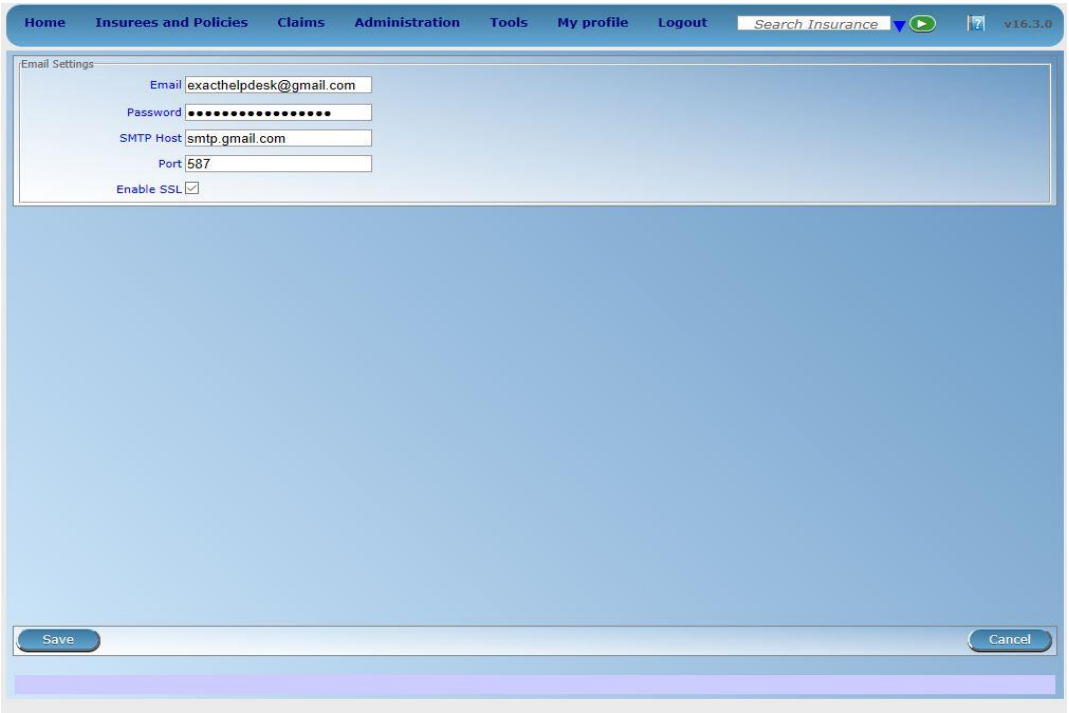
5.7.20. MoveLocation.aspx

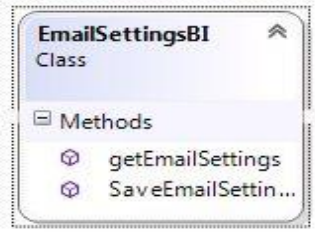


Interface object name:	*MoveLocation.aspx*
Menu path:	*Move Location*
Hyperlinks/Redirections:	*N/A*
Action Button:	*Move click on Locations page*
Use case Reference	*N/A*
Class Diagram	

Brief description:
This interface opens up when the user has clicked the move button on Locations page. The arrow button in between will be used move from one location to another location. User must select the source and the destination Location.
Village can be moved from one Municipality to another Municipality, Musicality can be moved from one district to another district and district can be moved from one region to another region.

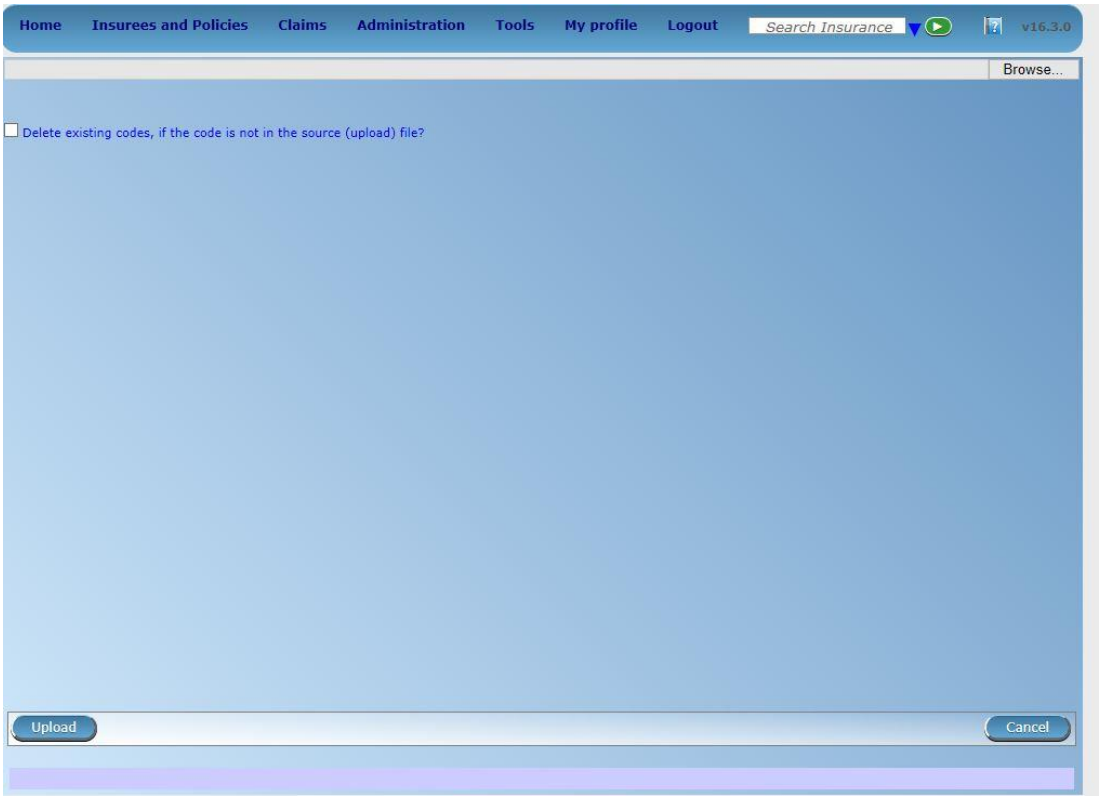
5.7.21. EmailSettings.aspx

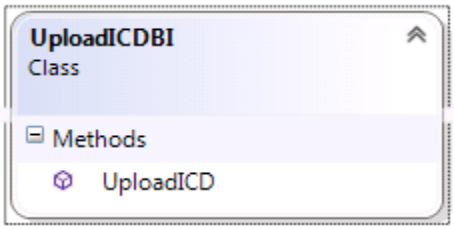


Interface object name:	*EmailSettings.aspx*
Menu path:	*Email Setting*
Hyperlinks/Redirections:	*N/A*
Action Button:	*N/A*
Use case Reference	*N/A*
Class Diagram	

Brief description:
This interface opens up when the user has clicked the menu option 'EmailSetting' in the 'Administration' menu. The save button will change default email setup this include email address (will be sender email), Password this is the email password.

5.7.22. UploadICD.aspx

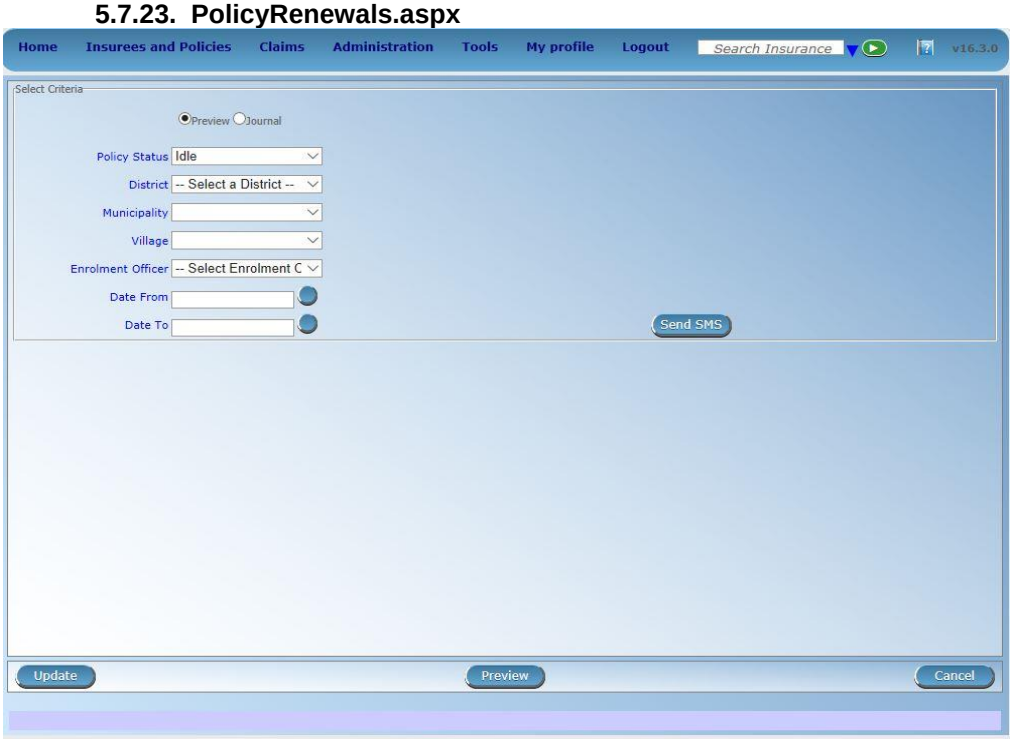


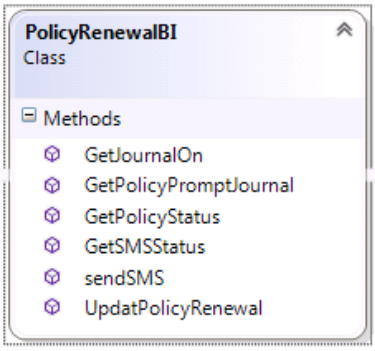
Interface object name:	*UploadICD.aspx*
Menu path:	*Upload Main DG List*
Hyperlinks/Redirections:	
Action Button:	
Use case Reference	*6.2.24 (Uploading code lists)*
Class Diagram	

Brief description:
This interface opens up when the user has clicked the menu option 'Upload Main Dg List in the 'Tools' menu. The first upload will be initiated by the developer on the basis of an Excel file layout to be provided.
After the initial upload, IMIS can be uploaded with a new (updated) list via this interface. The operator will have to select the location of the ICD file via the button Select file. By clicking the Upload button, the system will first verify if the file layout is correct. In case there is an issue with the actual Dg file, an operator message will be displayed. In case the file is valid to be uploaded, the file will be 'consumed' by IMIS and new ICD codes will amended.
The upload will work in such a way that new codes will be added to the system, changes in descriptions of codes will be updated with an audit record automatically generated, or deleted if the code is in the current list but not included in the list to be uploaded. Deletions are performed by setting the validity to field.
The layout of the excel file will be provided by the developer and will have an additional column for codes that would still exist but would need an update in the description.

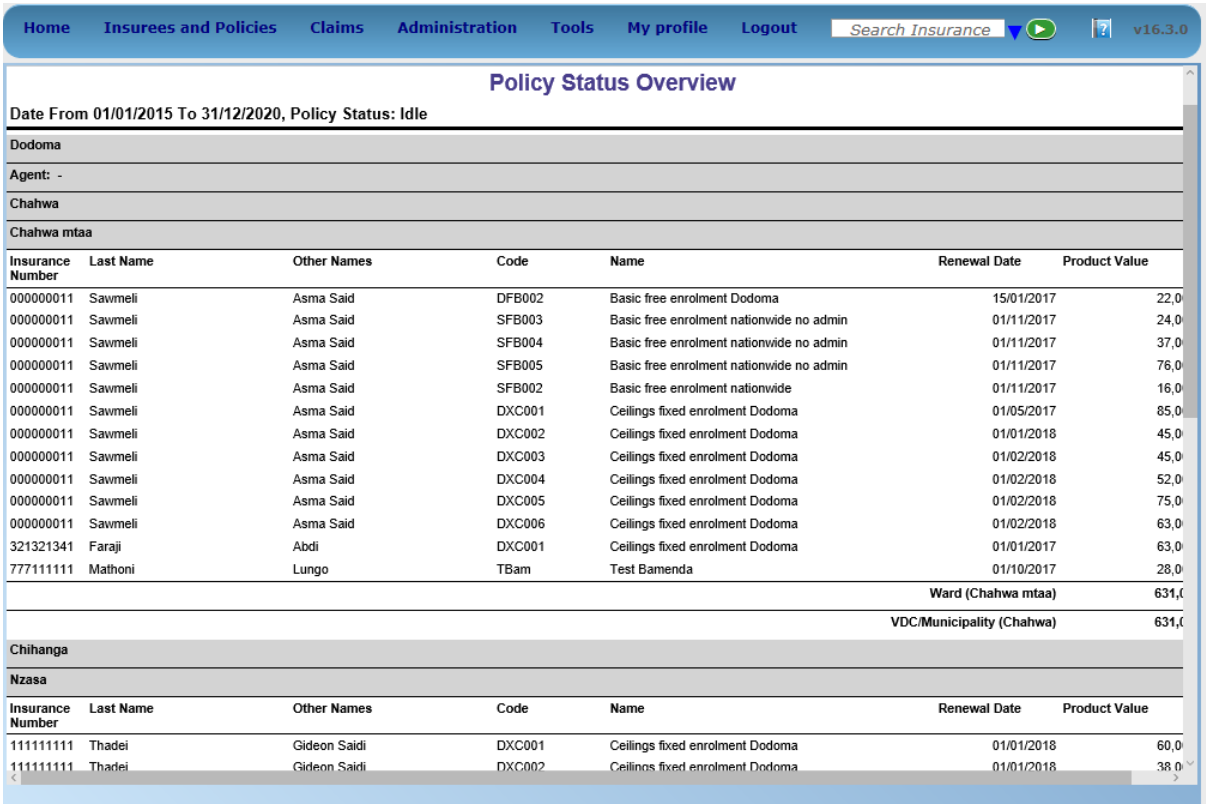
This feature is only available to the IMIS administrator

5.7.23. PolicyRenewals.aspx



Interface object name:	*PolicyRenewals.aspx*
Menu path:	*Policy Renewals*
Hyperlinks/Redirections:	*Automatic renewal prompts (SMS via web-service)*
Action Button:	
Use case Reference	*5.2.17 (prompting for policy renewal)*
Class Diagram	

Brief description:
This interface opens up when the user has clicked the menu option 'Policy Renewals' in the Tools menu. The filter options allow the operator to filter the district, village, enrollment officer and a specific period for the policy renewal report or renewal 'journal'. The Policy renewal report (via preview button) will contain all policies that will expire between the 'date from' and 'date to' for all officers or a specific officer. The layout for this report is shown below and is grouped by officer.



The Journal button will provide a report on all renewal prompts processed by the system. The report will be based on information kept in the tables: tblPolicyRenewals and tblPolicyRenewalsDetails

These tables are populated with data from a background running service that on daily basis will insert policies that expire within @Prompt Days (variable set at in web.config). The service will generate an entry for each expired policy and will also maintain information on photographs that need to be renewed considering the rules.

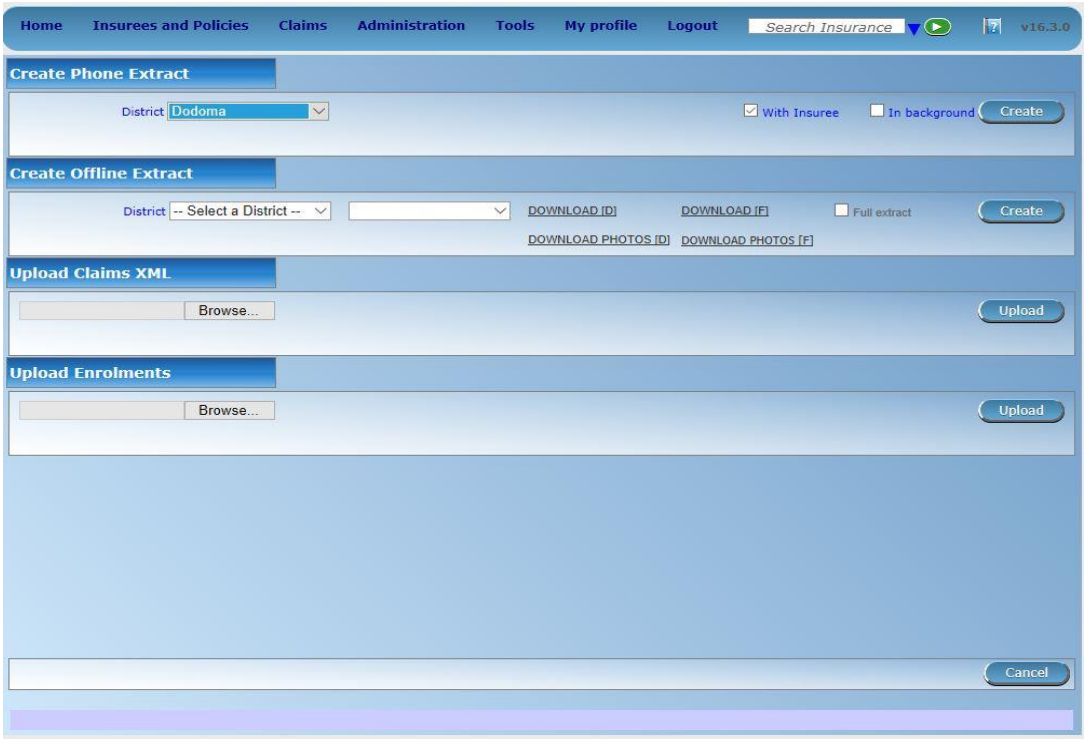
Another service will sent SMS messages to the phones of enrollment officers as described in use case 5.2.17. An extra status for SMS (not in current design) will be added to the design of the report selection screen to be able to filter out:

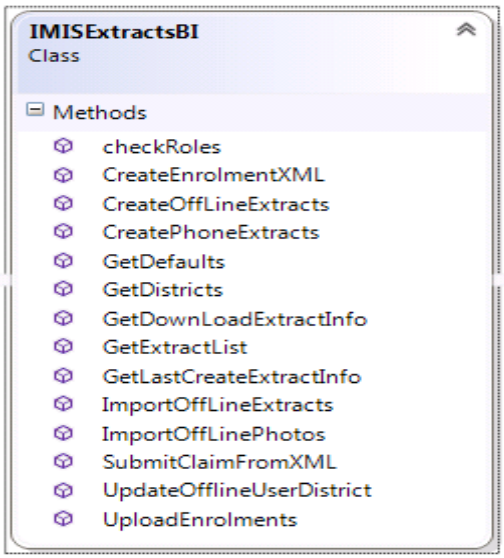
- Renewals not sent to enrollment officers phone (no phone number in registration record of the officer)
- Renewals unsuccessfully sent (problem with phone number?)
- Renewals successfully sent

5.7.24. IMIS Extracts

This page opens in two different modes depending on the type of IMIS installation: IMIS Central (live server) or IMIS Offline (installed on local network in a health facility or local network in IMIS).

IMIS Extracts (ONLINE MODE)



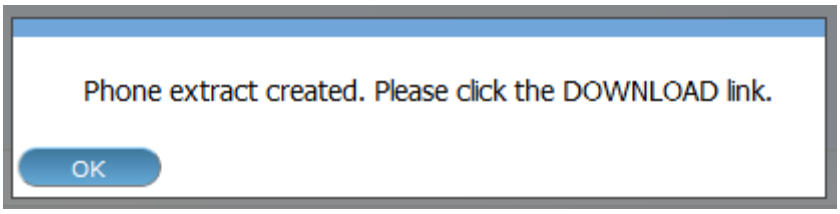
Interface object name:	*IMISExtracts.aspx*
Menu path:	*IMIS extracts(Tools menu)*
Hyperlinks/Redirections:	
Action Button:	
Use case Reference	*4.2.2 (Downloading data to an off-line client)*
Class Diagram	

A-Phone Extract panel

The Phone extract panel is used for the generation of so called SQLite database files for the mobile phone applications. Each district will have its own Phone extract file that needs to be distributed to the mobile phones within the district. To generate a phone extract file, the operator has to select a district from the list of available districts. In case the user is having access to its own district only, the district will be automatically selected and shown on the display.

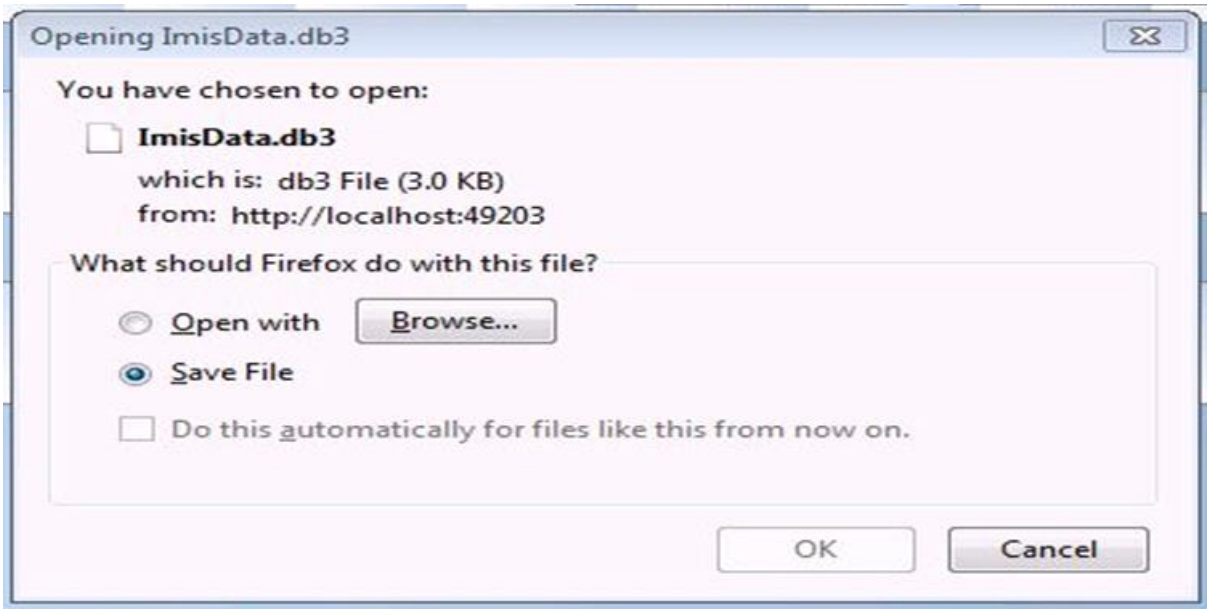
By clicking the 'create' button in panel A, a phone extract will be created. This process might take a while. As long as the hour glass (as a cursor) is shown, if you check in-background the service will download file in background while user continue working, after finish service will give the user message to download the package. The file size depends on the amount of photographs included in the extract. The file size could range into hundreds of MBs.

After the file has been created successfully the following message appears as shown below.



The extract will be downloaded to your local computer by clicking the 'download' link that will appear after the creation of the extract, as shown below.

The extract file is called 'IMISDATA.DB3' and needs first to be copied (downloaded) to the local machine. After clicking the Download button, the operator is able to select the destination folder (locally) for the file to download as shown below.



The extract is now ready to be transferred/copied to the mobile phones. This process is performed manually by connecting the mobile phone to the computer with the provided USB cable. The user needs to copy, manually, the file from the local machine into the 'IMIS' Folder on the mobile phone.

B-Offline Extract panel

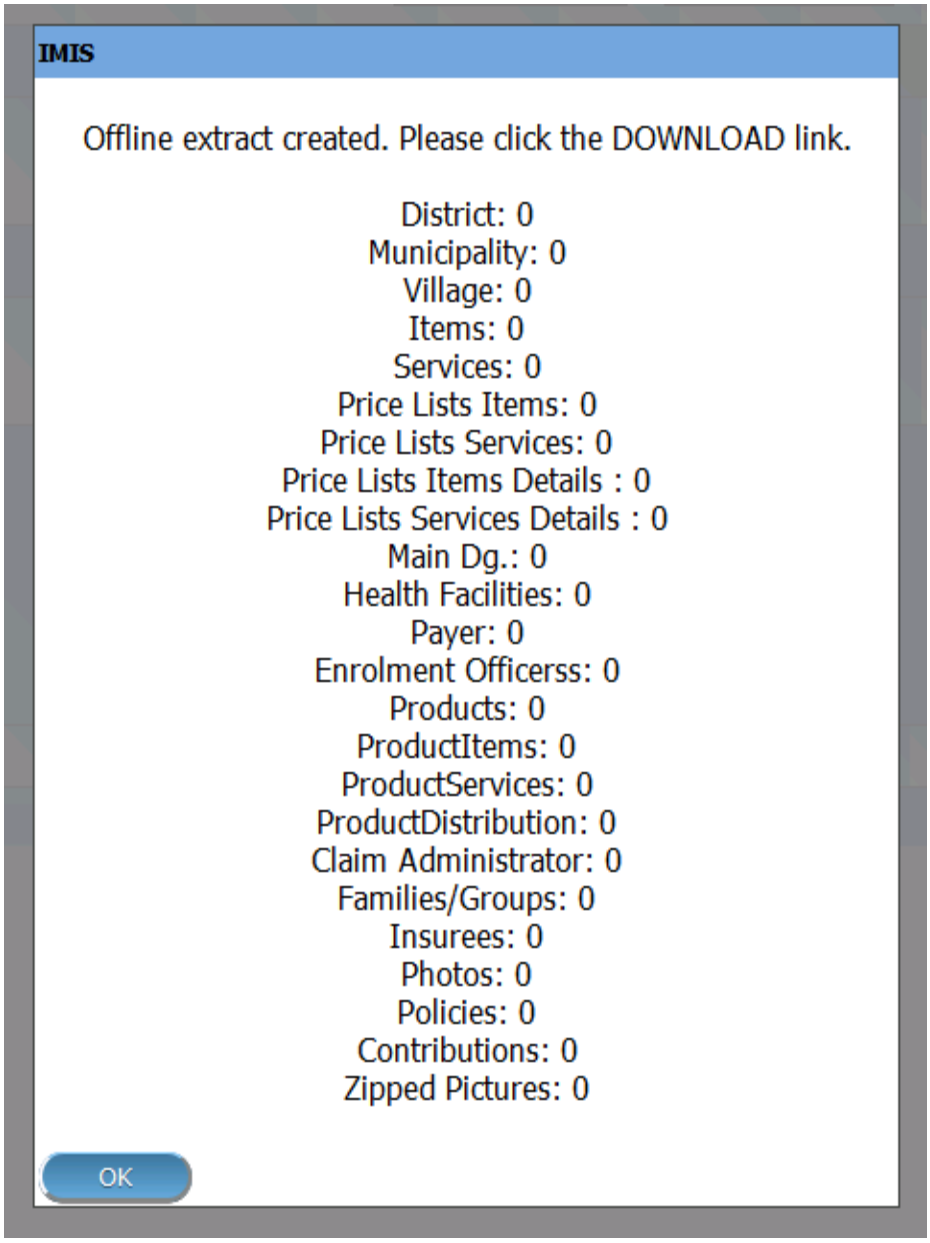
The Offline extract panel is used to generate the IMIS 'Offline' extract files for the health facilities that run IMIS offline. When an operator belongs to one specific district, the district box is already selected with the district of the user. To create a new extract, the operator needs to click the 'Create' button (in panel B).
Two types of extracts could be generated:

- Differential extract
  Differential extracts will only contain the differences in data compared with the previous extract. The first differential extract (sequence 000001) will contain all data as it will be the first extract. Thereafter, this type of extract, will only contain any differences after the previous extract. This will result in smaller files to sent to the Health Facilities in off-line mode. When we click the create button, the differential extract is ALWAYS generated and will be assigned the next sequence number. A separate Photo extract will be created containing only photographs linked to changes compared with the previous extract.

