**GW DEMO**

**Claim Process in three ways:**

**Claim- Centric**

**Collaborative**

**Controlled.**

**Notes**:

It is a Browser Based Application.

It supports multiple types of users.

Customer Service

Adjusters

Supervisors

**Claim**:

1. **Policy**
2. **Activities**
3. **Parties**
4. **Documents**
5. **Financials**
6. **Notes**
7. **Litigation**.

**ContactManager**

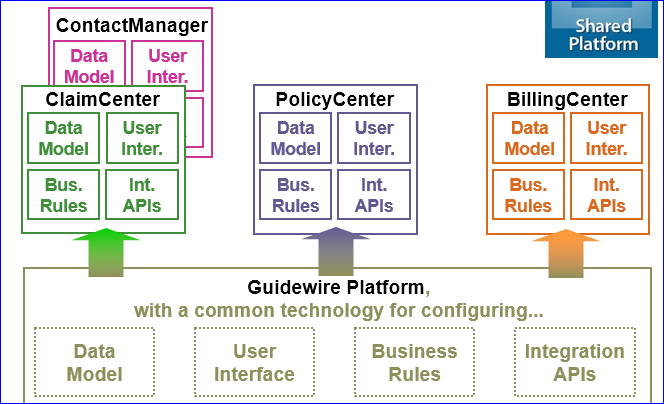
**ClaimCenter** **PolicyCenter** **BillingCenter**

**Guidewire Platform**

**Policy Center: It is an Application Designed to Issue, Modify and maintain Data about policies**

**Billing Center: It is an Application Designed to issue bills and track the premium payments for policies.**

**ClaimCenter: It is a Application designed to manage the process of reporting, verifying and making payments on claims against a policy.**

****

**External Systems connected to the ClaimCenter:**

**Policy Admin,**

**Address Book**

**Authentications**

**Check Printing**

**Reporting**

**………**

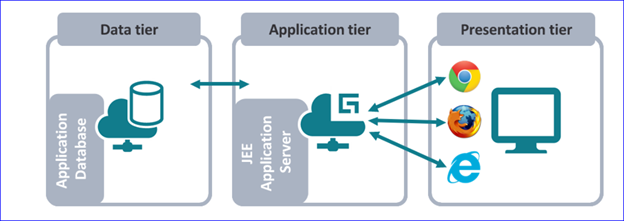
**Objectives:**

**I will describe the Product architecture.**

**Primary Components for configuring GW products.**

**Different types tiers in the GW CC.**

**The Three- tier Architecture:**



**Data Tier:**

Contains the business and Operational data.

The data is stored in a relational database which is typically hosted on a machine other than the application.

Guidewire supports below Databases.

Oracle Enterprise.

Microsoft SQL Server.

H2 for training and development.

**Application Tier**:

Contains the functional process and business logic.

The application logic is written in the GOSU.

Guidewire supports:

WebSphere

WebLogic

**Apache Tomcat**

JBoss EAP

Jetty for training and Dev Environments.

**Presentation tier:**

It contains the UI

Each application generators a collection of standard HTML pages that are rendered by the browser.

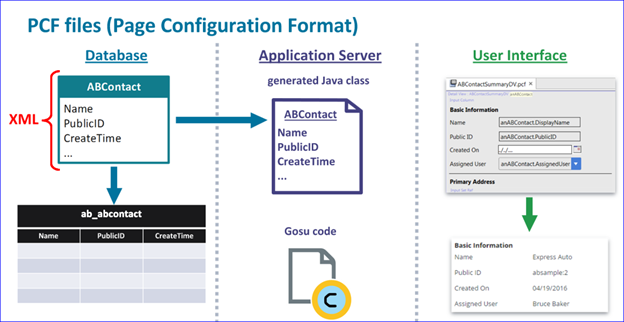
Guidewire supports:

Chrome

Firefox

MS IE

MS Edge.

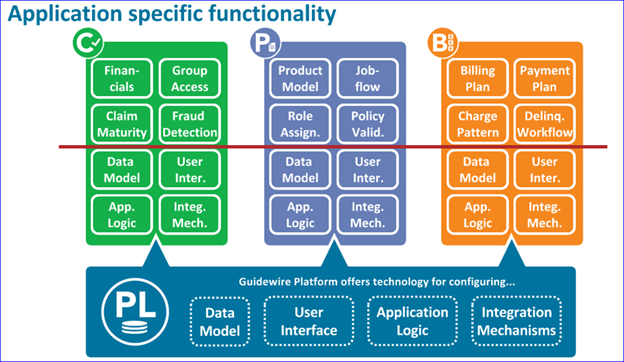


The Guidewire platform:

It contains common resourses.