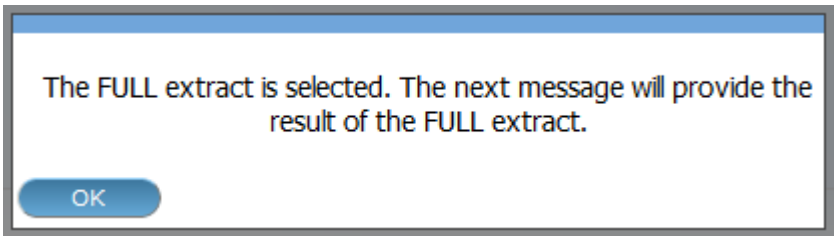
- Full extract ('Download F')
  The Full extract will always contain ALL information in the database. These extracts are only generated in case the 'Full Extract' checkbox is selected as shown below.

  By clicking the Create button, in case of 'Full extract' selected, two extracts will be generated, one differential extract and one FULL extract. Both extracts will have the same sequence number. This implies that FULL extracts are not always needed/generated. A separate Photo extract will be created containing ALL photographs.

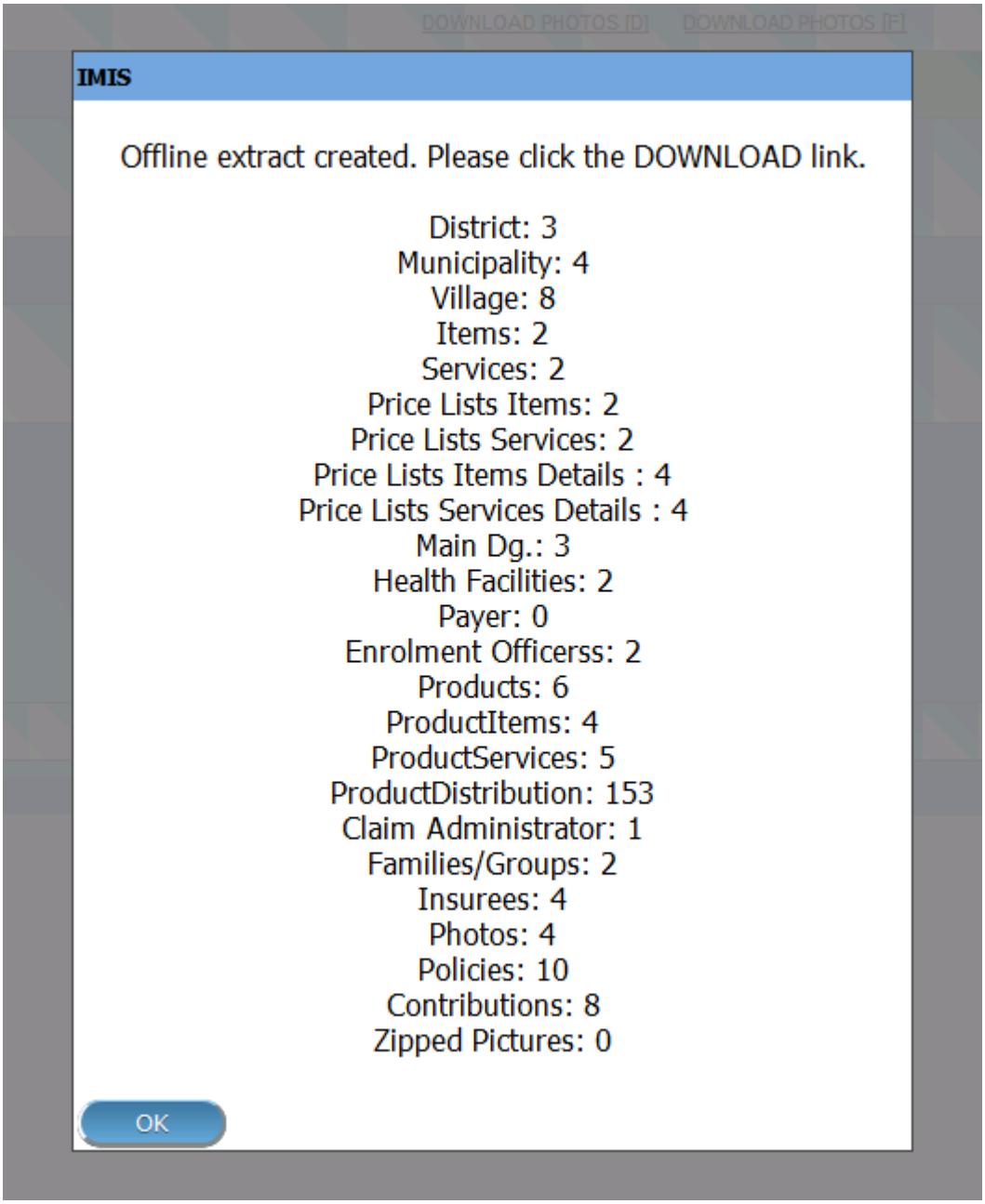
After clicking the 'Create' button, the system will create the extract file and will on completion display the following message:



The message is only shown to provide some details on how much information is exported to the extract file. Depending on the 'Full extract' option, we will be redirected to the extract page and will see the newly generated extract sequence in the list OR will get a new message as shown below:



After clicking OK the statistics of the FULL extract will be shown:



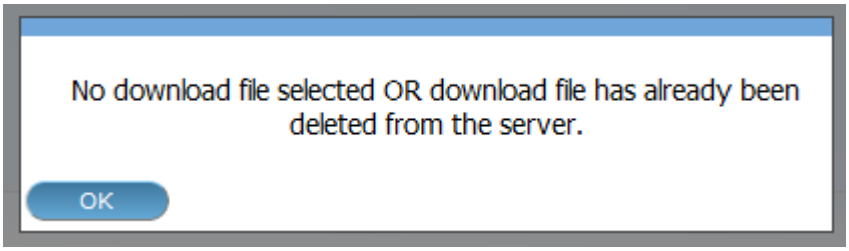
We are now ready to download the extract to our computer.
The combo box next to the district selector contains information on all generated extracts with the sequence number and date. (E.g. Sequence 000007 – Date 06-09-2012). If the extract selector does not show any entries (blank) it means that no previous extracts were created. At least one FULL extract needs to be generated. This is needed to initialise a new offline IMIS installation.
To download the actual extracts, the operator needs to select the desired extract sequence from the list of available extracts.
Four different types of extracts could be downloaded by clicking one of the following buttons:

- 'Download D' (Differential extract)
    - Will download the selected differential extract with the following filename *Filename: OE_D_<DistrictID>_<Sequence>.RAR (e.g. OE_D_1_8.RAR)*
- 'Download F' (Full extract)
    - Will download the latest FULL extract with the following filename *Filename: OE_F_<DistrictID>_<Sequence>.RAR (e.g. OE_F_1_8.RAR)*

- 'Download Photos D' (Differential Photo extract)
    - Will download the selected differential photo extract with filename:
      Filename: OE_D_<DistrictID>_<Sequence>.RAR (e.g. OE_D_1_8_Photos.RAR)
- 'Download Photos F' (Full Photo extract)
    - Will download the latest FULL photo extract with the following filename
      Filename: OE_D_<DistrictID>_<Sequence>.RAR (e.g. OE_F_1_8_Photos.RAR)

After clicking the desired extract download button, the file download dialog box appears to select the destination folder for the extract file as shown below:

In case the extract file is not available (anymore) on the server, the following dialog box might appear:



The reason for this box to appear could be that the file to be downloaded has been removed from the server or that you have attempted the download a 'FULL' extract but no 'FULL' extract was generated (only the differential extracts exist). It is also possible that you have attempted to download a Photograph extract but no photos were added since the last extract.

C-Import Extract panel

This panel will be disabled in the IMIS Online mode. (Only available for IMIS Offline)

D-Import Photos panel

This panel will be disabled in the IMIS Online mode. (Only available for IMIS Offline)

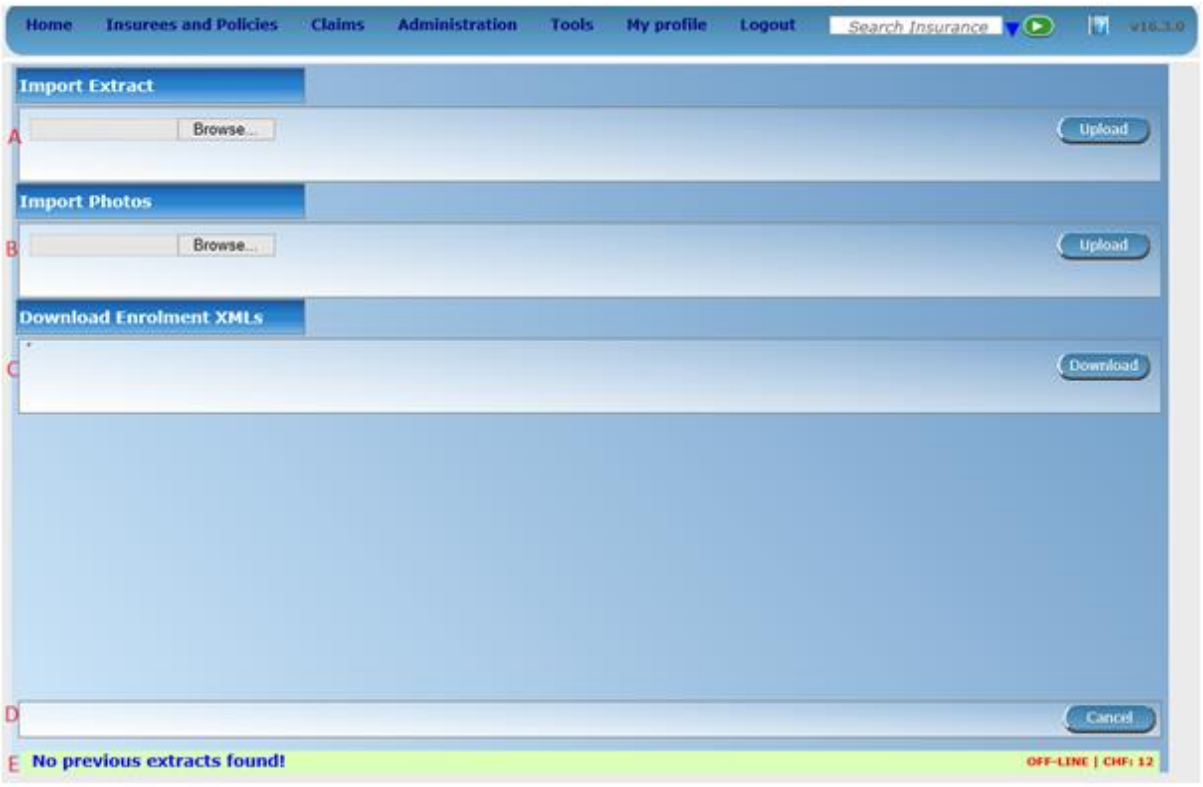
E- Button panel

The 'Cancel' button brings the operator back to the main page of IMIS.

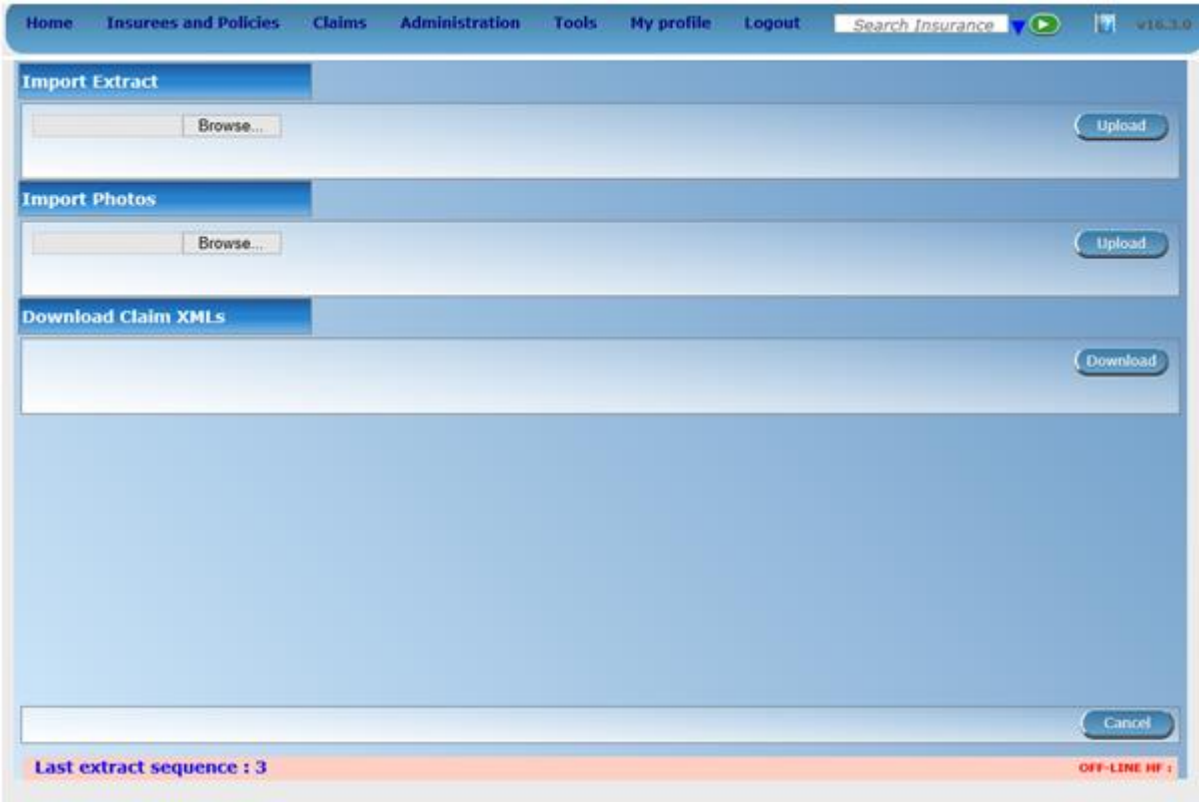
F- Information panel

The information panel is used to display messages back to the user. Messages will occur once an action has completed or if there was an error at any time during the process of these actions.

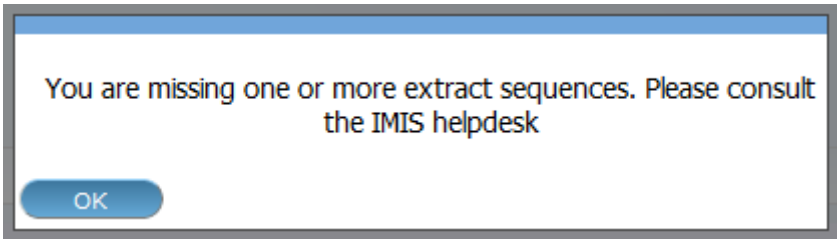
IMIS Extracts (OFFLINE MODE) HEALTH FACILITY



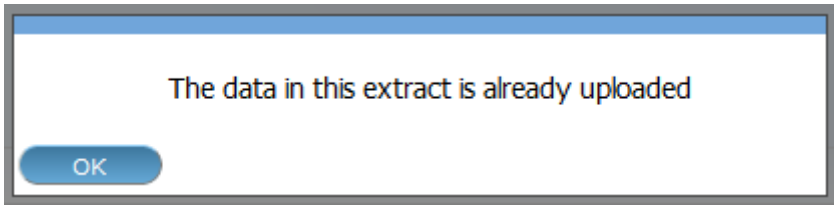
On clicking the Choose file button, the file selector dialog appears as shown below:



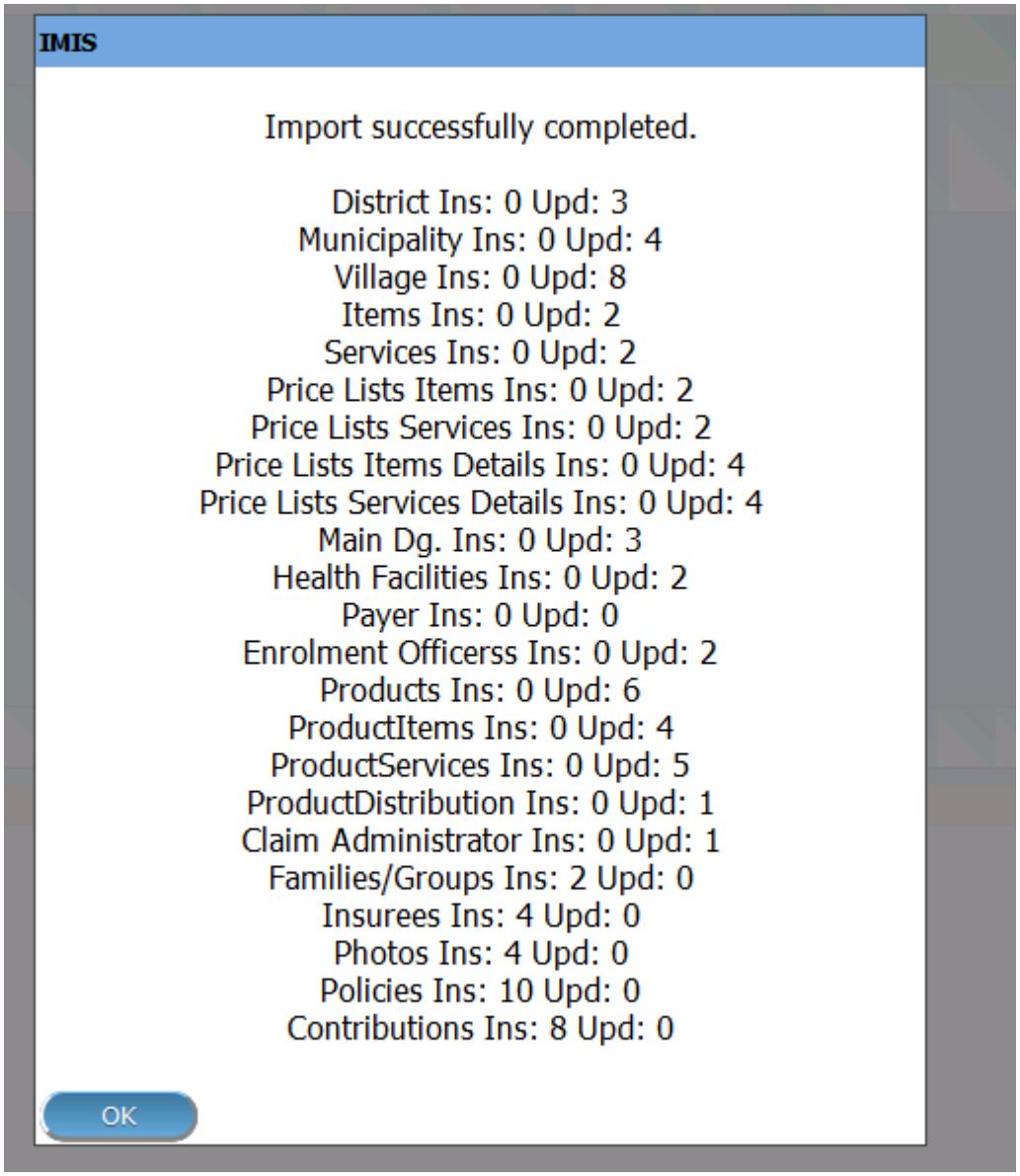
With the import/upload of an extract it is important to understand that each extract has its sequence number. This sequence number is found in the filename of the extract. We would in case of differential imports/uploads have to follow the sequence. In the example screen above, it shows in the status bar, that the last import was number 6. Therefore we should select in this case the differential extract number 7 as highlighted in the file selection dialog.
Alternatively the operator could select any 'FULL' extract with a sequence number higher than 6. In case a wrong extract is selected, warning messages will appear as shown below:



OR



In case you are missing extract sequences, additional extracts are needed to be uploaded before the extract selected. The extract selected, in this case, does not directly follow the last sequence as indicated in the status bar of the screen. The additional extracts are to be provided by IMIS scheme administrator.
In case the extract file selected is valid, the system will import the data. New data will be added and existing data might be modified. After a successful import of an extract (Differential and FULL), a form is displayed with the statistics of the import as shown below:

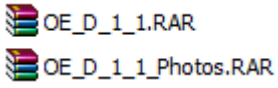


The above statistics are provided to give some quick overview of how many records were inserted or updated during the import process. In case we would for example update the phone number of an enrolment officer, it would result in one update and one insert as we always keep historical records. The photos inserts and updates are related to information on the photos, but are not the actual photographs. The actual Photographs (*.jpg) are uploaded separately.

B-Import Photos

The import of photos is optional and will have no further checking on sequence numbers. Scheme Administator should provide (if available) with each extract the photo extract as well.

E.g. (for Differential extract)

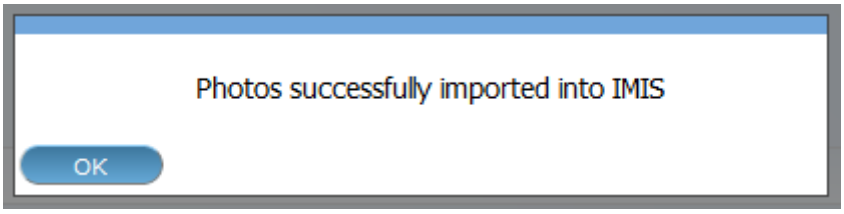


OR (for FULL extract)

OE_F_1_8.RAR
OE_F_1_8_Photos.RAR

The photo extract will contain all photographs associated with the actual extract in a zipped format. The Upload procedure will simply unzip the extract and copy the image files to the photo folder of IMIS.

After successful upload of the photographs the following message appears:



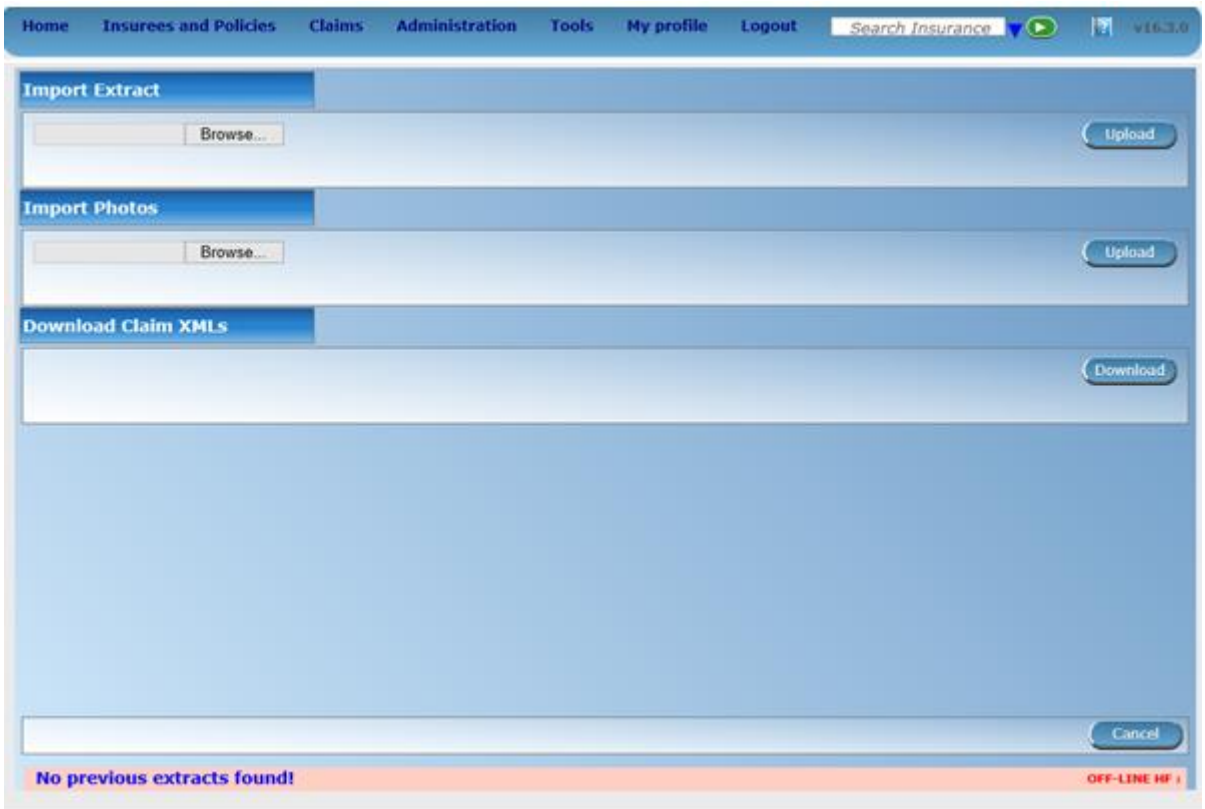
D- Button panel

The 'Cancel' button brings the operator back to the main page of IMIS.

E- Information panel

The information panel is used to display messages back to the user. Messages will occur once an action has completed or if there was an error at any time during the process of these actions. If the user opens the IMIS extracts page (in Offline mode only), the status bar will show the last sequence number uploaded.

IMIS Extracts (OFFLINE MODE)



- ***Import Extract***

Used to upload extract obtained from online IMIS, refer steps as mentioned on import extract in offline health facility.

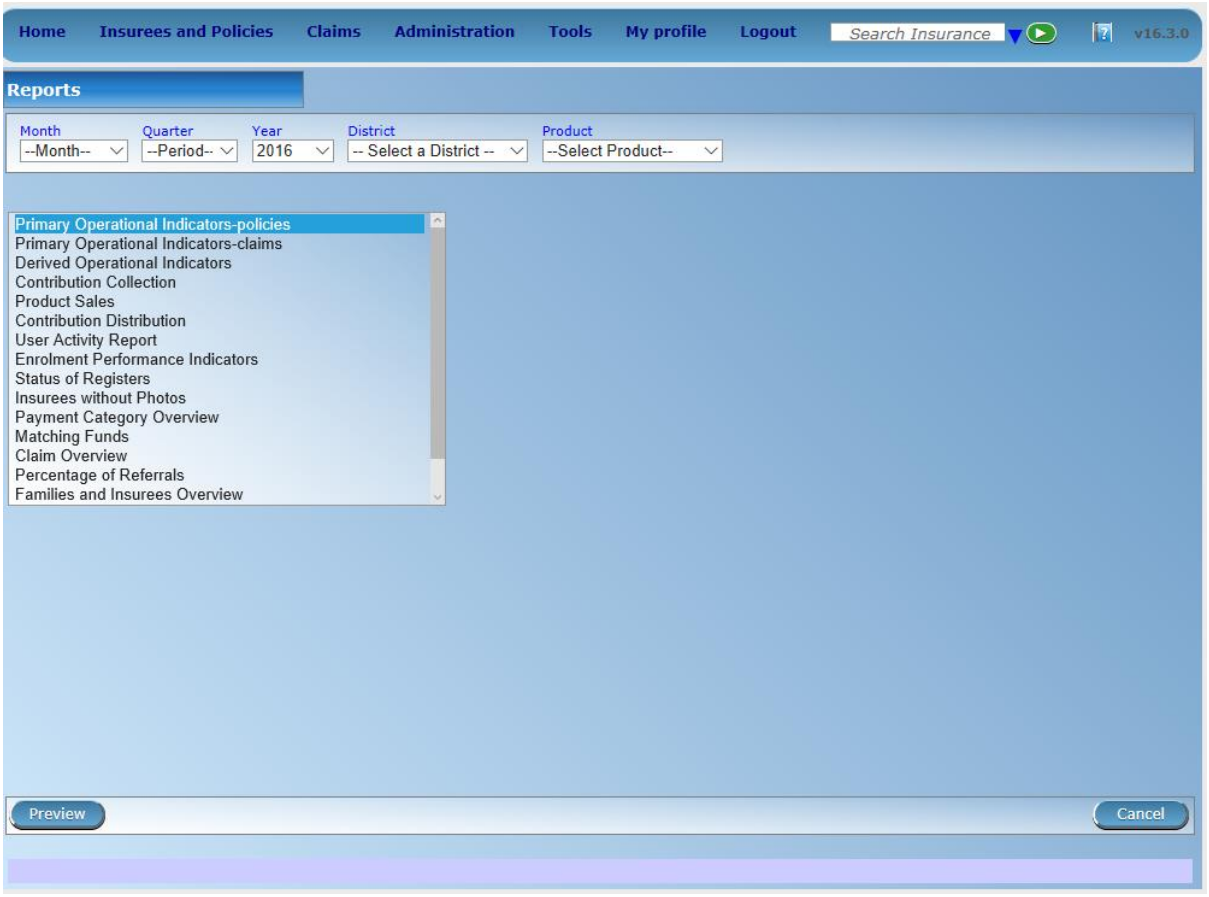
- ***Import Photos***

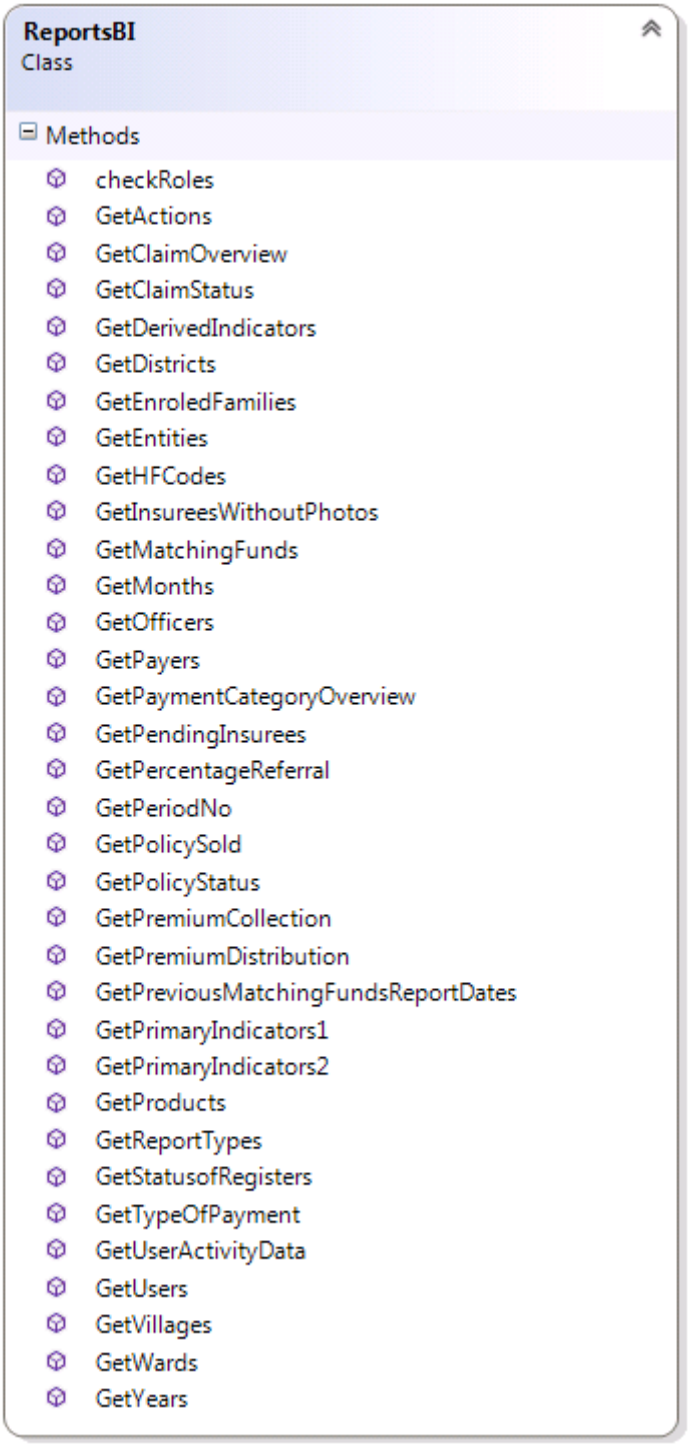
Used to upload photos obtained from online IMIS, refer steps as mentioned on import photos in offline health facility

- ***Download Enrolment XMLs***

Used to download families, insurees, policies and Contributions created in the offline IMIS HF prior to be sent to online IMIS.

5.7.25. Reports.aspx



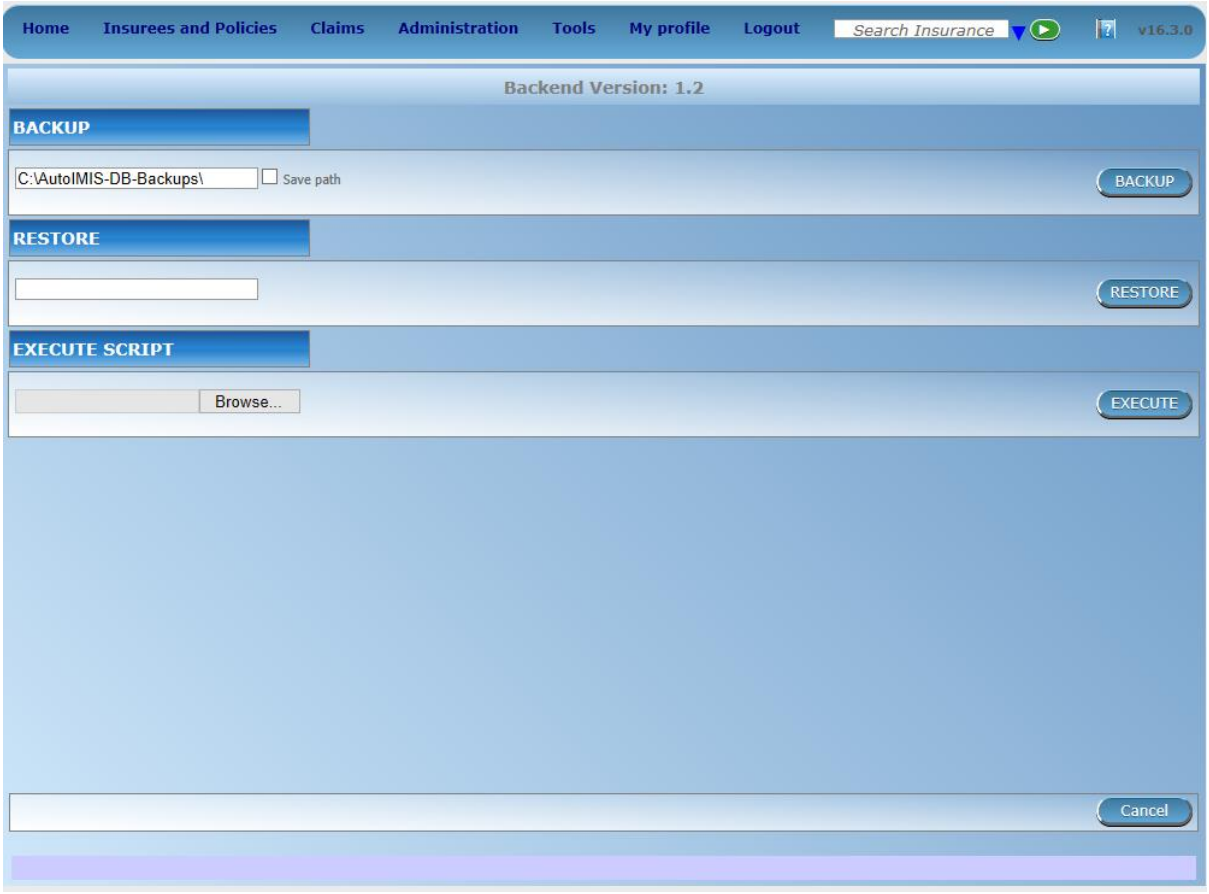
Interface object name:	*Reports.aspx*
Menu path:	*Reports (under Tools menu)*
Hyperlinks/Redirections:	
Action Button:	
Use case Reference	*(creation of all reports)*
Class Diagram	

Brief description:

This interface opens up when the user has clicked the 'Reports' option in the reports menu under the Tools top menu. This function is used to generate statistical reports from IMIS. The report selector has several filters as shown in the screenshot. By selecting any of the report, and clicking the preview button, the desired report will be displayed applying the desired filter setting.

The report will be generated using SQL Server reporting services (SSRS). The report will be exportable to MS Excel and PDF formats.

5.7.26. Utilities.aspx



Interface object name:	*Utilities.aspx*
Menu path:	*Utilities under Tools menu*
Hyperlinks/Redirections:	
Action Button:	
Use case Reference	*N/A*
Class Diagram	

Brief description:
The interface is only available for the Offline IMIS installation in health facilities.

This interface is to be used for the following purposes:

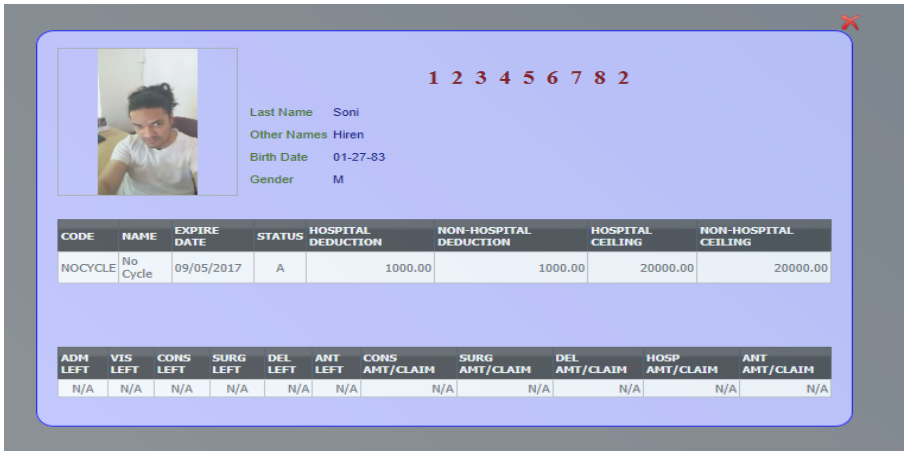
- Creation of database backup
- Restoring database
- Executing a database script

By clicking the 'backup' button, the full database backup will be created in the backup folder selected. Optionally one could change the backup folder name and save it as a default.

The restore feature will allow a full database to be restored using a previously created SQL Server database backup file. The operator will have to fully type the path and filename of the backup file. On clicking the restore button, the database will be overwritten with the contents of the database in the backup file. Obviously this action should be used with caution and will only be available for the IMIS HF Administrator (Role 512).

The script execution is only needed in case IMIS will have some structural changes on database level or some specific data manipulation is required. The operator should select the file provided and click the 'Execute' button. The system will now automatically apply the database changes. The files to run will be provided by Exact Software, the developers.

5.7.27. IMIS.MASTER (used for Menu and Quick Inquiry)

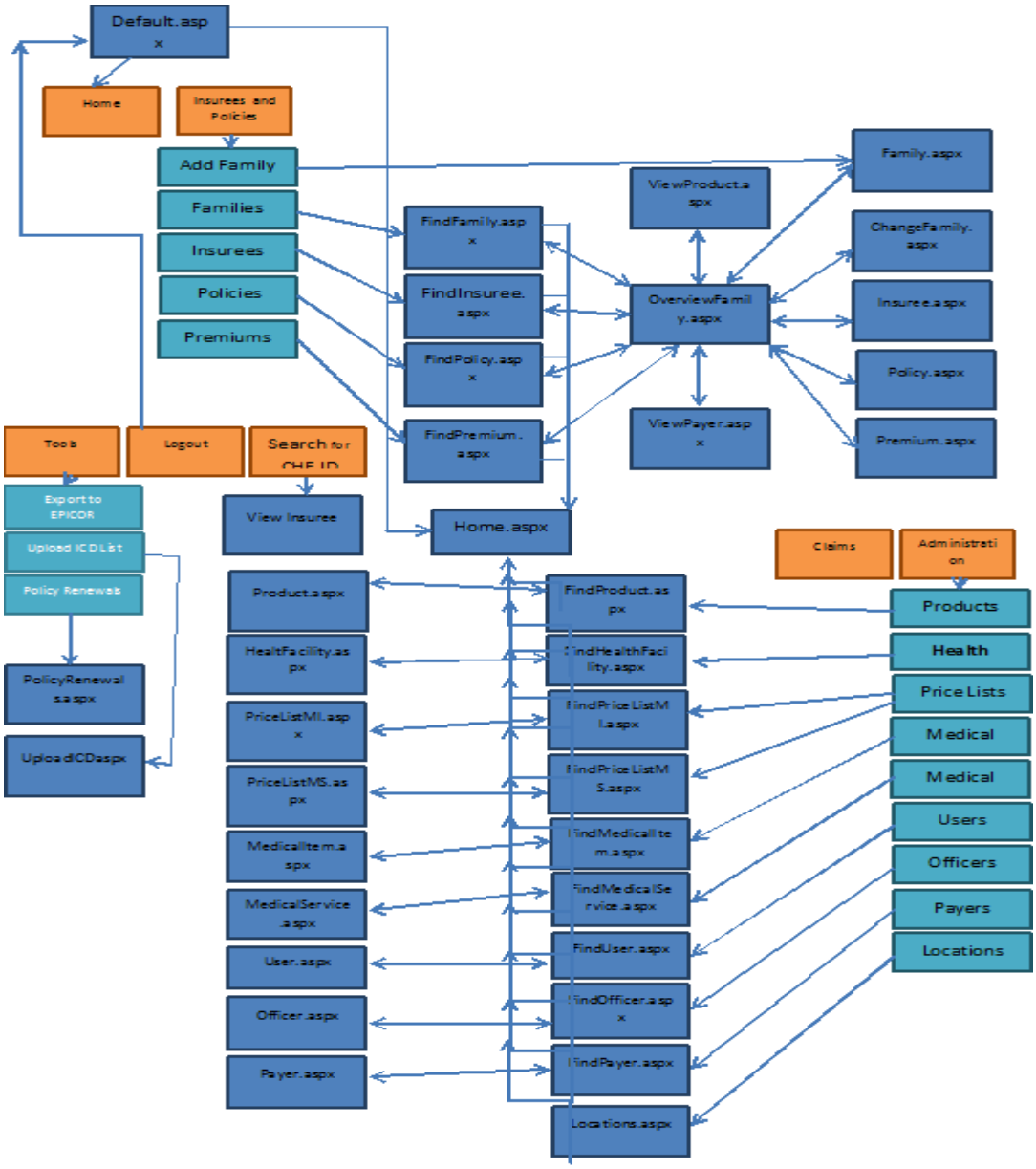


Although this object is not an aspx page as such it is included in this chapter as it is part of the presentation Tier and is included in the use case 5.2.20 scenario in the specification for phase 3.

The above window (policy overview insure) is activated via the Quick Insuree search facility in the top menu bar as shown below:

This page is now developed for the On-line and off - line system. A similar page is suggested to be developed for the quick enquiry screen for mobile phones. The designs for the mobile phone are currently under review by experts from the Swiss tropical health institute.

5.8. Graphical page routing overview



6. Database Design

This chapter will cover the database design for the Online IMIS and the Off-line clients that have no connectivity.

Paragraph 4.1 will provide the design of the database 'IMIS'. This database we will find in the on-line and off-line scenario. The off-line client will have an extra database for the Claim management objects in order to separate the 'master data' that originates from the Central database from the Claim management objects belonging to the offline facility.

6.1. Tables with main properties

Each record in each table in the database has an internal ID number. This number is automatically generated via the identity seed configuration for these ID fields. The ID fields are also used to enforce the integrity constraints between the tables. The database diagrams discussed later in this chapter will provide an overview on the relationships using these ID fields in the various tables.

Each table in the database has an additional 4 fields:

- ValidityFrom
- ValidityTo
- LegacyID
- AuditUserID

These 4 fields are used for data auditing and data archiving. Any record created in IMIS will have by default todays timestamp in the ValidityFrom field. This is automatically enforced by the default constraint 'GETDATE()' in the ValidityFrom field.

In case any record is about to be changed by the application, e.g. a price of a service, the following actions will occur:

1. Create a new record containing the original data of the record (before the change) with in the ValidityTo field the value of todays timestamp and in the LegacyID field the ID value of the original record.
2. Update the current record with the changed information.

The legacyID will be used to link all 'versions' of a record together with a ValidityFrom and ValidityTo date. The original ID of the record will always remain the unique key of the record and is used to relate to other objects in the database. The ValidityTo field of the original record will only be updated in the original record in case of delete actions.

The AuditUserID will always contain the UserID of the operator that inserted the record or '0' if it was updated by a service.

Certain tables (claim management tables) will have besides the normal audit fields as well similar audit fields for a 'reviewer':

- ValidityFromReview
- ValidityToReview
- AuditUserIDReview

These fields will get values at the moment the reviewer is adding information for the first time. This way we can prevent that always we will have multiple records while only information is added by the reviewer and original data is actually not changed. At the moment a reviewer is changing information (the audit information for the

reviewer has already been registered in the record), like in the normal archiving scenario, an additional record will be created to register the change of data and the legacyID will be used for keeping the versioning of records together.

Hereunder, we will further cover all tables individually.

6.1.1. tblRegion

Column Name	Data Type	Allow Nulls
RegionId	int	☐
RegionName	nvarchar(50)	☑
ValidityFrom	datetime	☑
ValidityTo	datetime	☑
LegacyId	int	☑
AuditUserId	int	☑
RowID	timestamp	☐
RegionCode	nvarchar(8)	☑

Objects that depend on tblRegion:	TblRegion depends on:
tblRegions tblDistricts udfNumberOfCurrentInsuree udfNumberOfCurrentPolicies udfNumberOfPoliciesExpired uspCleanTables uspExportOffLineExtract1 uspImportLocations uspImportOffLineExtract1 uspPremumAllocated uvwAmountApproved uvwAmountClaimed uvwAmountRejected uvwAmountValuated uvwClaimEntered uvwClaimProcessed uvwClaimRejected uvwClaimSent uvwClaimSubmitted uvwClaimValuated	**None**

Reference to Data Model Entity: Region

Notes: The table TblPayer will only depend on TblDistricts in case a Payer belongs to 1 district.

This table will be initially uploaded from external files. This table will be initially uploaded from external files provided by Administrator

6.1.2.tblDistricts

TblDistrcits will contain all districts to be used and referenced in IMIS.
This table will initially be populated from an external file to avoid excessive data entry. Thereafter this table will be maintained via the user interfaces.

Column Name	Data Type	Allow Nulls
DistrictID	int	☐
DistrictName	nvarchar(50)	☐
Region	nvarchar(50)	☐
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐
RowID	timestamp	☑
Prefix	smallint	☑

Objects that depend on tblDistricts:	TblDistricts depends on:
tblDistricts tblBatchRun tblOfficer tblPayer tblPLItems tblPLServices tblProduct tblUsersDistricts tblWards uspExportOffLineExtract1 uspImportLocations uspImportOffLineExtract1 uspSSRSFeedbackPrompt	tblDistricts

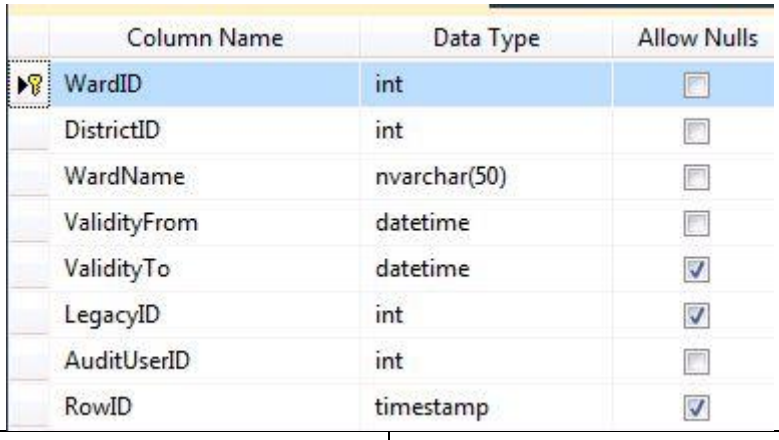
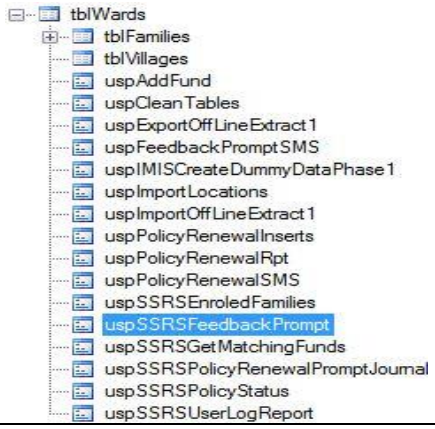
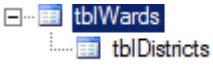
Reference to Data Model Entity: **District**

Notes: The table TblPayer will only depend on TblDistricts in case a Payer belongs to 1 district.

This table will be initially uploaded from external files. This table will be initially uploaded from external files provided by Administrator

6.1.3.TblWard (Municipality)

TblWard will contain all Municipality that fall under the villages. This table will initially be populated from an external file to avoid excessive data entry. Thereafter this table will be maintained via the user interfaces.

<table>
<tr><td colspan="2"> 
</td></tr>
<tr><td>Objects that depend on TblWard :</td><td>TblWard depends on:</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">Reference to Data Model Code List: Districts-Villages-Municipality</td></tr>
<tr><td colspan="2">Notes: This table will be initially uploaded from external files provided by Administrator</td></tr>
</table>

6.1.4.tblVillages

TblVillages will contain all villages that fall under the districts. This table will initially be populated from an external file to avoid excessive data entry. Thereafter this table will be maintained via the user interfaces.

Column Name	Data Type	Allow Nulls
VillageID	int	☐
WardID	int	☐
VillageName	nvarchar(50)	☐
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐
RowID	timestamp	☑
		☐

Objects that depend on tblVillages:	TblVillages depends on:
tblVillages tblFamilies	tblVillages tblWards

Reference to Data Model Code List: **Districts-Villages-Municipality**

Notes: This table will be initially uploaded from external files provided by Administrator

6.1.5.TblUsers

TblUser will contain all IMIS users with credentials and security role.
The security field RoleID has the following

Column Name	Data Type	Allow Nulls
UserID	int	☐
LanguageID	nvarchar(2)	☐
LastName	nvarchar(100)	☐
OtherNames	nvarchar(100)	☐
Phone	nvarchar(50)	☑
LoginName	nvarchar(25)	☐
RoleID	int	☐
HFID	int	☑
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐
password	varbinary(256)	☑
DummyPwd	nvarchar(25)	☑
EmailId	nvarchar(200)	☑

Objects that depend on tblUsers:	TblUsers depends on:
tblUsers tblClaim tblLogins tblUsersDistricts	tblUsers tblLanguages

Reference to Data Model Entity: **User**

Notes: RoleID will contain a bitmask for each of the User Roles:

- 1 = Enrolment Assistant

- 2 = Manager
- 4 = Accountant
- 8 = Clerk
- 16 = Medical Officer
- 32 = Scheme Administrator
- 64 = IMIS Scheme Administrator
- 128 = Receptionist
- 256 = Claim Administrator
- 512 = Claim Contributor
- 524288= HF Administrator
- 1048576= Scheme Offline Administrator

E.g. 48 means Medical Officer + Scheme Administrator

6.1.6. tblUsersDistricts

TblUsersDistricts will contain all districts that a user could select/use in IMIS. Within these districts the security role of the user is defined at user level (tblUser).

Column Name	Data Type	Allow Nulls
UserDistrictID	int	☐
UserID	int	☐
DistrictID	int	☐
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐
		☐

Objects that depend on tblUsersDistricts:	tblUsersDistricts depends on:
tblUsersDistricts	tblUsersDistricts tblDistricts tblUsers

Reference to Data Model Entity: **N/A**

Notes: All mappings of users towards Districts is hosted in this table to keep a uniform approach.

6.1.7.tblOfficer

tblOfficer contains all enrolment officers.

Column Name	Data Type	Allow Nulls
OfficerID	int	☐
Code	nvarchar(8)	☐
LastName	nvarchar(100)	☐
OtherNames	nvarchar(100)	☐
DOB	date	☑
Phone	nvarchar(50)	☑
DistrictID	int	☐
OfficerIDSubst	int	☑
WorksTo	smalldatetime	☑
VEOCode	nvarchar(8)	☑
VEOLastName	nvarchar(100)	☑
VEOOtherNames	nvarchar(100)	☑
VEODOB	date	☑
VEOPhone	nvarchar(25)	☑
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐
RowID	timestamp	☑
EmailId	nvarchar(200)	☑
PhoneCommunication	bit	☑

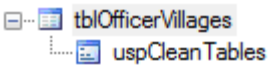
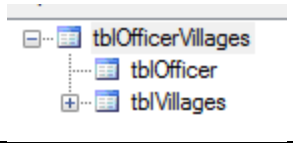
Objects that depend on tblOfficer:	tblOfficer depends on:
tblOfficer tblClaim tblPolicy uspImportOffLineExtract3 uspInsertFeedback uspSSRSFeedbackPrompt uspSSRSStatusRegister	tblOfficer tblDistricts

Reference to Data Model Entity: **Enrolment officer**

Notes: Internal self referencing field OfficerSubst

6.1.8. tblOfficerVillages

Column Name	Data Type	Allow Nulls
OfficerVillageId	int	☐
OfficerId	int	☑
VillageId	int	☑
ValidityFrom	datetime	☑
ValidityTo	datetime	☑
LegacyId	int	☑
AuditUserId	int	☑
RowId	timestamp	☐

Objects that depend on tblOfficerVillages:	tblOfficerVillages depends on:
	

Reference to Data Model Entity: **N/A**

6.1.9.tblICDCodes

This table will contain standard international codes for diseases. These codes are used at the entry of claims.

Column Name	Data Type	Allow Nu
ICDID	int	☐
ICDCode	nvarchar(6)	☐
ICDName	nvarchar(255)	☐
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐
RowID	timestamp	☑

Objects that depend on tblICDCodes:	tblICDCodes depends on:
tblICDCodes tblClaim	tblICDCodes

Reference to Data Model Code List: **Diseases**

Notes: This table will be initially uploaded from external files provided by Administator

6.1.10. tblItems

Column Name	Data Type	Allow Nulls
ItemID	int	☐
ItemCode	nvarchar(6)	☐
ItemName	nvarchar(100)	☐
ItemType	char(1)	☐
ItemPackage	nvarchar(255)	☑
ItemPrice	decimal(18, 2)	☐
ItemCareType	char(1)	☐
ItemFrequency	smallint	☑
ItemPatCat	tinyint	☐
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐
RowID	timestamp	☑

Objects that depend on tblItems:	tblItems depends on:
tblItems tblClaimItems tblPLItemsDetail tblProductItems	tblItems

Reference to Data Model Entity: **Medical Item**

Notes:

ItemType could contain the following options:

- 'D' = Drug
- 'P' = Prostheses

ItemCareType will contain a bitmask for each of the following options:

- I = In patient
- O = Out-patient
- B = Both

ItemPatCat will contain a bitmask for each of the following options:

- 1 = Man
- 2 = Woman
- 4 = Adult
- 8 = Child

6.1.11. tblPLItems

Column Name	Data Type	Allow Nulls
PLItemID	int	☐
PLItemName	nvarchar(100)	☐
DatePL	date	☐
DistrictID	int	☑
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐
RowID	timestamp	☑

Objects that depend on tblPLItems:	tblPLItems depends on:
tblPLItems tblHF tblPLItemsDetail	tblPLItems tblDistricts

Reference to Data Model Entity: **Pricelist of Medical Items**

Notes: DistrictID can contain null. This means that in that case all districts could use this pricelist definition

6.1.12. tblPLItemsDetail

Column Name	Data Type	Allow Nulls
PLItemDetailID	int	☐
PLItemID	int	☐
ItemID	int	☐
PriceOverule	decimal(18, 2)	☑
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐
RowID	timestamp	☑

Objects that depend on tblPLItemsDetail:	tblPLItemsDetail depends on:
tblPLItemsDetail	tblPLItemsDetail tblItems tblPLItems

Reference to Data Model Entity: **Remark5**

Notes: This table maps items with pricelists (PLItemID) with the option to overrule the standard price from tblItems (ItemID)

6.1.13. tblServices

Column Name	Data Type	Allow Nulls
ServiceID	int	☐
ServCode	nvarchar(6)	☐
ServName	nvarchar(100)	☐
ServType	char(1)	☐
ServLevel	char(1)	☐
ServPrice	decimal(18, 2)	☐
ServCareType	char(1)	☐
ServFrequency	smallint	☑
ServPatCat	tinyint	☐
ValidityFrom	datetime	☑
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☑
RowID	timestamp	☑
ServCategory	char(1)	☑
		☐

Objects that depend on tblServices:	tblServices depends on:
tblServices tblClaimServices tblPLServicesDetail tblProductServices	tblServices

Reference to Data Model Entity: **Medical Item**

Notes:

ServType could contain the following options:

- 'P' = Preventative
- 'C' = Curative

ServLevel:

- 'S' = Simple service
- 'V' = Visit
- 'D' = Day of stay
- 'H' = Hospital case

ServCareType will contain a bitmask for each of the following options:

- I = In patient
- O = Out-patient
- B = Both

ServPatCat will contain a bitmask for each of the following options:

- 1 = Man
- 2 = Woman
- 4 = Adult
- 8 = Child

6.1.14. tblPLServices

Column Name	Data Type	Allow Nulls
PLServiceID	int	☐
PLServName	nvarchar(100)	☐
DatePL	date	☐
DistrictID	int	☑
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐
RowID	timestamp	☑

Objects that depend on tblPLServices:	tblPLServices depends on:
tblPLServices tblHF tblPLServicesDetail	tblPLServices tblDistricts

Reference to Data Model Entity: **Pricelist of Services**

Notes: DistrictID can contain null. This means that in that case all districts could use this pricelist definition

6.1.15. tblPLServicesDetail

Column Name	Data Type	Allow Nulls
PLServiceDetailID	int	☐
PLServiceID	int	☐
ServiceID	int	☐
PriceOverule	decimal(18, 2)	☑
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐
RowID	timestamp	☑

Objects that depend on tblPLServicesDetail:	tblPLServicesDetail depends on:
tblPLServicesDetail	tblPLServicesDetail tblPLServices tblServices

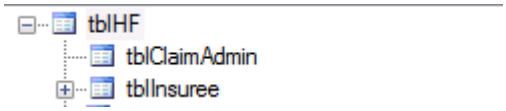
Reference to Data Model Entity: **Remark4**

Notes: This table maps items with pricelists (PLServiceID) with the option to overrule the standard price from tblServices (ServiceID)

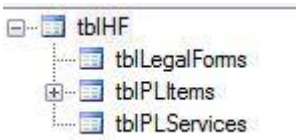
6.1.16. tblHF

Column Name	Data Type	Allow Nulls
HfID	int	☐
HFCode	nvarchar(8)	☐
HFName	nvarchar(100)	☐
LegalForm	char(1)	☐
HFLevel	char(1)	☐
HFSublevel	char(1)	☑
HFAddress	nvarchar(100)	☑
DistrictID	int	☐
Phone	nvarchar(50)	☑
Fax	nvarchar(50)	☑
eMail	nvarchar(50)	☑
HFCareType	char(1)	☐
PLServiceID	int	☑
PLItemID	int	☑
AccCode	nvarchar(25)	☑
OffLine	bit	☐
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐
RowID	timestamp	☑

Objects that depend on tblHF:



tblHF depends on:



Reference to Data Model Entity: **Health facility**

Notes: We might want to include in the design an extra field for the facility to indicate if the HF is online OR offline (e.g. isOffline bit). Also it might be useful to add an HFCode (nvarchar(4)) unique to indentify the offline database**.**

LegalForm:

- **'C' = Organization/Charity**
- **'D' = Government/District**
- **'P' = Organization/Private**

HFLevel:

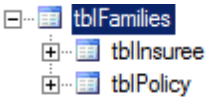
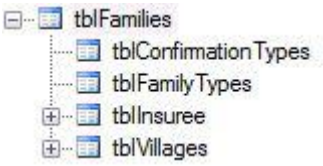
- **'S' = Sub health post**
- **'P' = Heath post**
- **'C' = Primary Health Centre**
- **'H' = Hospital**

HFCareType:

- **'I' = Inpatient**
- **'O' = Outpatient**
- **'B' = Both**

6.1.17. tblFamilies

Column Name	Data Type	Allow Nulls
FamilyID	int	☐
InsureeID	int	☐
DistrictID	int	☐
VillageID	int	☐
WardID	int	☐
Poverty	bit	☑
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐
RowID	timestamp	☑
FamilyType	nvarchar(2)	☑
FamilyAddress	nvarchar(200)	☑
isOffline	bit	☑
Ethnicity	nvarchar(1)	☑
ConfirmationNo	nvarchar(12)	☑
CurDistrict	int	☑
CurVDC	int	☑
CurWard	int	☑
ConfirmationType	nvarchar(3)	☑

Objects that depend on tblFamilies:	tblFamilies depends on:
	

Reference to Data Model Entity: **Insured family/group. Insuree field will determine the head of family/group. This field is included for simplicity of historic reporting as basically the family/group changes by change of head of family/group.**

Notes:

6.1.18. tblInsuree

Column Name	Data Type	Allow Nulls
InsureeID	int	☐
FamilyID	int	☐
CHFID	nvarchar(12)	☐
LastName	nvarchar(100)	☐
OtherNames	nvarchar(100)	☐
DOB	date	☐
Gender	char(1)	☑
Marital	char(1)	☑
IsHead	bit	☐
passport	nvarchar(25)	☑
Phone	nvarchar(50)	☑
PhotoID	int	☑
PhotoDate	date	☑
CardIssued	bit	☐
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐
RowID	timestamp	☑
Relationship	smallint	☑
Profession	smallint	☑
Education	smallint	☑
Email	nvarchar(100)	☑
isOffline	bit	☑
TypeOfId	nvarchar(1)	☑
HFID	int	☑
CurrentAddress	nvarchar(200)	☑
GeoLocation	nvarchar(50)	☑
CurDistrict	int	☑
CurVDC	int	☑
CurWard	int	☑

Objects that depend on tblInsuree:	tblInsuree depends on:
tblInsuree tblClaim tblClaimDedRem tblFamilies tblHealthStatus tblInsureePolicy tblPolicyRenewals	tblInsuree tblEducations tblFamilies tblProfessions tblRelations

Reference to Data Model Entity: **Insuree**

Notes: Biometrics field omitted from design as not yet determined in data model.

6.1.19. tblInsureePolicy

Column Name	Data Type	Allow Nulls
InsureePolicyId	int	☐
InsureeId	int	☑
PolicyId	int	☑
EnrollmentDate	date	☑
StartDate	date	☑
EffectiveDate	date	☑
ExpiryDate	date	☑
ValidityFrom	datetime	☑
ValidityTo	datetime	☑
LegacyId	int	☑
AuditUserId	int	☑
isOffline	bit	☑
RowId	timestamp	☑

Objects that depend on tblInsureePolicy:

- tblInsureePolicy
  - udfNumberOfCurrentInsuree
  - udfPremiumComposition
  - uspAddFund
  - uspAddInsureePolicy
  - uspCleanTables
  - uspExportOffLineExtract4
  - uspImportOffLineExtract4
  - uspIsValidRenewal
  - uspPolicyInquiry
  - uspProcessSingleClaimStep2
  - uspServiceItemEnquiry
  - uspSSRSPaymentCategoryOverview
  - uspSubmitSingleClaim
  - uvwNumberInsureeAcquired

tblInsureePolicy depends on:

- tblInsureePolicy
  - tblPolicy

Reference to Data Model Entity: **N/A**

Notes: This table store only active insuree

6.1.20. tblHealthStatus

Column Name	Data Type	Allow Nulls
HealthStatusID	int	☐
InsureeID	int	☐
Description	nvarchar(255)	☑
ValidityFrom	datetime	☑
ValidityTo	datetime	☑
AuditUserID	int	☑
LegacyID	int	☑

Objects that depend on tblHealthStatus:	tblHealthStatus depends on:
tblHealthStatus	tblHealthStatus tblInsuree

Reference to Data Model Entity: **N/A (will be determined later)**

Notes: This table has no further function in the current IMIS design but included for future use.

6.1.21. tblPhotos

Column Name	Data Type	Allow Nulls
PhotoID	int	☐
InsureeID	int	☑
CHFID	nvarchar(9)	☐
PhotoFolder	nvarchar(255)	☐
PhotoFileName	nvarchar(50)	☐
OfficerID	int	☐
PhotoDate	date	☐
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
AuditUserID	int	☑
RowID	timestamp	☑

Objects that depend on tblPhotos:	tblPhotos depends on:
tblPhotos tblClaim uspCleanTables uspExportOffLineExtract4 uspImportOffLineExtract4 uspPolicyInquiry uspPolicyInquiry2 uspPolicyRenewalInserts uspSSRSFeedbackPrompt uspSSRSPolicyStatus uspSSRSUserLogReport uspUploadEnrolments	tblPhotos

Reference to Data Model Entity: **N/A**

Notes: This table has references to externally stored photo image files.

6.1.22. tblPolicy

Column Name	Data Type	Allow Nulls
PolicyID	int	☐
FamilyID	int	☐
EnrollDate	date	☐
StartDate	date	☐
EffectiveDate	date	☑
ExpiryDate	date	☑
PolicyStatus	tinyint	☑
PolicyValue	decimal(18, 2)	☑
ProdID	int	☐
OfficerID	int	☐
PolicyStage	char(1)	☑
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐
RowID	timestamp	☑
isOffline	bit	☑
		☐

Objects that depend on tblPolicy:	tblPolicy depends on:
tblPolicy tblClaimDedRem tblInsureePolicy tblPolicyRenewals tblPremium	tblPolicy tblFamilies tblOfficer tblProduct

Reference to Data Model Entity: **Policy**

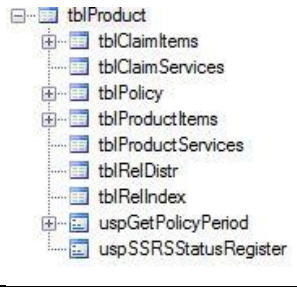
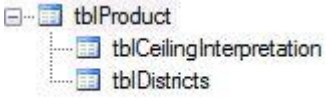
Notes:

PolicyStatus:

- 1 = Entered
- 2 = Active
- 4 = Suspended
- 8 = Expired

6.1.23. tblProduct

Column Name	Data Type	Allow Nulls
ProdID	int	☐
ProductCode	nvarchar(8)	☐
ProductName	nvarchar(100)	☐
DistrictID	int	☑
InsurancePeriod	tinyint	☐
DateFrom	smalldatetime	☐
DateTo	smalldatetime	☐
ConversionProdID	int	☑
LumpSum	decimal(18, 2)	☐
MemberCount	smallint	☐
PremiumAdult	decimal(18, 2)	☑
PremiumChild	decimal(18, 2)	☑
DedInsuree	decimal(18, 2)	☑
DedOPInsuree	decimal(18, 2)	☑
DedIPInsuree	decimal(18, 2)	☑
MaxInsuree	decimal(18, 2)	☑
MaxOPInsuree	decimal(18, 2)	☑
MaxIPInsuree	decimal(18, 2)	☑
PeriodRelPrices	char(1)	☑
PeriodRelPricesOP	char(1)	☑
PeriodRelPricesIP	char(1)	☑
AccCodePremiums	nvarchar(25)	☑
AccCodeRemuneration	nvarchar(25)	☑
DedTreatment	decimal(18, 2)	☑
DedOPTreatment	decimal(18, 2)	☑
DedIPTreatment	decimal(18, 2)	☑
MaxTreatment	decimal(18, 2)	☑
MaxOPTreatment	decimal(18, 2)	☑
MaxIPTreatment	decimal(18, 2)	☑
DedPolicy	decimal(18, 2)	☑
DedOPPolicy	decimal(18, 2)	☑
DedIPPolicy	decimal(18, 2)	☑
MaxPolicy	decimal(18, 2)	☑
MaxOPPolicy	decimal(18, 2)	☑
MaxIPPolicy	decimal(18, 2)	☑
GracePeriod	int	☐
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐
RowID	timestamp	☑
RegistrationLumpSum	decimal(18, 2)	☑
RegistrationFee	decimal(18, 2)	☑
GeneralAssemblyLumpSum	decimal(18, 2)	☑
GeneralAssemblyFee	decimal(18, 2)	☑
StartCycle1	nvarchar(5)	☑
StartCycle2	nvarchar(5)	☑
MaxNoConsultation	int	☑
MaxNoSurgery	int	☑
MaxNoDelivery	int	☑
MaxNoHospitalizaion	int	☑
MaxNoVisits	int	☑
MaxAmountConsultation	decimal(18, 2)	☑
MaxAmountSurgery	decimal(18, 2)	☑
MaxAmountDelivery	decimal(18, 2)	☑
MaxAmountHospitalization	decimal(18, 2)	☑
GracePeriodRenewal	int	☑
MaxInstallments	int	☑
WaitingPeriod	int	☑
RenewalDiscountPerc	int	☑
RenewalDiscountPeriod	int	☑
StartCycle3	nvarchar(5)	☑
StartCycle4	nvarchar(5)	☑
AdministrationPeriod	int	☑
Threshold	int	☑
MaxPolicyExtraMember	decimal(18, 2)	☑
MaxPolicyExtraMemberIP	decimal(18, 2)	☑
MaxPolicyExtraMemberOP	decimal(18, 2)	☑
MaxCeilingPolicy	decimal(18, 2)	☑
MaxCeilingPolicyIP	decimal(18, 2)	☑
MaxCeilingPolicyOP	decimal(18, 2)	☑

Objects that depend on tblProduct:	tblProduct depends on:
	
Reference to Data Model Entity: **Insurance product**	
Notes: ConversionID refers to a ProductID previously entered (self referenced). ConversionID will be Null if current entry was not referenced.	

6.1.24. tblProductItems

Column Name	Data Type	Allow Nulls
ProdItemID	int	☐
ProdID	int	☐
ItemID	int	☐
LimitationType	char(1)	☐
PriceOrigin	char(1)	☐
LimitAdult	decimal(18, 2)	☑
LimitChild	decimal(18, 2)	☑
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐
RowID	timestamp	☑
WaitingPeriodAdult	int	☑
WaitingPeriodChild	int	☑
LimitNoAdult	int	☑
LimitNoChild	int	☑
LimitationTypeR	char(1)	☑
LimitationTypeE	char(1)	☑
LimitAdultR	decimal(18, 2)	☑
LimitAdultE	decimal(18, 2)	☑
LimitChildR	decimal(18, 2)	☑
LimitChildE	decimal(18, 2)	☑
CeilingExclusionAdult	nvarchar(1)	☑
CeilingExclusionChild	nvarchar(1)	☑

Objects that depend on tblProductItems:	tblProductItems depends on:
tblProductItems	tblProductItems tblItems tblProduct

Reference to Data Model Entity: **Remark2**

Notes:

LimitationType:

- 'C' = Co-Insurance
- 'F' = Fixed Limit

PriceOrigin:

- 'P' = Price List
- 'O' = Own providers price
- 'R' = Relative Price

LimitAdult,LimitChild → will hold percentage covered by scheme Administrator for LimitationType = 'C' or limit to cover by scheme Administrator for Limitationtype = 'F'

6.1.25. tblProductServices

Column Name	Data Type	Allow Nulls
ProdServiceID	int	☐
ProdID	int	☐
ServiceID	int	☐
LimitationType	char(1)	☐
PriceOrigin	char(1)	☐
LimitAdult	decimal(18, 2)	☑
LimitChild	decimal(18, 2)	☑
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐
RowID	timestamp	☑
WaitingPeriodAdult	int	☑
WaitingPeriodChild	int	☑
LimitNoAdult	int	☑
LimitNoChild	int	☑
LimitationTypeR	char(1)	☑
LimitationTypeE	char(1)	☑
LimitAdultR	decimal(18, 2)	☑
LimitAdultE	decimal(18, 2)	☑
LimitChildR	decimal(18, 2)	☑
LimitChildE	decimal(18, 2)	☑
CeilingExclusionAdult	nvarchar(1)	☑
CeilingExclusionChild	nvarchar(1)	☑

Objects that depend on tblProductServices:	tblProductServices depends on:
tblProductServices	tblProductServices tblProduct tblServices

Reference to Data Model Entity: **Remark1**

Notes:

LimitationType:

- 'C' = Co-Insurance
- 'F' = Fixed Limit

PriceOrigin:

- 'P' = Price List
- 'O' = Own providers price
- 'R' = Relative Price

LimitAdult,LimitChild → will hold percentage covered by IMIS for LimitationType = 'C' or limit to cover by IMIS for Limitationtype = 'F'

6.1.26. tblRelDistr

Column Name	Data Type	Allow Nulls
DistrID	int	☐
DistrType	tinyint	☐
DistrCareType	char(1)	☐
ProdID	int	☐
Period	tinyint	☐
DistrPerc	decimal(18, 2)	☑
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐
RowID	timestamp	☑

Objects that depend on tblRelDistr:	tblRelDistr depends on:
tblRelDistr	tblRelDistr tblProduct

Reference to Data Model table: **Indexes of relative prices**

Notes:

DistrType could have three values

- 1 = Yearly record
- 4 = Quarterly record
- 12 = Monthly record

DistrCareType:

- 'I' = In patient
- 'O' = Out patient
- 'B' = Both

6.1.27. tblPremium

Column Name	Data Type	Allow Nulls
PremiumId	int	☐
PolicyID	int	☐
PayerID	int	☑
Amount	decimal(18, 2)	☐
Receipt	nvarchar(50)	☐
PayDate	date	☐
PayType	char(1)	☐
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐
RowID	timestamp	☑
isPhotoFee	bit	☑
isOffline	bit	☑
ReportingId	int	☑

Objects that depend on tblPremium:	tblPremium depends on:
tblPremium	tblPremium tblPayer tblPolicy

Reference to Data Model Entity: **Premium**

Notes: PayerID has a value if payment NOT made by Insuree/Family/Group otherwise NULL

6.1.28. tblPayer

Column Name	Data Type	Allow Nulls
PayerID	int	☐
PayerType	char(1)	☐
PayerName	nvarchar(100)	☐
PayerAddress	nvarchar(100)	☑
DistrictID	int	☑
Phone	nvarchar(50)	☑
Fax	nvarchar(50)	☑
eMail	nvarchar(50)	☑
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐
RowID	timestamp	☑

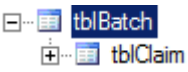
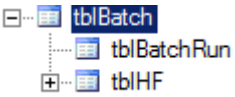
Objects that depend on tblPayer:	tblPayer depends on:
tblPayer tblPremium	tblPayer tblDistricts

Reference to Data Model Entity: **Payer**

Notes: DistrictID will be null if Payer is available for more than one district.

6.1.29. tblBatch

Column Name	Data Type	Allow Nulls
RunID	int	☐
DistrictID	int	☐
RunDate	datetime	☐
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐
RunYear	int	☐
RunMonth	tinyint	☐

Objects that depend on tblBatch:	TblBatch depends on:
	

Reference to Data Model Entity: **Batch of claims**

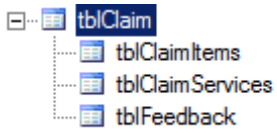
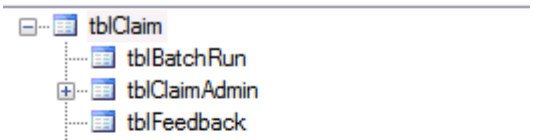
Notes:

Field BatchStatus:

- 1 = Rejected
- 2 = Opened
- 4 = Checked
- 8 = Processed
- 16 = Valuated

6.1.30. tblClaim

ClaimID	int	☐
InsureeID	int	☐
ClaimCode	nvarchar(8)	☐
DateFrom	smalldatetime	☐
DateTo	smalldatetime	☑
ICDID	int	☐
ClaimStatus	tinyint	☐
Adjuster	int	☑
Adjustment	ntext	☑
Claimed	decimal(18, 2)	☑
Approved	decimal(18, 2)	☑
Reinsured	decimal(18, 2)	☑
Valuated	decimal(18, 2)	☑
DateClaimed	date	☐
DateProcessed	smalldatetime	☑
Feedback	bit	☐
FeedbackID	int	☑
Explanation	ntext	☑
FeedbackStatus	tinyint	☑
ReviewStatus	tinyint	☑
ApprovalStatus	tinyint	☑
RejectionReason	tinyint	☑
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐
ValidityFromReview	datetime	☑
ValidityToReview	datetime	☑
AuditUserIDReview	int	☑
RowID	timestamp	☑
HFID	int	☐
RunID	int	☑
AuditUserIDSubmit	int	☑
AuditUserIDProcess	int	☑
SubmitStamp	datetime	☑
ProcessStamp	datetime	☑
Remunerated	decimal(18, 2)	☑
GuaranteeId	nvarchar(50)	☑
ClaimAdminId	int	☑
ICDID1	int	☑
ICDID2	int	☑
ICDID3	int	☑
ICDID4	int	☑
VisitType	char(1)	☑

Objects that depend on tblClaim:	tblClaim depends on:
	

Reference to Data Model Entity: C**laim**

Notes: Feedback indicates if claim expects feedback. FeedbackID relates to a record in the feedback table if present. FeedbackID might be redundant but currently kept in design.

Field ClaimStatus :

- 1 = Rejected
- 2 = Entered
- 4 = Checked
- 8 = Processed
- 16 = Valuated

Field ReviewStatus

- 1 = Not selected
- 2 = Selected for review
- 4 = Reviewed

Field Feedback Status:

- 1 = Not selected
- 2 = Selected for feedback
- 4 = Delivered

6.1.31. tblClaimItems

ClaimItemID	int	
ClaimID	int	
ItemID	int	
ProdID	int	✓
ClaimItemStatus	tinyint	
Availability	bit	
QtyProvided	decimal(18, 2)	
QtyApproved	decimal(18, 2)	✓
PriceAsked	decimal(18, 2)	
PriceAdjusted	decimal(18, 2)	✓
PriceApproved	decimal(18, 2)	✓
PriceValuated	decimal(18, 2)	✓
Explanation	ntext	✓
Justification	ntext	✓
RejectionReason	smallint	✓
ValidityFrom	datetime	
ValidityTo	datetime	✓
LegacyID	int	✓
AuditUserID	int	
ValidityFromReview	datetime	✓
ValidityToReview	datetime	✓
AuditUserIDReview	int	✓
LimitationValue	decimal(18, 2)	✓
Limitation	char(1)	✓
PolicyID	int	✓
RemuneratedAmount	decimal(18, 2)	✓
DeductableAmount	decimal(18, 2)	✓
ExceedCeilingAmount	decimal(18, 2)	✓
PriceOrigin	char(1)	✓
ExceedCeilingAmount...	decimal(18, 2)	✓

Objects that depend on tblClaimItems:	tblClaimItems depends on:
tblClaimItems	tblClaimItems tblClaim tblItems tblProduct

Reference to Data Model Entity: **Remark7**

Notes:

Field ClaimItemStatus:

- 1 = Rejected
- 2 = Passed

Field RejectionReason is reserved for possible use later

6.1.32. tblClaimServices

ClaimServiceID	int	
ClaimID	int	
ServiceID	int	
ProdID	int	☑
ClaimServiceStatus	tinyint	
QtyProvided	decimal(18, 2)	
QtyApproved	decimal(18, 2)	☑
PriceAsked	decimal(18, 2)	
PriceAdjusted	decimal(18, 2)	☑
PriceApproved	decimal(18, 2)	☑
PriceValuated	decimal(18, 2)	☑
Explanation	ntext	☑
Justification	ntext	☑
RejectionReason	smallint	☑
ValidityFrom	datetime	
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	
ValidityFromReview	datetime	☑
ValidityToReview	datetime	☑
AuditUserIDReview	int	☑
LimitationValue	decimal(18, 2)	☑
Limitation	char(1)	☑
PolicyID	int	☑
RemuneratedAmount	decimal(18, 2)	☑
DeductableAmount	decimal(18, 2)	☑
ExceedCeilingAmount	decimal(18, 2)	☑
PriceOrigin	char(1)	☑
ExceedCeilingAmount...	decimal(18, 2)	☑

Objects that depend on tblClaimServices:	tblClaimServices depends on:
tblClaimServices	tblClaimServices tblClaim tblProduct tblServices

Reference to Data Model Entity: **Remark6**

Notes:

Field ClaimServiceStatus:

- 1 = Rejected
- 2 = Passed

Field RejectionReason is reserved for possible use later

6.1.33. tblClaimDedRem

Column Name	Data Type	Allow Nulls
ExpenditureID	int	☐
PolicyID	int	☐
InsureeID	int	☐
ClaimID	int	☐
DedG	decimal(18, 2)	☑
DedOP	decimal(18, 2)	☑
DedIP	decimal(18, 2)	☑
RemG	decimal(18, 2)	☑
RemIP	decimal(18, 2)	☑
RemOP	decimal(18, 2)	☑
RemConsult	decimal(18, 2)	☑
RemSurgery	decimal(18, 2)	☑
RemDelivery	decimal(18, 2)	☑
RemHospitalization	decimal(18, 2)	☑
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐

Objects that depend on tblClaimDedRem:	tblClaimDedRem depends on:
tblClaimDedRem	tblClaimDedRem tblInsuree tblPolicy

Reference to Data Model Entity: **Expenditures for insuree**

Notes:

6.1.34. tblFeedback

Column Name	Data Type	Allow Nulls
FeedbackID	int	☐
ClaimID	int	☐
CareRendered	bit	☑
PaymentAsked	bit	☑
DrugPrescribed	bit	☑
DrugReceived	bit	☑
Asessment	tinyint	☑
CHFOfficerCode	int	☑
FeedbackDate	datetime	☑
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐

Objects that depend on tblFeedback:	tblFeedback depends on:
tblFeedback tblClaim	tblFeedback tblClaim

Reference to Data Model Entity: **Feedback**

Notes: ClaimID might be redundant but currently kept in design.

6.1.35. tblPolicyRenewals

Column Name	Data Type	Allow Nulls
RenewalID	int	☐
RenewalPromptDate	date	☐
RenewalDate	date	☐
NewOfficerID	int	☑
PhoneNumber	nvarchar(25)	☑
SMSStatus	tinyint	☐
InsureeID	int	☐
PolicyID	int	☐
NewProdID	int	☐
RenewalWarnings	tinyint	☑
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditCreateUser	int	☑
ResponseStatus	int	☑
ResponseDate	datetime	☑

Objects that depend on PolicyRenewals:	PolicyRenewals depends on:
tblPolicyRenewals tblPolicyRenewalDetails	tblPolicyRenewals tblOfficer tblPolicy tblProduct

Reference to Data Model Entity: **N/A**

Notes: This table is used by a webservice that populates renewal prompts with the (new) OfficerID, (new Product code as per 'use case' 5.17. A separate service will loop through this table and will sent SMS messages to the phones and will update the SMS status.

6.1.36. tblPolicyRenewalsDetails

Column Name	Data Type	Allow Nulls
RenewalDetailID	int	☐
RenewalID	int	☐
InsureeID	int	☐
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditCreateUser	int	☐

Objects that depend on PolicyRenewals:	PolicyRenewals depends on:
tblPolicyRenewalDetails	tblPolicyRenewalDetails tblInsuree tblPolicyRenewals

Reference to Data Model Entity: **N/A**

Notes: This table ise used by a webservice that populates renewal prompts. Every insuree that needs a Photo update (internal rules) will be hosted in this table in order to alert the officer.

6.1.37. tblRelIndex

Column Name	Data Type	Allow Nulls
RelIndexID	int	☐
ProdID	int	☐
RelType	tinyint	☐
RelCareType	char(1)	☐
RelYear	int	☐
RelPeriod	tinyint	☐
CalcDate	datetime	☐
RelIndex	decimal(18, 4)	☑
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐
DistrictId	int	☑
		☐

Objects that depend on tblClaimDedRem:	tblClaimDedRem depends on:
tblRelIndex	tblRelIndex tblProduct

Reference to Data Model Entity: **Indexes of relative prices**

Notes:

Field RelCareType:

- I = In patient
- O = OutPatient
- B = Both (general)

RelType:

- 1 = Yearly
- 4 = Quarterly
- 12 = Monthly

6.1.38. tblBatchRun

Column Name	Data Type	Allow Nulls
RunID	int	☐
DistrictID	int	☐
RunDate	datetime	☐
ValidityFrom	datetime	☐
ValidityTo	datetime	☑
LegacyID	int	☑
AuditUserID	int	☐
RunYear	int	☐
RunMonth	tinyint	☐
		☐

Objects that depend on tblBatchRun:	tblbatchRun depends on:
tblBatchRun tblBatch	tblBatchRun tblDistricts

Reference to Data Model Entity: **N/A**

Notes:

This table is used to hold the batches that were fully 'valuated' together for reporting toward accounting

6.1.39. tblIMISDefaults

Column Name	Data Type	Allow Nulls
DefaultID	int	☐
PolicyRenewalInterval	int	☑
FTPHost	nvarchar(50)	☑
FTPUser	nvarchar(50)	☑
FTPPassword	nvarchar(20)	☑
FTPPort	int	☑
FTPEnrollmentFolder	nvarchar(255)	☑
AssociatedPhotoFolder	nvarchar(255)	☑
FTPClaimFolder	nvarchar(255)	☑
FTPFeedbackFolder	nvarchar(255)	☑
FTPPolicyRenewalFolder	nvarchar(255)	☑
FTPPhoneExtractFolder	nvarchar(255)	☑
FTPOffLineExtractFolder	nvarchar(255)	☑
AppVersionBackEnd	decimal(3, 1)	☑
AppVersionEnquire	decimal(3, 1)	☑
AppVersionEnroll	decimal(3, 1)	☑
AppVersionRenewal	decimal(3, 1)	☑
AppVersionFeedback	decimal(3, 1)	☑
AppVersionClaim	decimal(3, 1)	☑
OffLineHF	int	☑
WinRarFolder	nvarchar(255)	☑
DatabaseBackupFolder	nvarchar(255)	☑
OfflineCHF	int	☑
SMSLink	nvarchar(500)	☑
SMSIP	nvarchar(15)	☑
SMSUserName	nvarchar(15)	☑
SMSPassword	nvarchar(50)	☑
SMSSource	nvarchar(15)	☑
SMSDlr	int	☑
SMSType	int	☑

Objects that depend on tblExtracts:	tblExtracts depends on:
None	**None**

Reference to Data Model Entity: **N/A**

Notes:

This table is used to hold default information for IMIS to function.
All FTP settings are required for the mobile phones to connect to IMIS and to transfer the information to the correct folders under the IMIS root folder.
The OfflineHF field should only contain a value <> 0 in case of an Offline installation. This field should in that case contain the HFID (DB Key of tblHF) of the Health Facility.

6.1.40. tblReporting

Column Name	Data Type	Allow Nulls
ReportingId	int	☐
ReportingDate	datetime	☐
DistrictId	int	☐
ProdId	int	☐
PayerId	int	☑
StartDate	date	☐
EndDate	date	☐
RecordFound	int	☐

Objects that depend on tblReporting:	tblReporting depends on:
None	**None**

Reference to Data Model Entity: **N/A**

Notes: This table is used to hold criterias used to generate Matching Funds Reports

6.1.41. tblControl

Column Name	Data Type	Allow Nulls
FieldName	nvarchar(50)	☐
Adjustibility	nvarchar(1)	☐
Usage	nvarchar(200)	☑
		☐

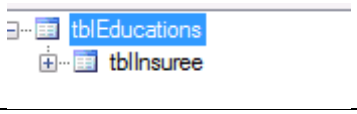
Objects that depend on tblControl:	tblControl depends on:
None	**None**

Reference to Data Model Entity: **N/A**

Notes: This table is used to handle the accessibility of the controls on the user interface for the country specific versions

6.1.42. tblEducations

Column Name	Data Type	Allow Nulls
EducationId	smallint	☐
Education	nvarchar(50)	☐
SortOrder	int	☑
AltLanguage	nvarchar(50)	☑

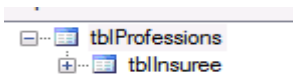
Objects that depend on tblEducations:	tblEducations depends on:
	**None**

Reference to Data Model Entity: **N/A**

Notes: This table is used to hold records for the list of Education Level

6.1.43. tblProfessions

Column Name	Data Type	Allow Nulls
ProfessionId	smallint	☐
Profession	nvarchar(50)	☐
SortOrder	int	☑
AltLanguage	nvarchar(50)	☑
		☐

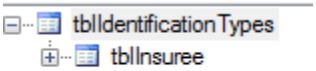
Objects that depend on tblProffessions:	tblProffessions depends on:
	**None**

Reference to Data Model Entity: **N/A**

Notes: This table is used to hold records for the list of Professions in the Country.

6.1.44. tblIdentificationTypes

Column Name	Data Type	Allow Nulls
IdentificationCode	nvarchar(1)	☐
IdentificationTypes	nvarchar(50)	☐
AltLanguage	nvarchar(50)	☑
SortOrder	int	☑
		☐

Objects that depend on tblIdentificationTypes:	tblIdentificationTypes depends on:
	**None**

Reference to Data Model Entity: **N/A**
Notes: This table is used to hold records for the list of Professions in the Country.

6.1.45. tblLegalForms

Column Name	Data Type	Allow Nulls
LegalFormCode	char(1)	☐
LegalForms	nvarchar(50)	☐
SortOrder	int	☑
AltLanguage	nvarchar(50)	☑

Objects that depend on tblLegalForms:	tblLegalForms depends on:
tblLegalForms tblHF	**None**
Reference to Data Model Entity: **N/A**	
Notes: This table is used to hold records for the list of Legal forms.	

6.1.46. tblRelations

Column Name	Data Type	Allow Nulls
RelationId	smallint	☐
Relation	nvarchar(50)	☐
SortOrder	int	☑
AltLanguage	nvarchar(50)	☑

Objects that depend on tblRelations:	tblRelations depends on:
tblRelations tblInsuree	**None**
Reference to Data Model Entity: **N/A**	
Notes: This table is used to hold records for the list of Relationships.	

6.1.47. tblFamilyTypes

Column Name	Data Type	Allow Null
FamilyTypeCode	nvarchar(2)	☐
FamilyType	nvarchar(50)	☐
SortOrder	int	☑
AltLanguage	nvarchar(50)	☑

Objects that depend on tblFamilyTypes:	tblFamilyTypes depends on:
tblFamilyTypes tblFamilies	**None**

Reference to Data Model Entity: **N/A**

Notes: This table is used to hold records for the list of Family Types.

6.1.48. tblConfirmationTypes

Column Name	Data Type	Allow Nulls
IdentificationCode	nvarchar(1)	☐
IdentificationTypes	nvarchar(50)	☐
AltLanguage	nvarchar(50)	☑
SortOrder	int	☑

Objects that depend on tblConfirmationTypes:	tblConfirmationTypes depends on:
tblConfirmationTypes tblFamilies	**None**

Reference to Data Model Entity: **N/A**

Notes: This table is used to hold records for the list of Confirmation Type.

6.1.49. tblCeilingInterpretation

Column Name	Data Type	Allow Nulls
CeilingIntCode	char(1)	☐
CeilingIntDesc	nvarchar(100)	☐
AltLanguage	nvarchar(100)	☑
SortOrder	int	☑

Objects that depend on tblCeilingInterpretation:	tblCeilingInterpretation depends on:
tblCeilingInterpretation tblProduct	**None**

Reference to Data Model Entity: **N/A**

Notes: This table is used to hold records for the list of Ceiling Interpretation.

6.1.50. tblLanguages

Column Name	Data Type	Allow Nulls
LanguageCode	nvarchar(2)	☐
LanguageName	nvarchar(50)	☐
SortOrder	int	☑

Objects that depend on tblLanguages:	tblLanguages depends on:
tblLanguages tblUsers	**None**

Reference to Data Model Entity: **N/A**

Notes: This table is used to hold records for the list of Languages.

6.1.51. tblEmailSettings

Column Name	Data Type	Allow Nulls
EmailId	nvarchar(200)	☐
EmailPassword	nvarchar(200)	☐
SMTPHost	nvarchar(200)	☐
Port	int	☐
EnableSSL	bit	☐

Objects that depend on tblEmailSettings:	tblEmailSettings depends on:
None	**None**

Reference to Data Model Entity: **N/A**

Notes: This table is used for Handling email setting.

6.1.52. tblHFSublevel

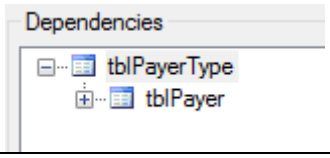
Column Name	Data Type	Allow Nulls
HFSublevel	char(1)	☐
HFSublevelDesc	nvarchar(50)	☑
SortOrder	int	☑
AltLanguage	nvarchar(50)	☑

Objects that depend on tblHFSubLevel:	tblHFSubLevel on:
Dependencies tblHFSublevel tblHF	**None**

Reference to Data Model Entity: **N/A**

6.1.53. tblPayerType

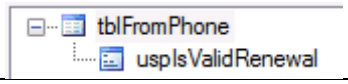
Column Name	Data Type	Allow Nulls
Code	char(1)	☐
PayerType	nvarchar(50)	☐
AltLanguage	nvarchar(50)	☑
SortOrder	int	☑

Objects that depend on tblPayerType	tblPayerType Depends on:
	**None**

Reference to Data Model Entity: **N/A**

6.1.54. tblFromPhone

Column Name	Data Type	Allow Nulls
FromPhoneId	int	☐
DocType	nvarchar(3)	☐
DocName	nvarchar(200)	☐
DocStatus	nvarchar(3)	☑
LandedDate	datetime	☐
OfficerCode	nvarchar(8)	☑
CHFID	nvarchar(9)	☑
PhotoSumittedDate	datetime	☑
ClaimId	int	☑

Objects that depend on tblFromPhone:	tblFromPhone depends on:
	**None**

Reference to Data Model Entity: **N/A**

Notes: This table is used for Handling all records which are uploaded from the phone.

6.1.55. tblSubmittedPhotos

Column Name	Data Type	Allow Nulls
PhotoId	int	☐
ImageName	nvarchar(50)	☑
CHFID	nvarchar(9)	☑
OfficerCode	nvarchar(8)	☑
PhotoDate	date	☑
RegisterDate	datetime	☑

Objects that depend on tblSubmittedPhotos:	tblSubmittedPhotos on:
	None

Reference to Data Model Entity: **N/A**

Notes: This table is used for Handling photograph records when the service runs.

6.1.56. tblLogins

Column Name	Data Type	Allow Nulls
LoginId	int	☐
UserId	int	☑
LogTime	datetime	☑
LogAction	int	☑

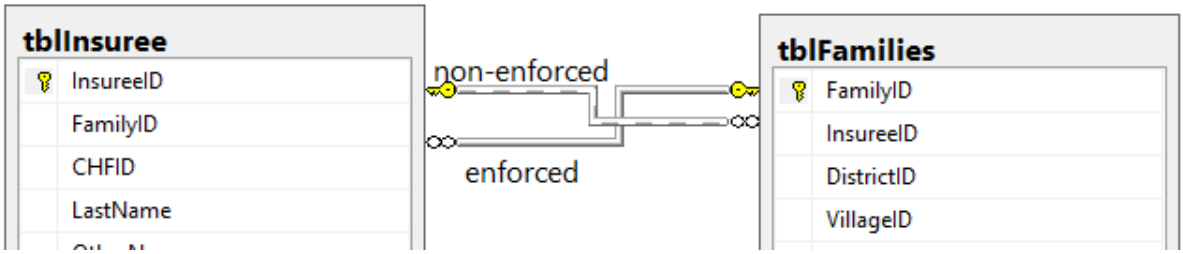
Objects that depend on tblLogins:	tblLogins depends on:
tblLogins uspCleanTables uspSSRSUserLogReport	tblLogins tblUsers

Reference to Data Model Entity: **N/A**

Notes: This table is used for store user activities when they login

6.2. Graphical relationship overviews

The graphical representation of relationships is covered in this paragraph in separate so-called database diagrams. We could have enforced and non-enforced relationships as shown below:



Each diagram contains a limited amount of tables to make the diagrams more readable. Some tables might be included in several diagrams for clarity of the model.

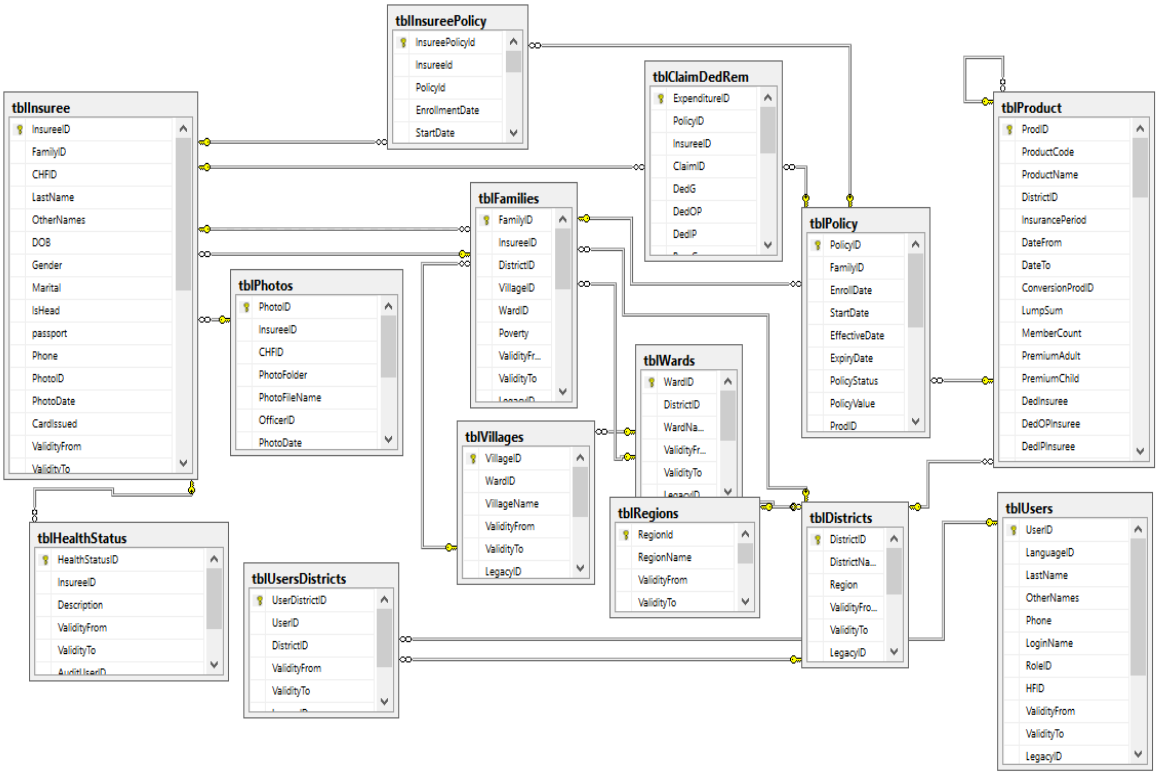


Diagram 1: Family-Districts-Villages-Municipality -Users-Insuree-Policy-Product-Relation-Language Professions

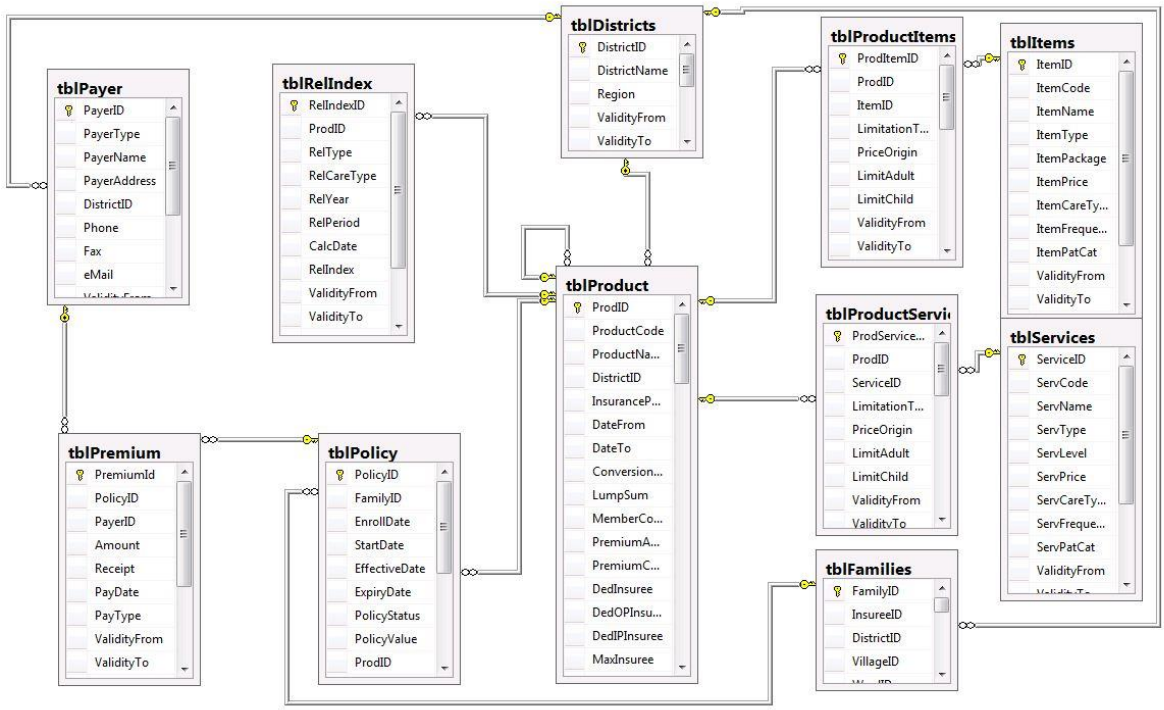


Diagram 2: Payer-Contributions-Policy-Product-Family

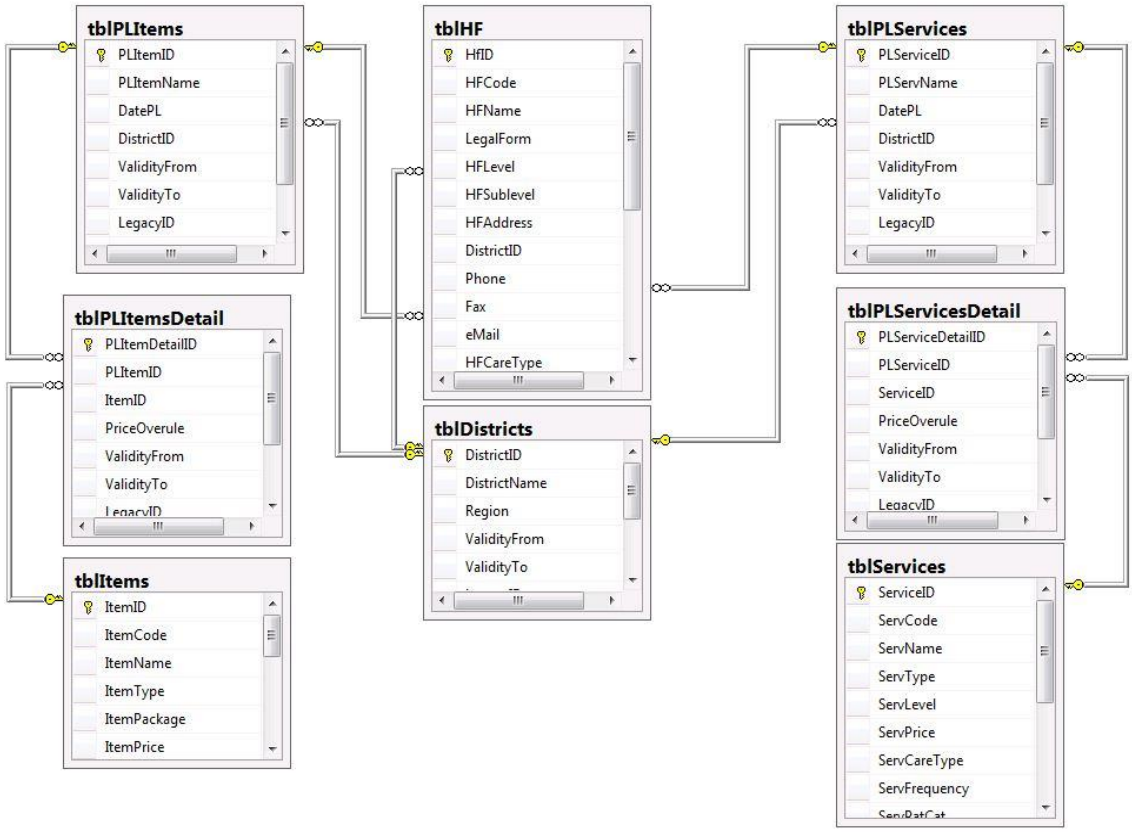


Diagram 3: HF-Pricelists

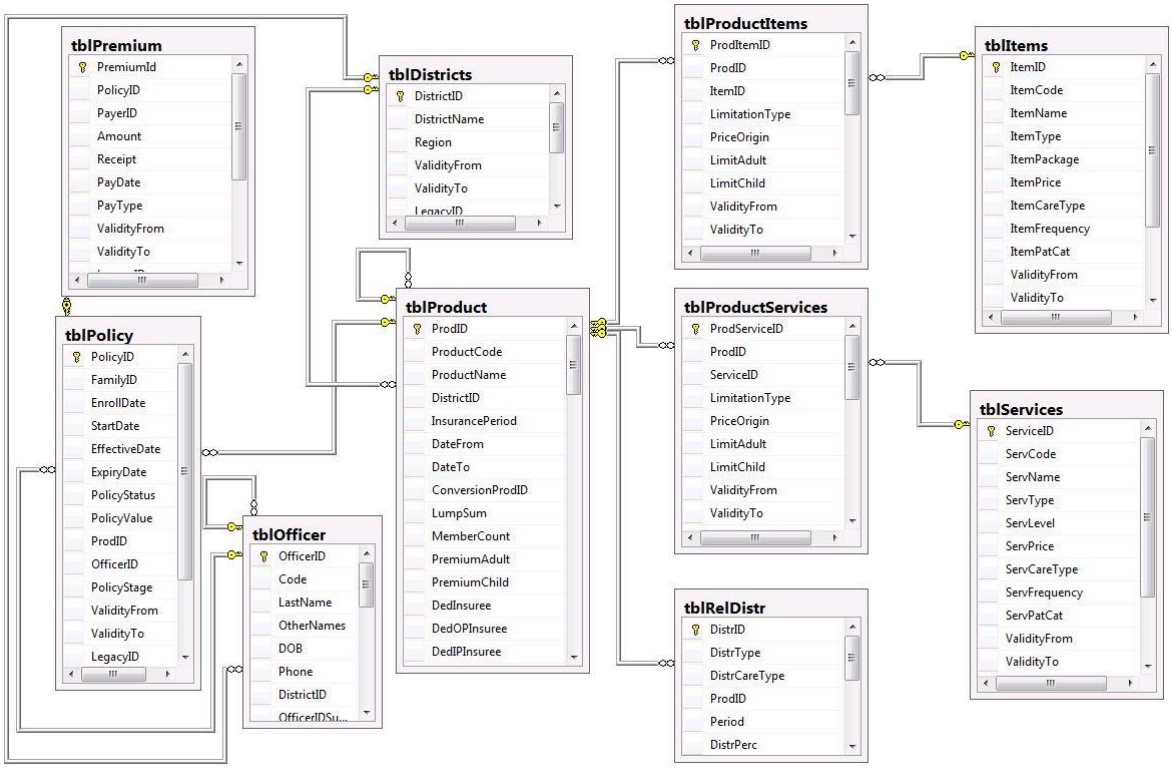


Diagram 4: Policy-Product-Contributions

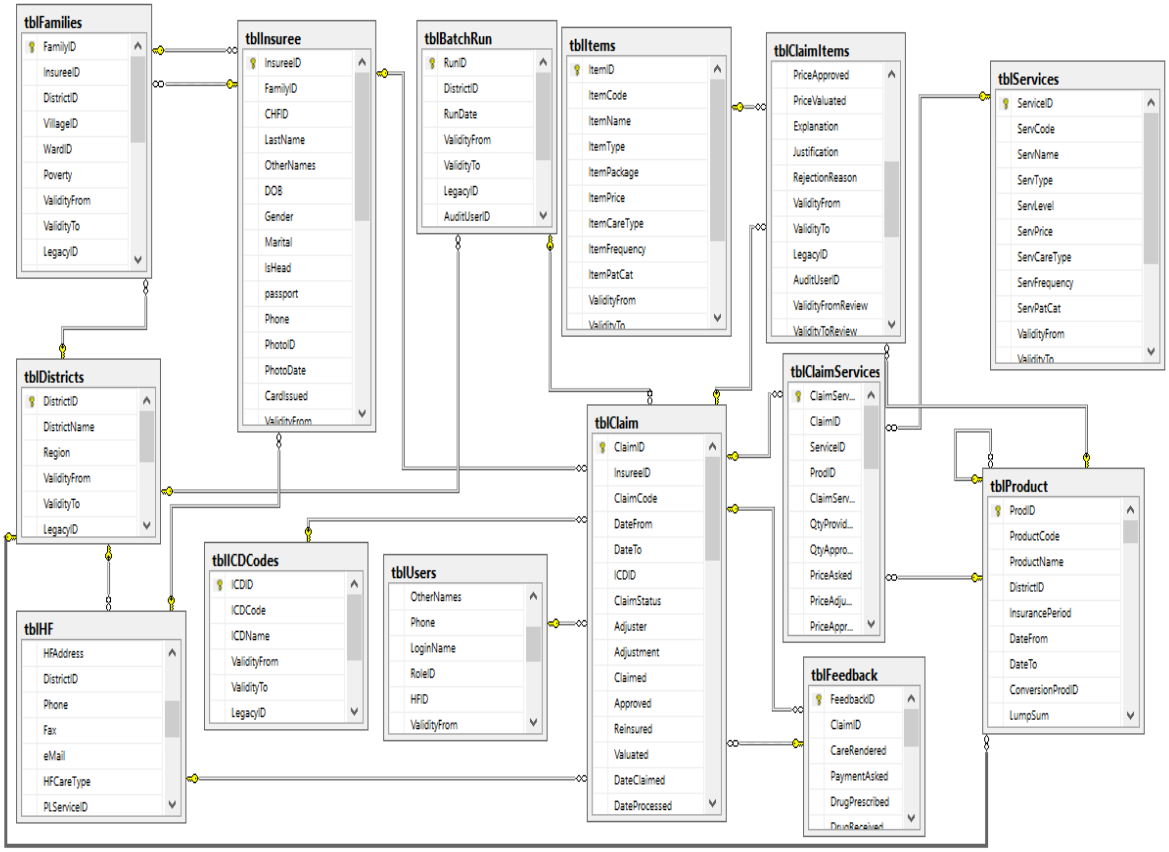


Diagram 5: HF-(Batch)-Claims-Products-family-Insuree

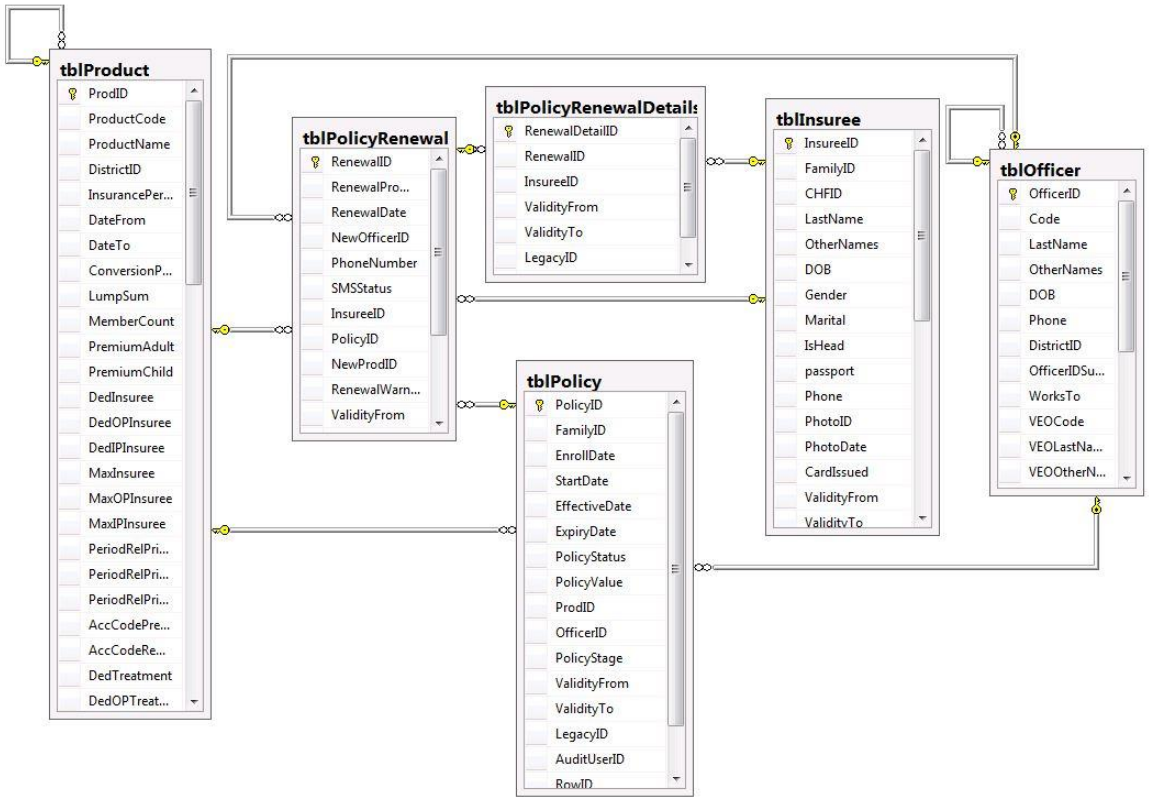


Diagram 6: Policies-Renewals-Officers

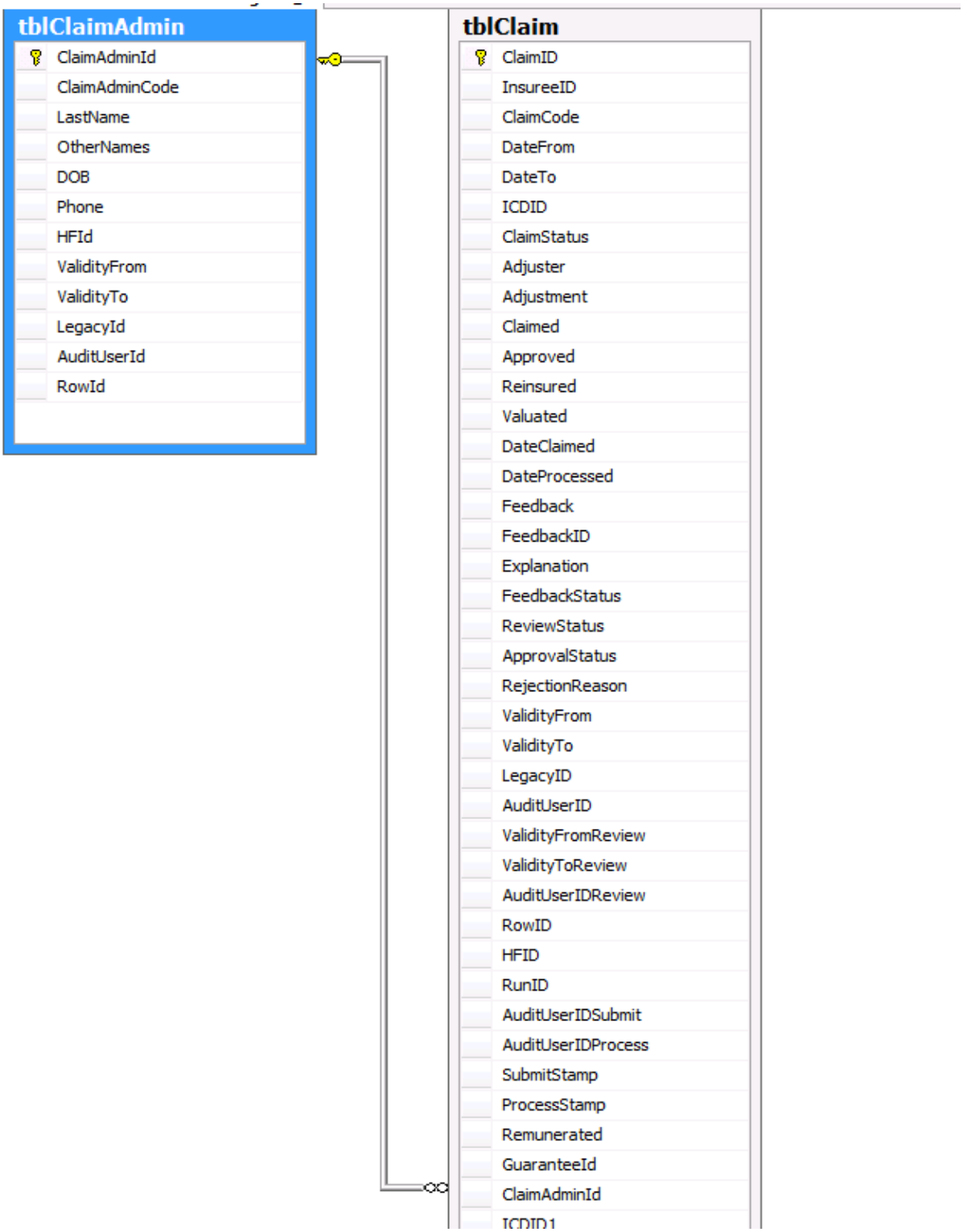


Diagram 7: Claim Administrator

6.3. IMIS On-line and Off-line

The online and offline client will use exactly the same 'web-application' but differ on few areas:

- Off line client has an additional database for keeping claim objects separate
- Off line client will have an additional feature for creating XML batches

The off-line client will operate independent from the Central database and will connect to an instance of a locally (LAN) installed SQL Server. The SQL Server will host 2 separate databases. These databases will include in their names 'XXX_IMIS', where XXXX stands for the unique code for the facility/hospital (TBD).

- 'Copy' of the IMIS: XXX_IMIS
- XXX_IMIS

The 'Copy' of Central will be restored on periodical basis with a special developed tool. All databases will be protected by password security for restore actions (so called SQL Server backup password shipped with the physical backup).
Automatically after restoration, the data not needed for the off-line client, will be flushed from the system. Depending on the rules to be set for the availability of data for the off-line client and security constraints, data will be (partially) flushed from the IMIS database. Data flushing parameters might be for example DistrictID and HFID.

At this stage (in the absence of the design of the Claim management module) we anticipate flushing the following tables from the IMIS. These tables will be hosted in the additional offline database 'XXX_IMIS'

- tblBatch
- tblClaim
- tblClaimItems
- tblClaimServices
- tblFeedback
- tblPolicyRenewals
- tblpolicyRenewalsDetails

The following tables might be candidates for manipulation/partial deletion:

- tblUsers
- tblUsersDistricts

- tblHF
- tblPLItems
- tblPLItemsDetail
- tblPLServices
- tblPLServicesDetail

- tblFamilies
- tblInsuree
- tblHealthStatus
- tblPolicy
- tblPhotos
- tblPremium
- tblPayer
- tblOfficer

The following tables are anticipated to remain as a full copy in the XXXX_IMIS:

- tblDistricts
- tblVillages
- tblWards
- tblItems
- tblServices
- tblProduct
- tblProductItems
- tblProductServices
- tblICDCodes

The OFFLINE_IMIS database will host the following tables:

- tblBatch
- tblClaim
- tblClaimItems
- tblClaimServices
- tblFeedback

These tables will have exactly the same structure as the tables hosted in the IMIS database.
The offline application will utilize all other information from the IMIS database for referencing.

7. Windows and Web Services

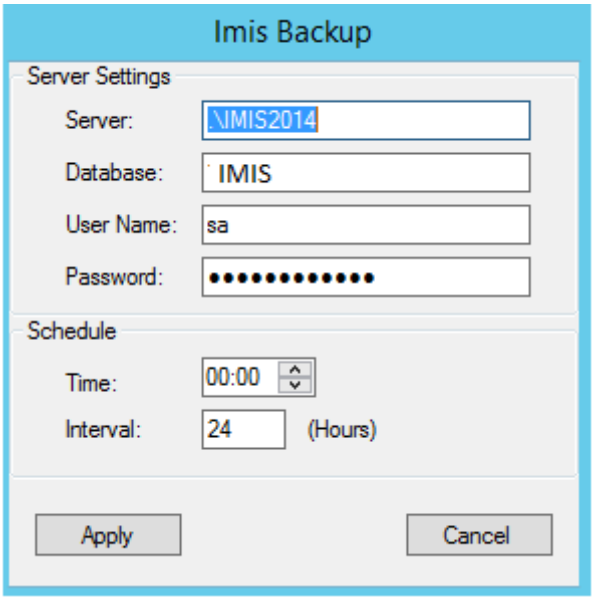
7.1. Windows Services

The IMIS Windows services are automatically started when the computer boots, but can be paused and restarted at any time through the user interface as discussed below.

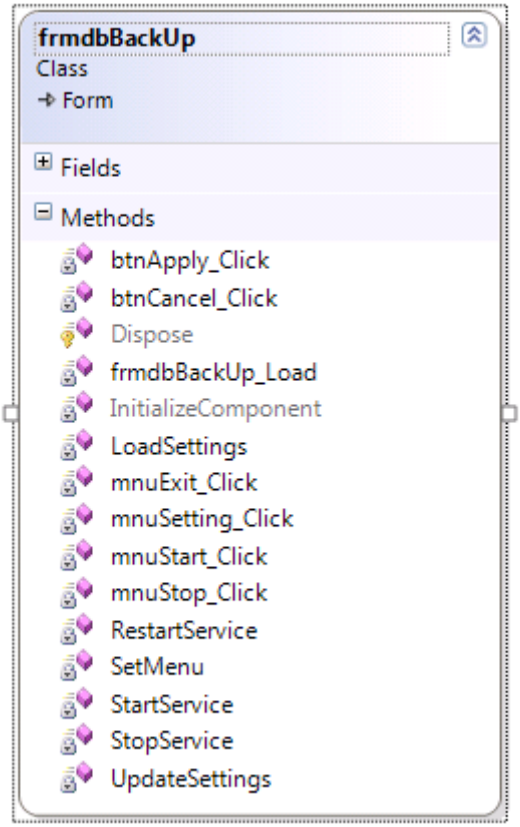
7.1.1. IMIS Backup

This service is used for database backup, after the service installation, the user has to set the name of the server, the database name, the SQL-server instance credentials, and the service schedule for the backups.

The backup location is specified in table **tblIMISDefaults** under the live database. The figure below shows the interface for the database backup service.



IMIS Backup Interface

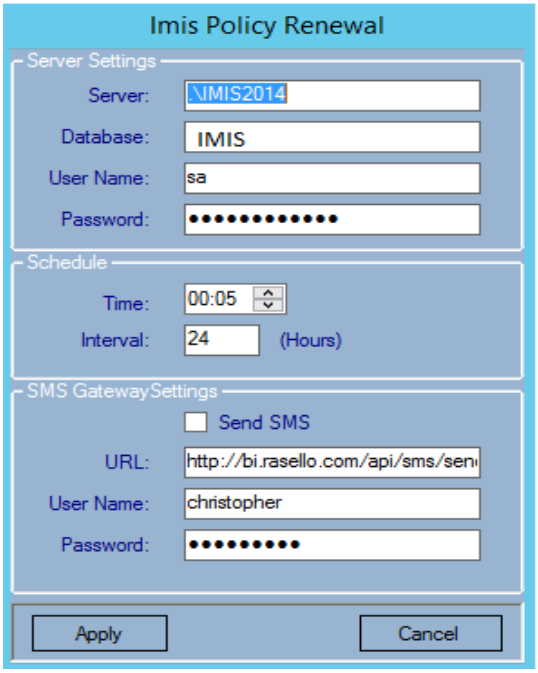
Interface object name:	*frmDbBackUp*
Action Button:	*Setting*
Class Diagram	

7.1.2.IMIS Policy Renewal

This service is used for updating insurance policies and runs every day at the specified time while check if there is any policy which is about to expire within specified period of time. If it finds a policy which meets the specified expiry period, it flags it and sets the policy status to expired.

The Service also send SMS to insuree if their policy is about to expire so that they can renew the policy. After the service installation, the user has to set the name of the server, the database name, the SQL-server instance credentials, and the service schedule for checking which policy is to be flagged as expired policy.

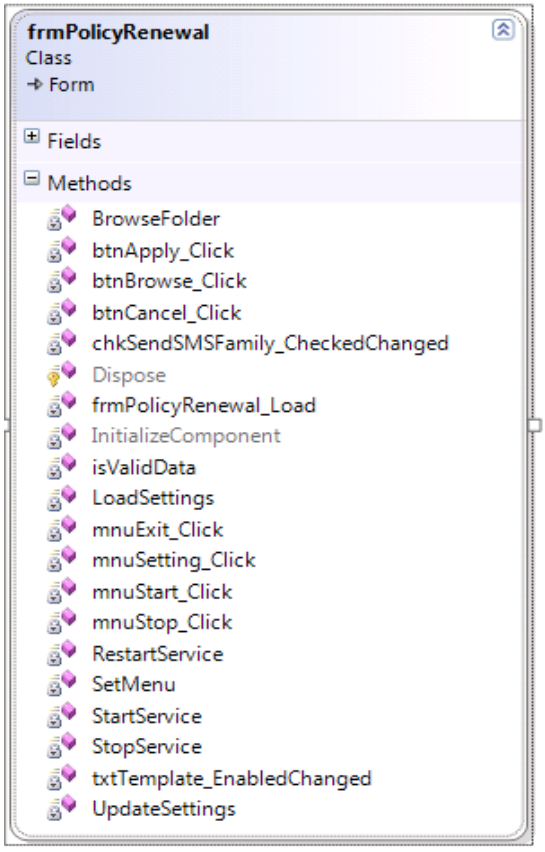
The days specified before the exact policy expiry date is provided in table **tblIMISDefaults** under the IMIS Live database. The figure below shows the interface for policy renewal service.



Imis Policy Re-new Service

Under the SMS gateway setting, you need to specify the url given by SMS Gateway provider. You also need to provide the Username and password to allow the service to send SMS to the insuree.

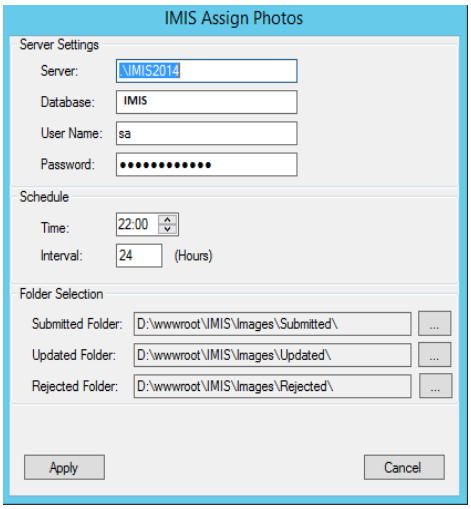
Class Diagram

Interface object name:	*frmPolicyRenew.vb*
Action Button:	*Setting*
Class Diagram	

7.1.3.AssignPhotoService

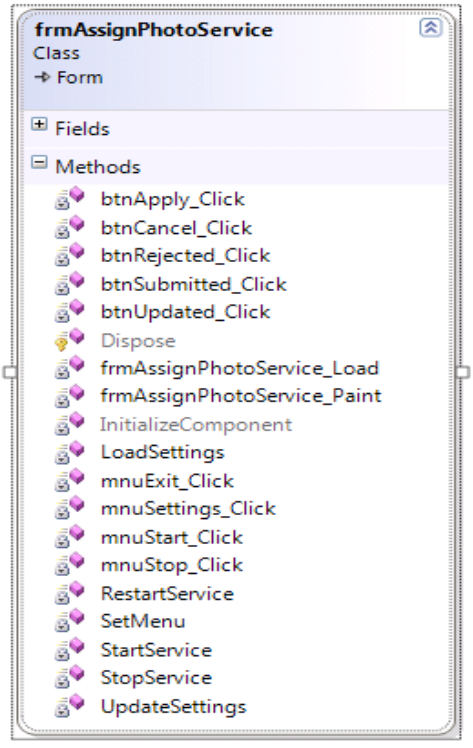
This service is used to move pictures from submitted folder to updated folder once the picture are assigned to an insuree.
After install the service user has to set the name of the server, database name, username and password of the SQL-server instance, and the schedule that service will use for updating the pictures.

Folder selection: the user has to provide a path for the submitted pictures, the updated pictures and the Rejected pictures as shown in the image below.



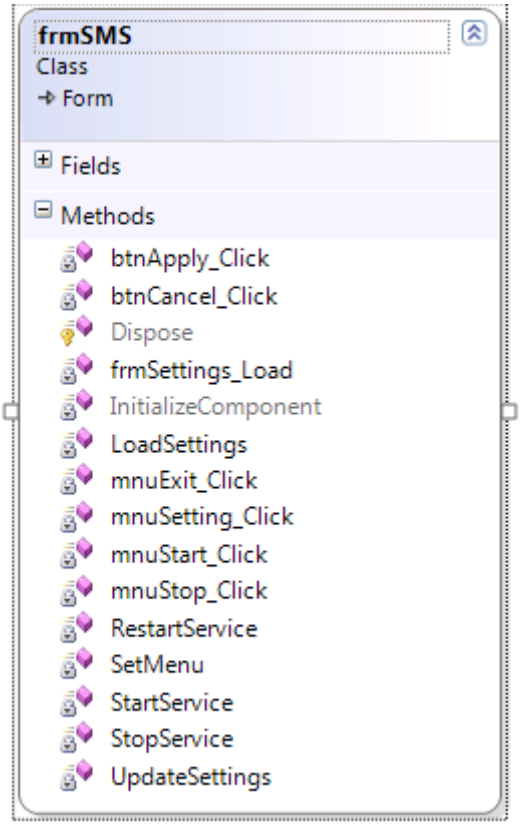
Assign Photo

Class Diagram

Interface object name:	*frmAssignPhotoService.vb*
Action Button:	*Setting*
Class Diagram	

7.1.4.SMSONEFFECTIVE

This service is used to send SMS to the insuree when their policy becomes effective.
On setting the service, the user has to specify the url given by SMS Gateway provider. They also need to provide the Username and password to allow the service to send SMS to the insuree.

Interface object name:	*frmSMS*
Action Button:	*Setting*
Class Diagram	

7.2.Web Services

A Web Service is used for exchanging data between applications or systems.
The IMIS Web Services is used for the exchange of information between the android application (Java based) and the IMIS Application (aspx). The Web Service can be used by any other application to communicate with Core IMIS application.

The IMIS Web Service named as getFTPCredentials, which is used for communication between android application and the core IMIS application has the below functions:

- CheckServerPath
  This function is used to check if the server path specified in the application is available in a server or not.

- CreatePhoneExtracts

This function is used for Phone extract in the background; the service does the extraction without the user intervention. As soon as this function completes the extraction, the service will send a link via email to the user with information regarding the extract downloads.

- DiscontinuePolicy
  This function is used to discontinue the policy when the insuree decides that they no longer require that policy.
- EnquireInsuree
  This function is used for requesting the insuree information from the enquire application.

- GetClaimStats
  This function is used to show the claim statement. It shows the number of claims passed or failed in a specified period of time.

- GetCurrentVersion
  This function is used to check the current version of the android application.

- GetEnrolmentStats
  This function is used to get the statement for the insure enrolments. The enrolment officer will be able to check the number of insurees enrolled over a specified period of time.

- GetFeedbackStats
  This function is used to show the feedback statements for the claims which have been selected for feedback for the specified period of time by a given officer.

- GetPayers
  This function is used to retrieve all payers.

- GetRenewalStats
  This function is used to get Renewal statements for the policy that Enrolment officer has renewed for a specified period of time.

- InsertPhotoEntry
  This function is used to insert photo details to a table and also upload the photos to server.

- SendEmail
  This function is used for sending emails.

- getFTPCredentials
  This function is to get the ftp credential.

- getFeedbacks
  This function is used to retrieve all the claims which are selected for feedback for a specific Enrolment officer.

- getRenewals
  This function is used to retrieve all the policies which are pending renewal by a specific Enrolment officer.

- isUniqueReceiptNo
  This function is used to check if the receipt number exists or not.

- isValidClaim

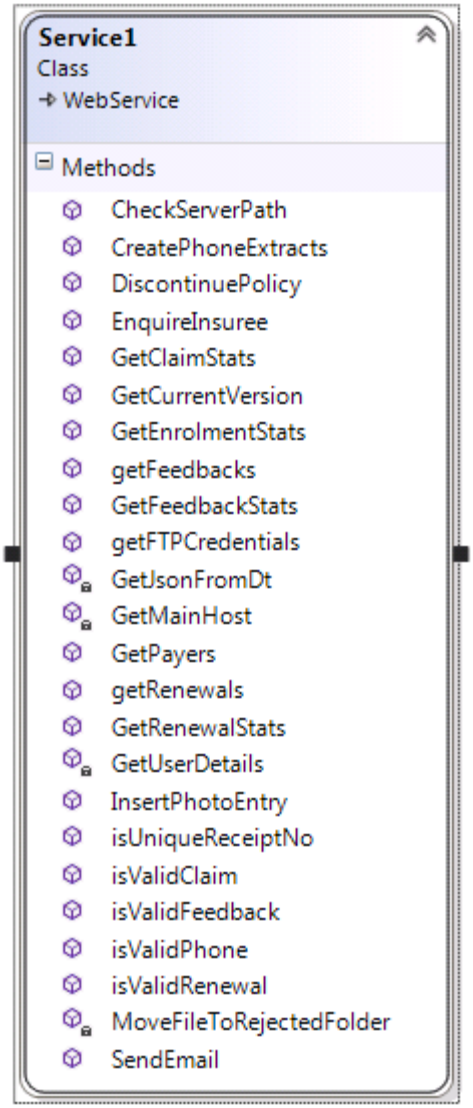
This function is used for checking if the submitted claim is successful or not.

- isValidFeedback
  This function is used for checking if the submitted feedback is successful or not.

- isValidPhone
  This function is used for checking if the phone belongs to the valid officer.

- isValidRenewal
  This function is used to check if the policy is valid for renewal.

Class Diagram for web services

Interface object name:	*GetFTCredential Service*
Action Button:	
Class Diagram	

8. Mobile phone Concept

The mobile phone concept is using standard XML files and making use of naming conventions for photo storage, XML is having its own internal structure (schema) and could be used for exchange of standard data such as feedback and Claim data.

XML files could be prepared from the Main IMIS database and serve as a database for off-line Mobile phones.

The concept is for almost all scenarios similar….. First the data is stored in XML and JPEG format and will be stored on the local SD card. Thereafter the phone app will attempt to transmit to the FTP folder in case the phone is 'on-line'. In case the phone is off-line, it will keep the files locally stored. A separate function/feature will attempt to send the files at the moment the phone app detects a phone signal.

Apps used are

- Enrollment Application
- Policy Inquiring Application
- Claim Application
- Feedback and Renew Application.

Before we further discuss each of the applications, cover the technologies used for the development of the Phone Apps.

8.1.1. Mobile phone technologies

System Requirements:

- Android Platform 2.1 and above
- Inbuilt camera
- External Storage Device (e.g. SD card)
- Mobile phone/Internet connection (Optional)

Languages:

- JAVA

Technologies used *:

- Android
  - Android is a software stack for mobile devices that includes an operating system, middleware and key applications.
- JSON
  - JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.
- XML

  To store the information of the insuree we have 2 choices, **1.** SQLite database or **2.** XML Files. We are working on XML rather than SQLite. There are many reason behind choosing XML and not SQLite. Few of them are listed below.

- XML is **platform free** that means it does not need any kind of installation or third party utility to read or write.
- XML is supported on computers as well as on phones. SQLite is limited to the mobile phone. So in that latter case we will have to rely on other technologies and services to transfer and receive data from/to server.
- Transferring data from SQLite to MSSQL server requires extra web services to be created.
- XMLs are light weight. So very easy to transfer.
- XMLs can be transferred via email or any other external device as it is one of the most used and accepted data exchange formats on file level. This implicates any one can take an XML file on a Flash drive and transfer it to the server or phone in case no network coverage is available.

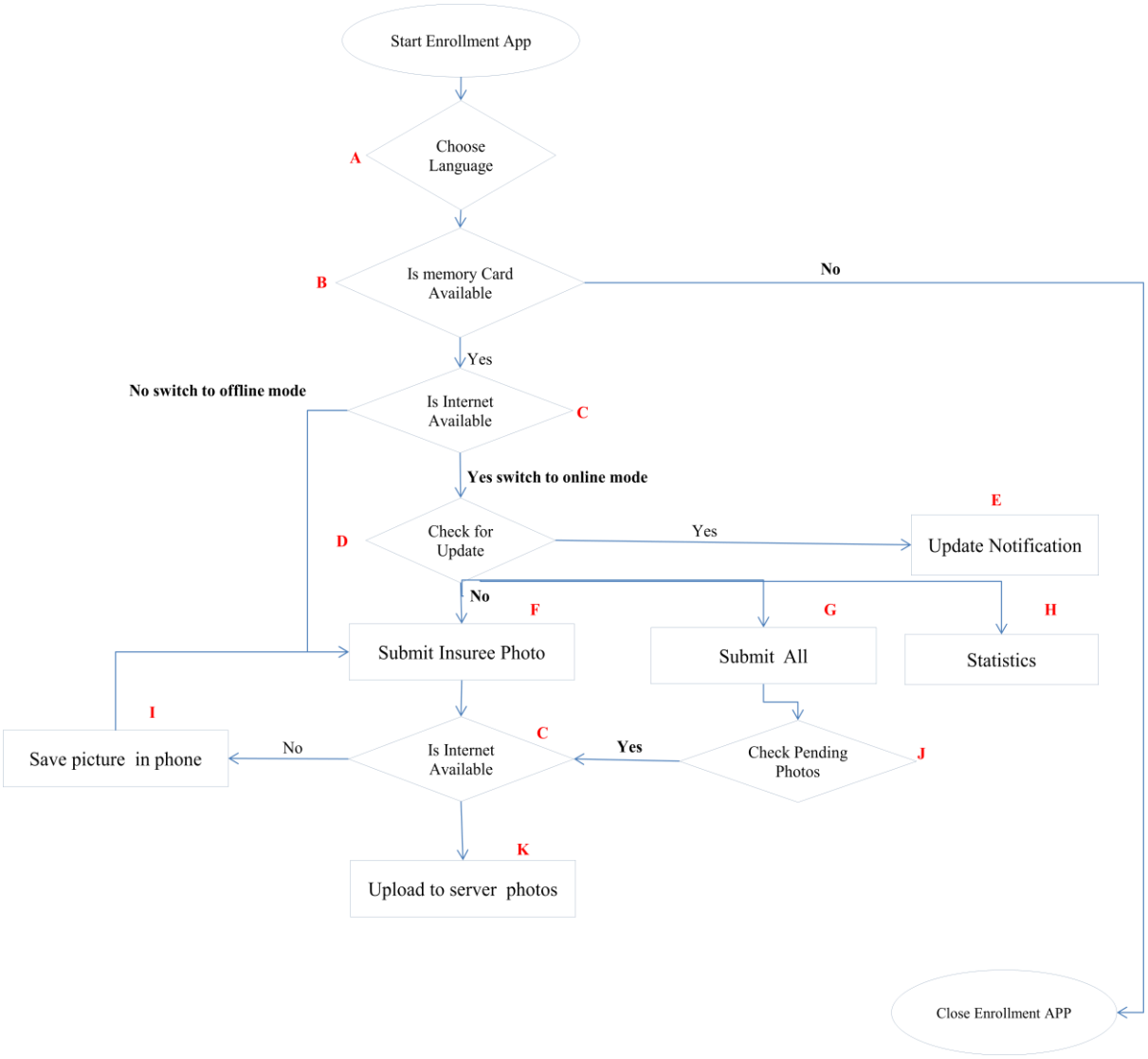
With the above in mind XML will be used for the data exchange format between server and mobile phones for all applications.

- SQLite
  - SQLite is available on every Android device. Using a SQLite database in Android does not require any database setup or administration. We might opt to use SQLite for internal purposes on the 'claims' phone app only.

8.2. Enrollment Application

This application is used by the enrollment officers for the purpose of enrolling families with their dependants (Insurees) in the system.

8.2.1. Enrollment Flow chart diagram



In the code snippets below, the main functions of importance are shown in **bold**.

- The function for Choosing Language **(A)**

```
public void ChangeLanguage(Context ctx,String Language){
 Resources res = ctx.getResources();
 DisplayMetrics dm = res.getDisplayMetrics();
 android.content.res.Configuration config = res.getConfiguration();
 config.locale = new Locale(Language.toLowerCase());
 res.updateConfiguration(config, dm);
     }
```

- The function for checking for Memory Card **(B)**

```
public int isSDCardAvailable(){
String State = Environment.getExternalStorageState();
if (State.equals(Environment.MEDIA_MOUNTED_READ_ONLY)){
```

```
return 0;
        else if(!State.equals(Environment.MEDIA_MOUNTED)){
return -1;
        }else{
return 1;
        }
}
```

- The function for checking the internet Availability **(C)**

```
public boolean isNetworkAvailable(Context ctx)
{
ConnectivityManager cm = (ConnectivityManager)
ctx.getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo ni = cm.getActiveNetworkInfo();

return (ni != null && ni.isConnected());

        }
```

- The function for checking if there is a new application version **(D)**

```
public boolean isNewVersionAvailable(String Field,Context ctx, String PackageName)
{
String result;
CallSoap cs = new CallSoap();
cs.setFunctionName("GetCurrentVersion");
result = cs.GetCurrentVersion(Field);
if (result == "") return false;
return (!getVersion(ctx,PackageName).toString().equals(result));

}
```

- The event for button **"Submit"** - uploads insuree photo **(F)**

```
btnSubmit.setOnClickListener(new OnClickListener() {

@Override
public void onClick(View v) {
if (!isValidate())return;

pd = ProgressDialog.show(EnrollmentActivity.this, "", getResources().getString(R.string.Uploading));
new Thread(){
public void run(){
try {
result = SubmitData();
} catch (IOException e) {
e.printStackTrace();
}

runOnUiThread(new Runnable() {
public void run() {
switch(result){
//case for return the message after upload
}
}
});
 pd.dismiss();
}
}.start();
        }
});
```

- The **"Upload All"** button event - for uploading images to the server which were saved in the phone(offline mode) **(G)**

```
Case R.id.Upload:
                    //Get the total number of files to upload
            Images = GetListOfFiles(Path);
            TotalImages = Images.length;

            //If there are no files to upload give the message and exit
            if (TotalImages == 0){
                    ShowDialog(getResources().getString(R.string.NoImages));
                    return false;
            }

            //If internet is not available then give message and exit
            if (!_General.isNetworkAvailable(this)){
                    ShowDialog(getResources().getString(R.string.CheckInternet));
                    return false;
            }

            pd = new ProgressDialog(this);
            pd.setCancelable(false);

            pd = ProgressDialog.show(this,"",getResources().getString(R.string.Uploading));

            new Thread(){
                    public void run(){

                            //Check if valid ftp credentials are available
                            if(ConnectsFTP()){
                                    //Start Uploading images
                                    UploadAllImages();
                            }else{
                                    result = -1;
                            }

                            runOnUiThread(new Runnable() {

                                    @Override
                                    public void run() {
                                            switch(result){
                                             //Case return result Value
                                            }
                                    }
                            });

                              pd.dismiss();
                    }

            }.start();

      return true;
```

- The **"Statistics"** button event **(H)**

```
 case R.id.mnuStatistics:

    if(!_General.isNetworkAvailable(EnrollmentActivity.this)){
       ShowDialog(getResources().getString(R.string.InternetRequired));
       return false;
    }
```

```
if(etOfficer.getText().toString().length() == 0){
   ShowDialog(getResources().getString(R.string.MissingOfficer));
   return false;
}

Intent Stats = new Intent(EnrollmentActivity.this,Statistics.class);
Stats.putExtra("Title",getResources().getString(R.string.Statistics));
Stats.putExtra("OfficerCode",etOfficer.getText().toString());
EnrollmentActivity.this.startActivity(Stats);
return true;
```

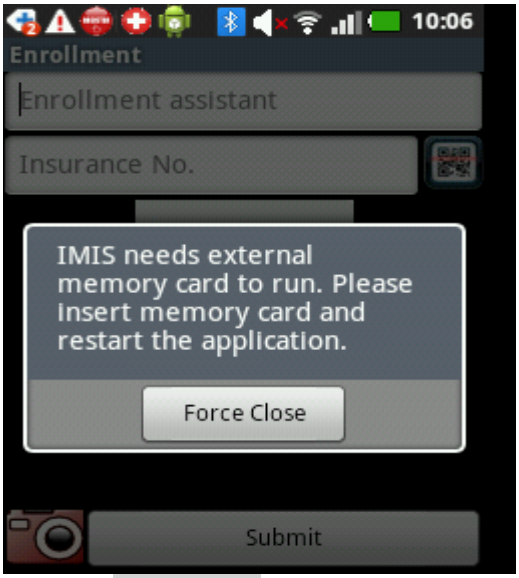
- The function for Uploading images to the server **(K)**

```
//Upload image to the server if network is available
    if(_General.isNetworkAvailable(this)){

            if(uf.uploadFileToServer(this,file)){
                    //File uploaded to server successfully
        RegisterUploadDetails(file.getName());
                    file.delete();
                    return 1;
            }else
            {
                    //Network is available but file not uploaded
                    return 2;
            }
    }else{
            //Network is not available so file is stored on external memory
            return 0;
    }
```

The application will work with 2 modes i.e online mode and offline mode depending on the availability of the data network.
On every start of the application it checks the following:

- SD Card availability
- Connectivity availability.

If the system does not find a SD card or the SD Card is in read only mode, it will prompt the user to first insert a SD card and forces the user to close the application. This because the application stores all data in external the storage device and the application cannot run without this storage. Below is the actual screen prompting the user.



This task is carried out in the class called "**General.java**"

The function "`public int isSDCardAvailable()`" is used to check if a SDCard is available for storage or not.

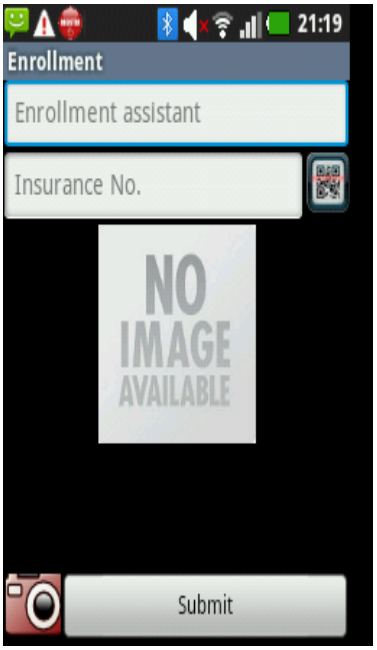
This task is carried out in the class called "**General.java**"

The function "`public boolean isNetworkAvailable(Context ctx)`" returns Boolean value.

The enroll function has three main input fields:

- Enrollment Assistant Code
- Insurance Number
- Image

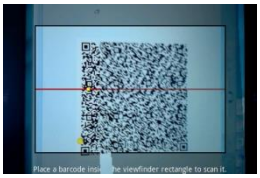
All functionality related to enrollment is implemented in the class "EnrollActivity.java"



The operator has to identify by entering his/her enrollment officers' code.

Only when the application starts one needs to enter this code, the code remains valid for subsequent entries of Insuree thereafter.

The Insurance Number can be scanned by using the inbuilt Scanner of the mobile phone or could alternatively be entered manually.
By clicking on the Scan Code button, the application will launch the phone's inbuilt camera and will start scanning the QR code. Refer to the image below.



In case the scan was successful, the Insurance Number will be shown and the operator could assign a Photograph to the newly scanned Insurance Number.

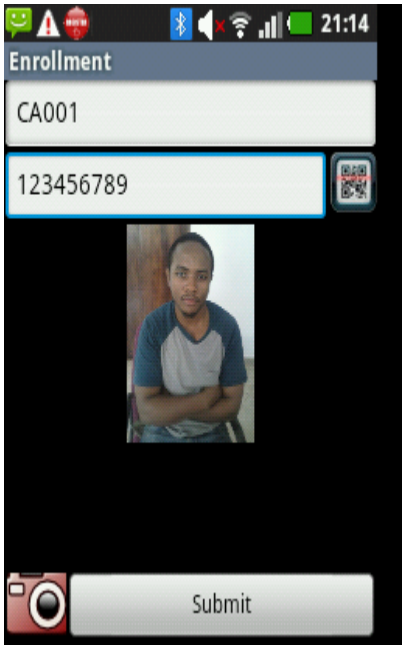
Please refer the procedure "`btnScan.setOnClickListener(new OnClickListener()`" which calls the function "`startActivityForResult(<intent>,1)` " with the request code 1. Intent is the instruction object, passed to this procedure, for activating a QR code scan using the inbuilt camera.

To assign the image of an insuree, one has to click on the 'Take photo' button. By clicking on the Take Photo button, the application will launch the phone's inbuilt camera with 2 buttons: 'Save' and 'Retry' on other phone you will see camera to retry or attachment picture to save image, as displayed in the image below.

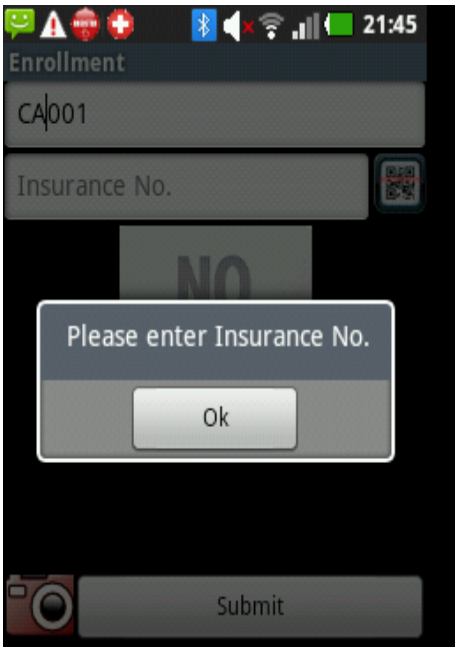
By clicking on 'Retry' Button, the current image will be discarded and application will be ready to take another image. By clicking on the 'Save' Button, the application will return to the previous screen with the taken image.

The function "`btnTakePhoto.setOnClickListener(new OnClickListener()`" is called to carry out this activity. This function calls the function "`startActivityForResult(intent, 0)`" with the request code 0.
Intent is the instruction object, passed to this procedure, for activating in this case the camera for taking a photograph.

After taking the photograph and clicking the 'Save' button, the following screen will be displayed.



Information can only be submitted if all fields are entered correctly. In case of missing mandatory fields, the user will be prompted to enter the required missing information. Refer the image below.



This task (of verifying entries) is carried out by the function "`protected boolean isValidate()`" which returns Boolean value if all the requirements are met, it will return false otherwise.

Once the information is entered and confirmed, the user has to press the 'Submit' Button to save the information. On clicking the 'submit' button, we will first construct the filename of the image file. This is important as the naming convention will be used by the web service to upload the image into the central server.

Naming convention:

<Insurance Number>_<Enrollment Assistant code>_<Date>.jpg

After saving successfully, the application will check for internet/mobile connectivity. In case we are connected, the file stored on the SD card will be sent to the designated FTP folder for enrollment files. After successful transmission, the file will be automatically deleted from the local SD card and a message will be displayed: "Image has been uploaded to the server successfully."
In case of failure the message "Error occurred while uploading the image to the server" will be displayed and the file will remain stored on the SD card.

The uploading task is implemented in the separate class named "UploadFile.java". The function "`public boolean uploadFileToServer(File file)`" is called along with the parameter File. This is the actual file to be uploaded (*.jpg)

If we were working on the phone without network coverage, the message "Image is saved on external storage device" will be shown after successful saving of the image.

The application has another utility which will upload all the images available on the SD Card (not yet submitted to the server) .
By executing this service, the application will go through all the images and will upload them one by one to the server and delete them from the SD Card on successful upload.

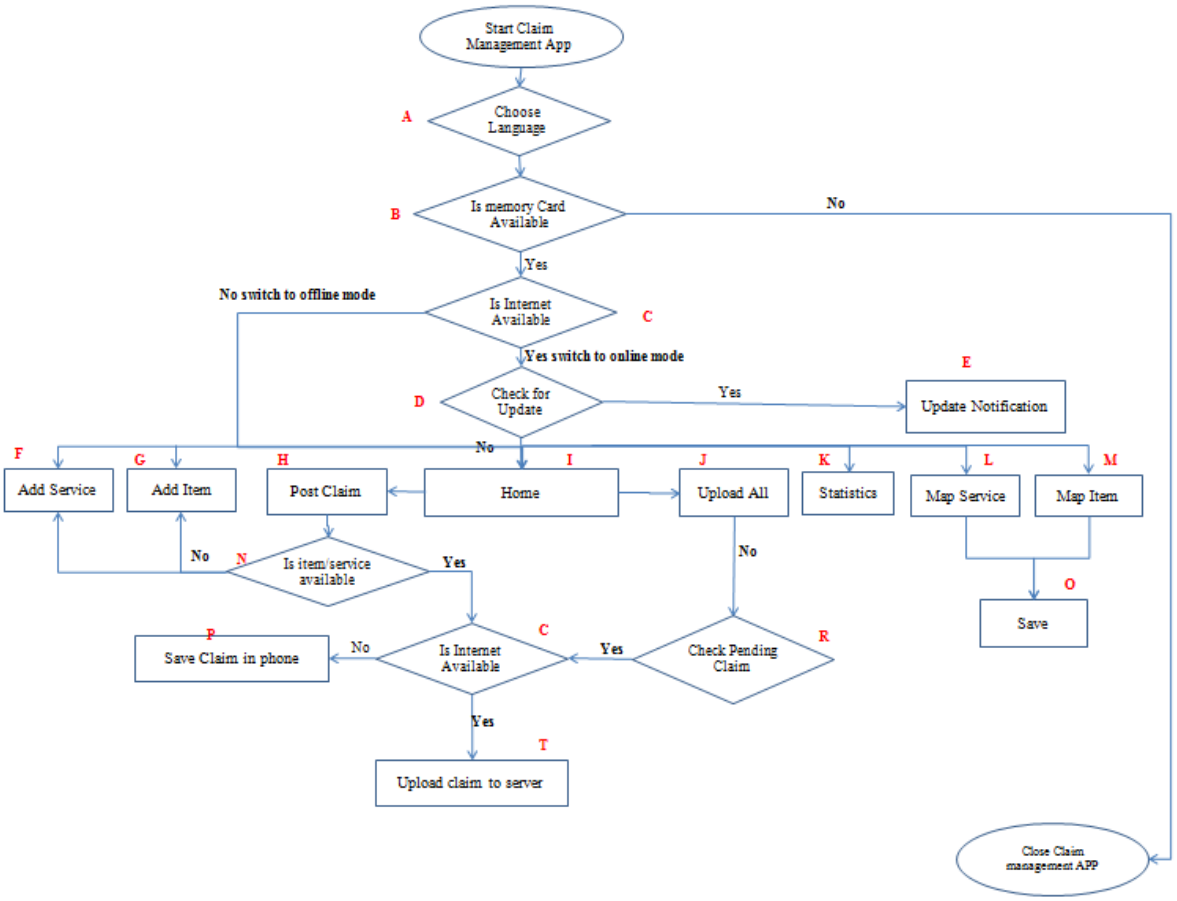
Once the data is saved or uploaded successfully, the application will be ready to enroll another insuree. All the input, except Officer, will be cleared.

This whole task is performed in the function "`btnSubmit.setOnClickListener(new OnClickListener()`"

8.3. Claim application

This application is used by the claim administrator for the purpose of entering and submits claims in their respective health facility.

8.3.1. Claim Flow chart diagram



In the code snippets below, the main functions of importance are shown in **bold**.

- The function for Choosing Language **(A)**

```
public void ChangeLanguage(Context ctx,String Language){
 Resources res = ctx.getResources();
 DisplayMetrics dm = res.getDisplayMetrics();
 android.content.res.Configuration config = res.getConfiguration();
 config.locale = new Locale(Language.toLowerCase());
 res.updateConfiguration(config, dm);
     }
```

- The function for checking for Memory Card **(B)**

```
public int isSDCardAvailable(){
String State = Environment.getExternalStorageState();
if (State.equals(Environment.MEDIA_MOUNTED_READ_ONLY)){
return 0;
        else if(!State.equals(Environment.MEDIA_MOUNTED)){
return -1;
        }else{
return 1;
        }
}
```

- The function for checking internet Availability **(C)**

```
public boolean isNetworkAvailable(Context ctx)
{
ConnectivityManager cm = (ConnectivityManager)
ctx.getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo ni = cm.getActiveNetworkInfo();

return (ni != null && ni.isConnected());

        }
```

- The function to check version availability**(D)**

```
  public Boolean isNewVersionAvailable(String Field,Context ctx, String PackageName)
  {
  String result;
CallSoap cs = new CallSoap();
cs.setFunctionName("GetCurrentVersion");
result = cs.GetCurrentVersion(Field);
if (result == "") return false;
return (!getVersion(ctx,PackageName).toString().equals(result));

}
```

- The **"Add Service"** button event - for adding services **(F)**

```
 btnAdd.setOnClickListener(new OnClickListener() {

  @Override
 public void onClick(View v) {
try {
 if(oService == null) return;
String Amount,Quantity = "1";

HashMap<String,String> lvService = new HashMap<String,String>();
lvService.put("Code", oService.get("Code"));
lvService.put("Name",oService.get("Name"));
Amount = etSAmount.getText().toString();
lvService.put("Price", Amount);
if(etSQuantity.getText().toString().length() == 0) Quantity = "1"; else Quantity =
etSQuantity.getText().toString();
lvService.put("Quantity", Quantity);
ClaimManagementActivity.lvServiceList.add(lvService);

alAdapter.notifyDataSetChanged();
                etServices.setText("");
                etSAmount.setText("");
                etSQuantity.setText("");

} catch (Exception e) {

Log.d("AddLvError", e.getMessage());
            }
        }
    });
```

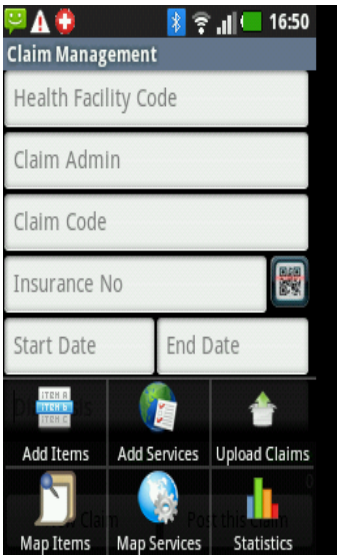
- The **"Post Claim"** button event **(T)**

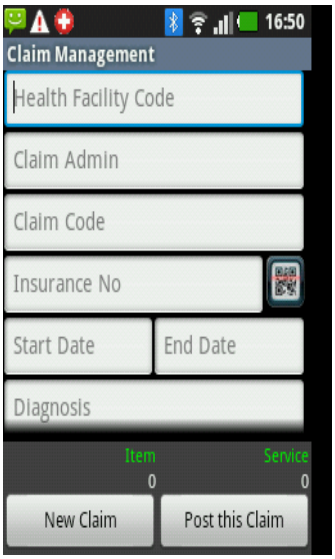
```
btnPost.setOnClickListener(new OnClickListener() {

@Override
public void onClick(View v) {
if(!isValidData())return;
```

```
WriteXML();
ClearForm();
        ShowDialog(getResources().getString(R.string.ClaimPosted));
        }
});
```

This application includes:

- Adding claims
- Adding services and medical items to claim
- Mapping services and item
- View Statistics





Add Claims:

All the activities related to this page is implemented in the class named "Claims.java"

By starting the application the user will be redirected to a page with 4 inputs.

1. Health Facility Code: This is a textbox. The user will have to enter the Health facility code. This is a mandatory field. This will be populated automatically if the user had used it before. The last used health facility code will be always saved in the memory.
2. Claim Code: This is a textbox. The user has to enter the claim code displayed on the claim form. This is a mandatory field.
3. Insurance Number: This is a textbox. The user will have to enter the Insurance Number of the insuree. This is a mandatory field. This filed can be filled by manually typing the code or by scanning.
4. Start Date: This will be a textbox. The user will have to select the start date of the treatment. On focus of this textbox a date dialog will appear. The user can navigate to the desire date. This is a mandatory field.
5. End Date: This will be a textbox. The user will have to select the end date of the treatment. On focus of this textbox a date dialog will appear. User can navigate to the desire date. Initially the value of this field will be the same as the start date, but the user is allowed to change it. This is a mandatory field.

Please find below a screen shot of the Date Dialog. By default these dialogs display the current date of your phone.

6. Disease: This will be a spinner (combo-box). The user will have to select the code of the disease (ICD code).

In this screen one will have 3 buttons.

1. Scan: This button is used to scan the QR code. This task is accomplished by the function "`btnScan.setOnClickListener(new OnClickListener()`"

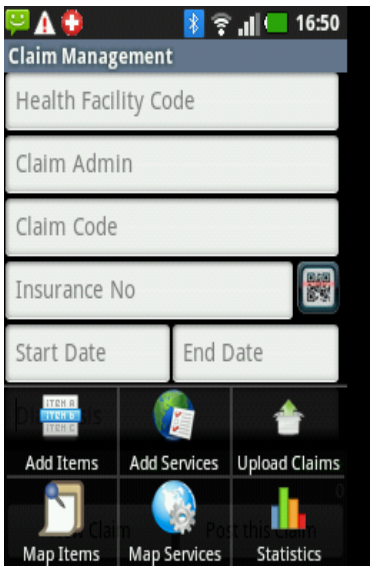
2. Post this claim: If all the required fields are entered, including Item or Service, both or any of them, then the application will save the data to the XML file. To validate, we will use the Boolean function named "private booleanIsValidate()" which will return true if all the conditions are fulfilled false otherwise.
   To save the data into the XML file application will use the function "`btnSave.setOnClickListener(new OnClickListener()`"

3. Add new claim: If the user presses this button, the application will prompt the user with the message "This claim is unsaved. Do you want to discard the claim and add a new claim?" If Yes button is pressed by the user then application will ignore all the changes made and clears the form. No action will be taken if the user selects No button.

Other buttons will be shown by clicking the default menu button of the page, generally located at the left side of the Home Button.

1. Add Items: By selecting this option, user will be redirected to the Add Item Activity.
2. Add Services: By selecting this option, user will be redirected to the Add Services Activity.
3. Upload all claims: By selecting this option, the user can upload all the claims to the server.
4. Map Item: By selecting this option, user will be redirected to the map item Activity.
5. Map Services: By selecting this option, user will be redirected to the map Services Activity
6. Statistics

Please find below a screen shot of the Add Claim screen with option menus shown.



Add items:

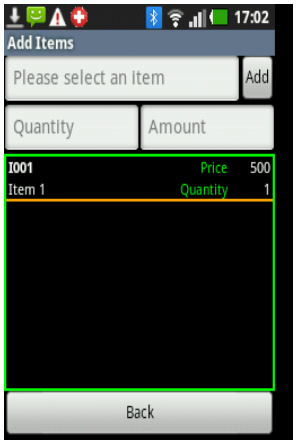
All the activities related to this page will be implemented in the class called "AddItems.java"
The user is redirected here if he/she has selected option menu named Add Items. This screen will have three inputs:

- Item Code (in the form of a spinner)
- Quantity
- Price

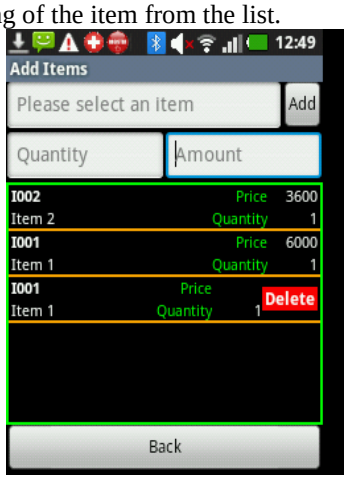
After selecting an item and entering the quantity and price, the user has to press the 'Add' button to add it to the list. Once the Item is selected Quantity and Price will be auto filled with the default quantity to 1 and default price of the item which can be changed before adding to the list.

All the added items will be added to the list view, which can be deleted later. After adding all the items to the list, the user can press the 'Back' button to go back to the previous page.

The application will use the function named "GetAllItems()" to populate dropdown list.
Once an item is selected and has the quantity and price set, the application will use the function `btnAdd.setOnClickListener(new OnClickListener()` " to add the item into the list view.



Even after adding the item to the list, if the user wishes to remove it from the list then it can be done by just pressing the desire item for a little bit longer. Once the item is pressed for a couple of seconds a delete button will appear and the user can remove the item from the list easily.
Below is the screen shot for the removing of the item from the list.



Add services:

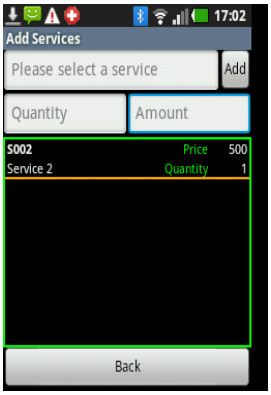
All the activities related to this page is implemented in the class called "AddService.java"
The user is redirected here if he/she has selected the menu option named Add Services. This screen will have three inputs:

- Service Code (in the form of a spinner)
- Quantity
- Price

After selecting the service and entering the quantity and price, the user has to press the 'Add' button to add it to the list. Once the service is selected Quantity and Price will be auto filled with the default quantity to 1 and default price of the service which can be changed before adding it to the list.

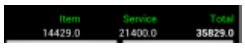
All the added services to the list view can be deleted later. After adding all the services to the list, the user can press the 'Back' button to go back to the previous page.

The application will use the function named "GetAllServices()" to populate dropdown list.
Once an item is selected and has the quantity and price set, the application will use the function `btnAdd.setOnClickListener(new OnClickListener()` " to add the service into the list view.
Below is the sample model screen.



Even after adding the service to the list, if the user wishes to remove it from the list then it can be done by just pressing the desire service for a little bit longer. Once the service is pressed for a couple of seconds a delete button will appear and the user can remove the service from the list easily.
Below is the screen shot for the removing of the item from the list.

Once the user presses the back button after adding Items or Services, he/she will be redirected to the main page of the application. And user can see the total amount of selected Items and the services individually and the total amount of the claim as displayed in the screen below.

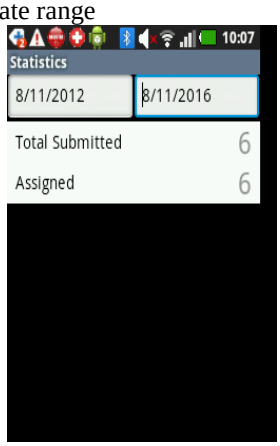


Once everything is confirmed, user can press Post This Claim button to save the data in XML file. Creating and saving XML file function can be found it ClaimManagement.java file under the function name called WriteXML(). Once the claim is posted all the fields will be cleared except health facility code. And user will be prompted with the message "**Claim is saved on phone memory successfully.**"

No matter if user is online or offline. In both the cases application will save the XML file to the external memory of the phone. Which can be uploaded all the files together at once just by selecting the option menu "**Upload all claims**". Once the claim is uploaded on the server, application will get acknowledgement from the server whether the claim has been accepted or rejected by the server. And according to the message the claim will be moved to the AcceptedClaims or RejectedClaims folder.

Statistics:

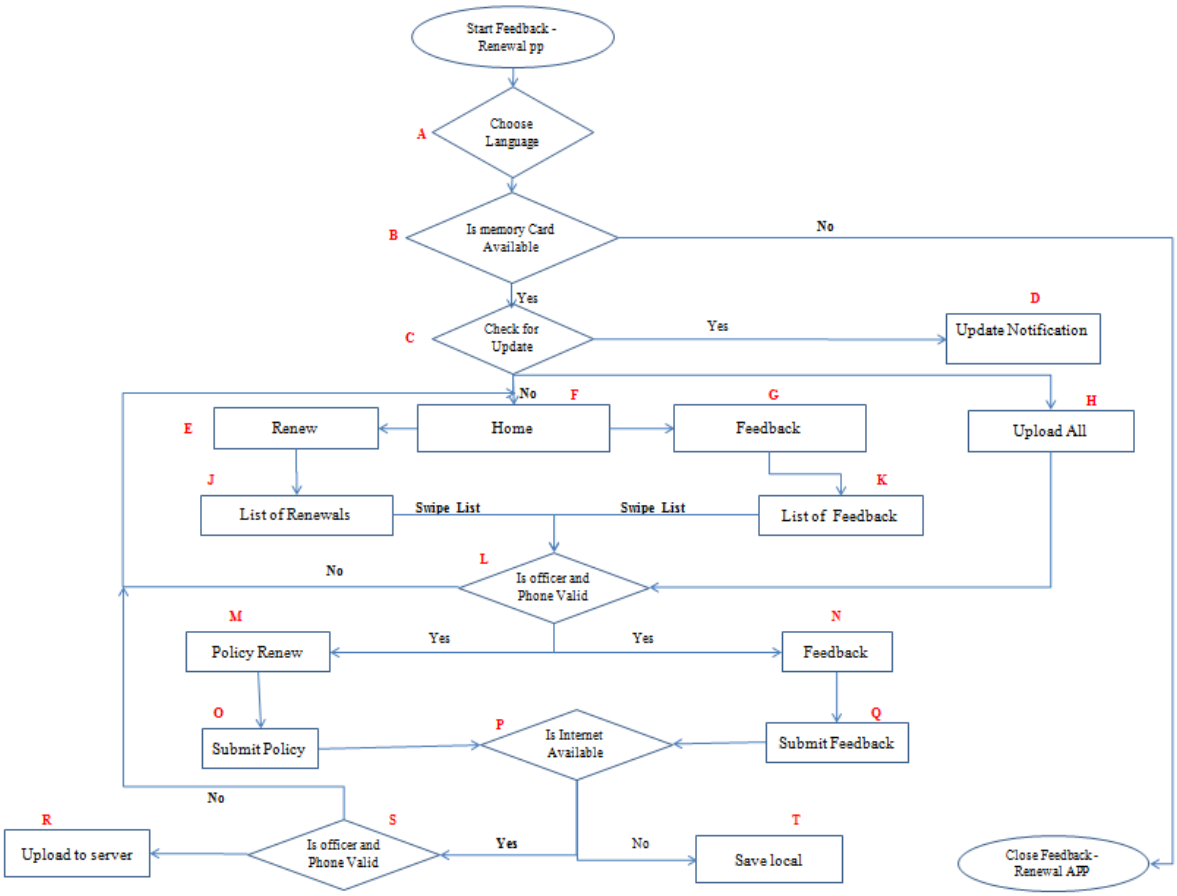
By selecting this option, user will be redirected to the statistics activity which will provide total submitted claim and also assigned claims according to the date range



8.4. Feedback & Renew application

This application will be used by the Enrollment Officers, the application home page allow user to put enrolment officer code which is related to the displayed number.

8.4.1. Feedback-Renewal Flow chart diagram



In the code snippets below, the main functions of importance are shown in **bold**.

- The function for Choosing Language **(A)**

```
public void ChangeLanguage(Context ctx,String Language){
 Resources res = ctx.getResources();
 DisplayMetrics dm = res.getDisplayMetrics();
 android.content.res.Configuration config = res.getConfiguration();
 config.locale = new Locale(Language.toLowerCase());
 res.updateConfiguration(config, dm);
      }
```

- The function for checking the Memory Card availability**(B)**

```
public int isSDCardAvailable(){
String State = Environment.getExternalStorageState();
if (State.equals(Environment.MEDIA_MOUNTED_READ_ONLY)){
return 0;
        else if(!State.equals(Environment.MEDIA_MOUNTED)){
return -1;
        }else{
return 1;
        }
}
```

- The function for checking internet Availability **(P)**

```
public boolean isNetworkAvailable(Context ctx)
{
ConnectivityManager cm = (ConnectivityManager)
ctx.getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo ni = cm.getActiveNetworkInfo();

return (ni != null && ni.isConnected());

  }
```

- The function for checking the new version availability **(C)**

```
public boolean isNewVersionAvailable(String Field,Context ctx, String PackageName)
{
      String result;
CallSoap cs = new CallSoap();
cs.setFunctionName("GetCurrentVersion");
result = cs.GetCurrentVersion(Field);
if (result == "") return false;
return (!getVersion(ctx,PackageName).toString().equals(result));

}
```

- Swipe event to list policy pending renewals **(J)**

```
swipe.setOnRefreshListener(new SwipeRefreshLayout.OnRefreshListener() {
    @Override
    public void onRefresh() {
       swipe.setRefreshing(true);
       (new Handler()).postDelayed(new Runnable() {
          @Override
          public void run() {
             try {
                RefreshRenewals();
                swipe.setRefreshing(false);
             } catch (IOException e) {
                e.printStackTrace();
             } catch (XmlPullParserException e) {
                e.printStackTrace();
             }
          }
       }, 3000);
```

```
        }
    });
```

- Swipe event to list claims which are selected for Feedback **(K)**

```
swipe.setOnRefreshListener(new SwipeRefreshLayout.OnRefreshListener() {
        @Override
        public void onRefresh() {
            swipe.setRefreshing(true);
            (new Handler()).postDelayed(new Runnable() {
                @Override
                public void run() {
                    try {
                        swipe.setRefreshing(false);
                        RefreshFeedbacks();
                    } catch (IOException e) {
                        e.printStackTrace();
                    } catch (XmlPullParserException e) {
                        e.printStackTrace();
                    }
                }
            }, 3000);
        }
    });
```

- The function for validating enrollment officer**(L,S)**

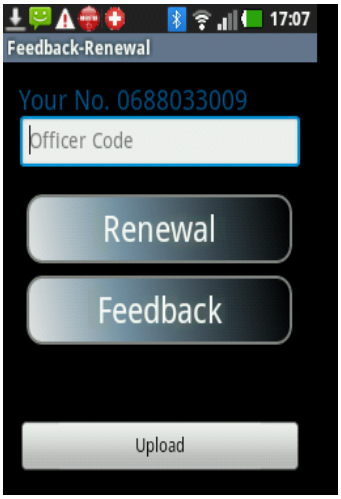
```
private boolean isValidPhone(){
    int result;
     CallSoap cs = new CallSoap();
     cs.setFunctionName("isValidPhone");
//       result = cs.isValidPhone(etOfficerCode.getText().toString(), UniqueId);
     result = cs.isValidPhone(etOfficerCode.getText().toString(), PhoneNumber);
     if (result == 0) {
          ShowDialog(getResources().getString(R.string.InvalidPhone) + " " +
          etOfficerCode.getText().toString());
         return false;
     } else if(result == 1) {
        return true;
     }else{
        ShowDialog(getResources().getString(R.string.ConnectionFail));
        return false;
     }
}
```

- The **"Submit"** button event - for uploading the renewal policy to the server **(R)**

```
btnSubmit.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {

            if (!chkDiscontinue.isChecked())
                if (isValidate() == false) return;

            pd = ProgressDialog.show(Renewal.this, "", getResources().getString(R.string.Uploading));

            new Thread() {
                public void run() {
                    WriteXML();

                    //Upload if internet is available
                    if (_General.isNetworkAvailable(Renewal.this)) {
                        if (!isValidPhone()) return;
                        UploadFile uf = new UploadFile();
```

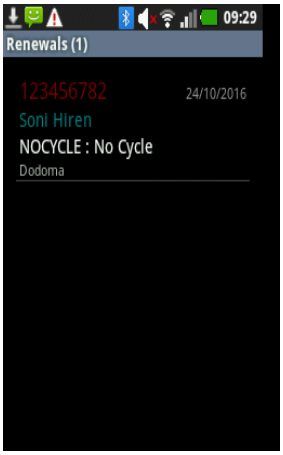
```
                if (uf.uploadFileToServer(Renewal.this, PolicyXML)) {
                    if (ServerResponse()) {
                        result = 1;
                    } else {
                        result = 2;
                    }
                } else {
                    result = 3;
                }
            } else {
                result = 3;
            }
            File file = PolicyXML;
            MoveFile(file);

            runOnUiThread(new Runnable() {

                @Override
                public void run() {
                    switch (result) {
                        case 1:
                            DeleteRow(RenewalId);
                            ShowDialog(getResources().getString(R.string.UploadedSuccessfully));
                            break;
                        case 2:
                            DeleteRow(RenewalId);
                            ShowDialog(getResources().getString(R.string.ServerRejected));
                            break;
                        case 3:
                            UpdateRow(RenewalId);
                            ShowDialog(getResources().getString(R.string.SavedOnSDCard));
                            break;
                    }
                    //Go back to the previous activity.
                    finish();
                }
            });

            pd.dismiss();
        }
    }.start();

  }
});
```

All the activities related to Feedback page is implemented in the class called "Feedback.java" and activities related to renew page is implemented in the class called "Renew.java". By click either of the two buttons the application will list all Renewals for renew buttons and all claims require feedback for feedback buttons.

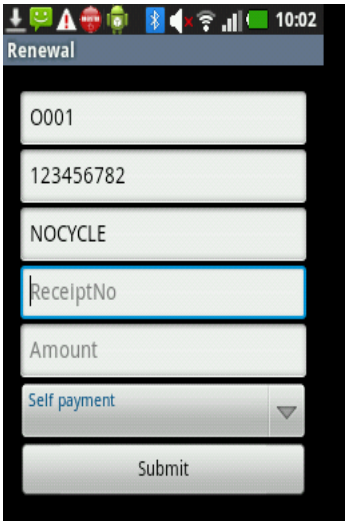


Here is a brief description of each input field and submit button.

1. Officer Code: This will be a textbox. In this field the user will enter his/her assigned code. This will be a mandatory field.
2. If user clicks feedback after put the officer code, then the application will display all Claims which require feedback.



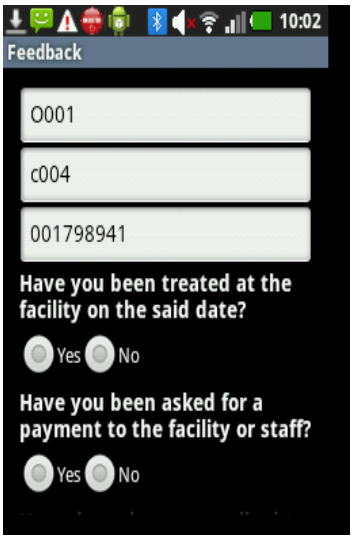
3. User will select the policy he wants to renew the fill the data as shown in figure bellow.

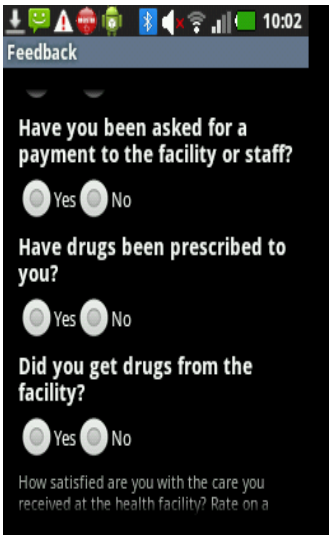


Receipt number is a mandatory field which needs to be entered manually.
The amount is also a mandatory field. This field accepts only numbers. The user has to put the amount collected from the insuree.

Information can only be submitted if all fields are entered correctly. In case of missing mandatory fields, the user will be prompted to enter the required missing information.

4. If user click Renew then the application will display all policy require feedback or renew





5. For feedback there will be a few questions with Yes/No options. Here is a list of the questions.
   - Has been the claimed care actually rendered?
   - Has a payment been asked?
   - Has been prescribed a drug?
   - Has been received the prescribed drug?
6. Here we will give 5 stars to rate the overall satisfaction of the insuree. User can select from 0-5 rating.

.
This task will be carried out by the function
"`btnSave.setOnClickListener(new OnClickListener()`"
Once the data is saved on the external storage device, the application will check for the availability of the internet connection.

This checking of connectivity will be implemented in the class called "General.java". This class will have a boolean function called "public booleanisNetworkAvailable()" which will return true if connectivity is available; false otherwise.

If the internet connection is available then the XML file will be uploaded to the remote FTP server and the original file will be deleted from the external storage device of the phone. In this case the user will be notified with the message "*Feedback has been uploaded to the server successfully.*"

Otherwise, (no connectivity) the user will be notified with the message "*Feedback is saved on external storage device.*" In this case no deletion task will be performed and the user can manually upload all the feedback to the server later.

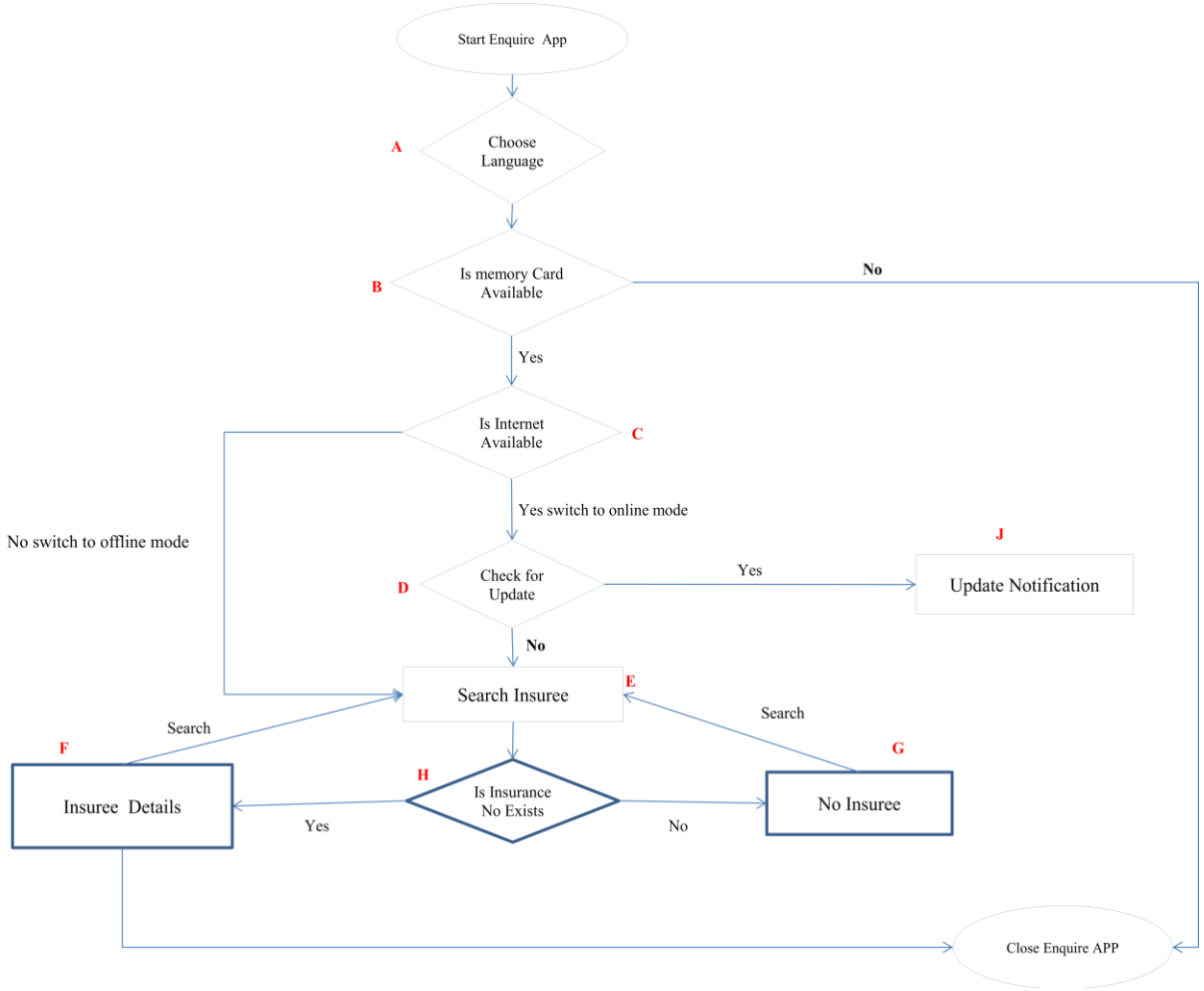
This task will be implemented in the class called "UploadFile.java" which will have a function named "UploadFileToServer(File file)". This function will accept a file to be uploaded as a parameter. In this case it will be an XML file.

Once the save function is executed successfully, the application will clear all the inputs and the user will be ready for the next feedback.

8.5. Policy Inquiring Application

This application will be used extensively by dispensaries, health facilities and hospitals for verifying on policies.

8.5.1. Inquiring Flow chart diagram



In the code snippets below, the main functions of importance are shown in **bold**.

- The function for Choosing Language **(A)**

```
public void ChangeLanguage(Context ctx,String Language){
Resources res = ctx.getResources();
DisplayMetrics dm = res.getDisplayMetrics();
android.content.res.Configuration config = res.getConfiguration();
config.locale = new Locale(Language.toLowerCase());
res.updateConfiguration(config, dm);
}
```

- The function for checking the Memory Card availability**(B)**

```
public int isSDCardAvailable(){
String State = Environment.getExternalStorageState();
if (State.equals(Environment.MEDIA_MOUNTED_READ_ONLY)){
return 0;
else if(!State.equals(Environment.MEDIA_MOUNTED)){
return -1;
}else{
return 1;
}
}
```

- The function for checking the internet Availability **(C)**

```
public boolean isNetworkAvailable(Context ctx)
        {
        ConnectivityManager cm = (ConnectivityManager)
        ctx.getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo ni = cm.getActiveNetworkInfo();

return (ni != null && ni.isConnected());

}
```

- Function to check for new version availability**(D)**

```
        public boolean isNewVersionAvailable(String Field,Context ctx, String PackageName)
        {
        String result;
    CallSoap cs = new CallSoap();
    cs.setFunctionName("GetCurrentVersion");
    result = cs.GetCurrentVersion(Field);
    if (result == "") return false;
    return (!getVersion(ctx,PackageName).toString().equals(result));

    }
```

- The **"Go"** button event - for searching Insuree details **(E)**

```
        btnGo.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v) {
            InputMethodManager inputManager = (InputMethodManager)
            getSystemService(Context.INPUT_METHOD_SERVICE);

            inputManager.hideSoftInputFromWindow(getCurrentFocus().getWindowToken(),
            InputMethodManager.HIDE_NOT_ALWAYS);

            ClearForm();
            if (!CheckCHFID()) return;
            pd = ProgressDialog.show(EnquireActivity.this, "",
            getResources().getString(R.string.GetingInsuuree));
                        new Thread() {
            public void run() {
            getInsureeInfo();
            pd.dismiss();
            }
                    }.start();
            }
        });
```

- The function to validate if Insurance number is correct **(H)**

```
        private boolean CheckCHFID(){
        if (etCHFID.getText().length() == 0){
        ShowDialog(tvCHFID, getResources().getString(R.string.MissingCHFID));
        return false;       }
        if (!isValidCHFID()){
        ShowDialog(etCHFID,getResources().getString(R.string.InvalidCHFID));
        return false;
        }
        return true;
        }
```

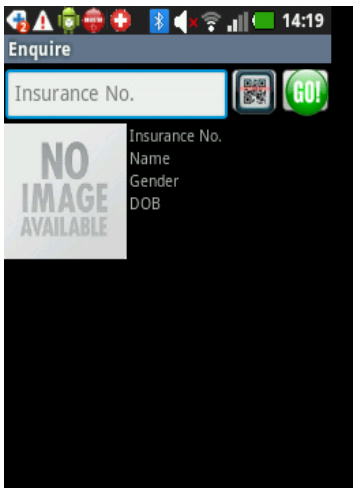
- Dialog box to show that the insuree does not exist **(G)**

```
protected AlertDialog ShowDialog(final TextView tv,String msg){
return new AlertDialog.Builder(this).setMessage(msg) .setCancelable(false)
.setPositiveButton("Ok", new android.content.DialogInterface.OnClickListener()
{
@Override
        public void onClick(DialogInterface dialog, int which) {
        tv.requestFocus();
        }
        }).show();
}
```

- The function to return insuree Details **(F)**

```
private String getDataFromDb(String chfid)
{
result = "[{";
db = openOrCreateDatabase(Path +"ImisData.db3",SQLiteDatabase.OPEN_READONLY, null);

String[] columns = {"CHFID" ,"Photo" , "InsureeName",
"DOB", "Gender","ProductCode", "ProductName", "ExpiryDate", "Status", "DedType", "Ded1",
"Ded2", "Ceiling1", "Ceiling2"};
Cursor c = db.query("tblPolicyInquiry", columns, "CHFID=" + "\'"+ chfid +"\'" , null, null, null, null);
}
```

The user can enquire about the policy status of an insuree by providing the Insurance Number.
All the activities on this page are implemented in the class called "EnquireActivity.java" Below is the sample screen of the application.



A user can enter the Insurance Number manually or scan it by clicking on the 'Scan' button. By clicking on the **'Scan'** button, the application will launch the phone's inbuilt camera to scan the code. Once the code is scanned successfully, the application will return to the main screen with the Insurance Number and will start searching for the information of the insuree.
If Insurance Number is entered manually then user has to press '**Go'** button to fetch the information.

The before mentioned task of scanning is performed by the function called "`btnScan.setOnClickListener(new OnClickListener()` which loads the camera and on successfully scan calls the function "`startActivityForResult(intent, 1)`" with the request code 1.

As mentioned earlier this Enquiry application operates offline and online.
In case the phone has connectivity, the Insurance Number will be sent to the remote server where a web service will be running. The web service will provide the phone with the required information.

In this case, the application will use JSON technology with a POST method to execute the web service. Once the data is fetched from the live server, all the required information like Last name, Other names, Date of birth, Gender and policy and products of the insuree will be displayed in the above screen.

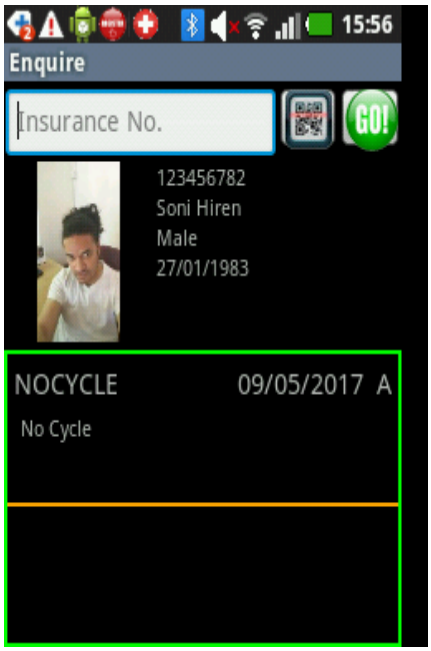
In case we are off line, the Inquiry will be launched on the locally stored SQLite file with Insuree information rather than sending a request to the central server.
The SQLite file in the phone will be created from the Central database via a separate functionality (to be further defined later) and could be created based upon district.
In offline mode, application will start looking for the information in this SQLite file rather than online server.

The task of retrieving Insuree information of the locally stored SQLite file is performed by the function "`publicvoidFetchInformation(String NSHIP NUMBER)`". This function accepts a string parameter named Insurance Number.

There are 3 ways to load the SQLite file in the phone:

- Transfer via laptop (USB cable, Bluetooth)
- Sending via e-mail as an attachment

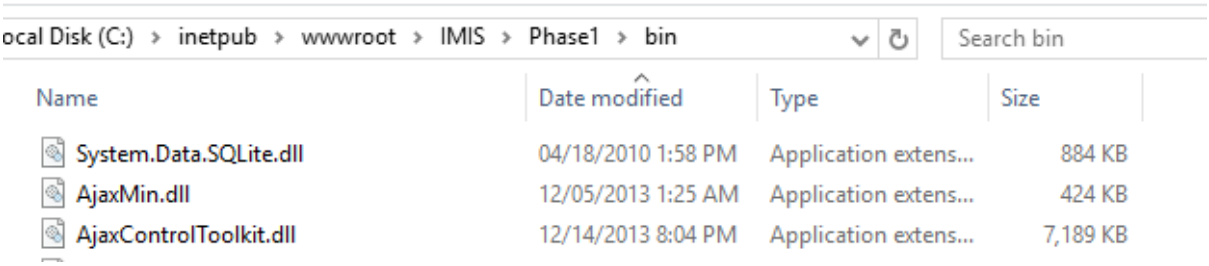
If the size of the SQLite file is very big, it is recommended that the user uses manual transfer and not attempting to send to the file via a mail attachment. This file will contain all the information about the each insuree including their photographs.
Below is the screen on successful search.



Once the result is found, the Insurance Number textbox will be cleared, so that the user can enter another Insurance Number enquire.

9. Library

The IMIS application uses the normal windows libraries, but 3 other additional libraries are required.
The figure below shows the 3 additional libraries:



Other third party applications used by IMIS includes:

- WinRar - used for zipping the offline extracts.
- Sqlite - used for creating .db3 Database, used by mobile applications.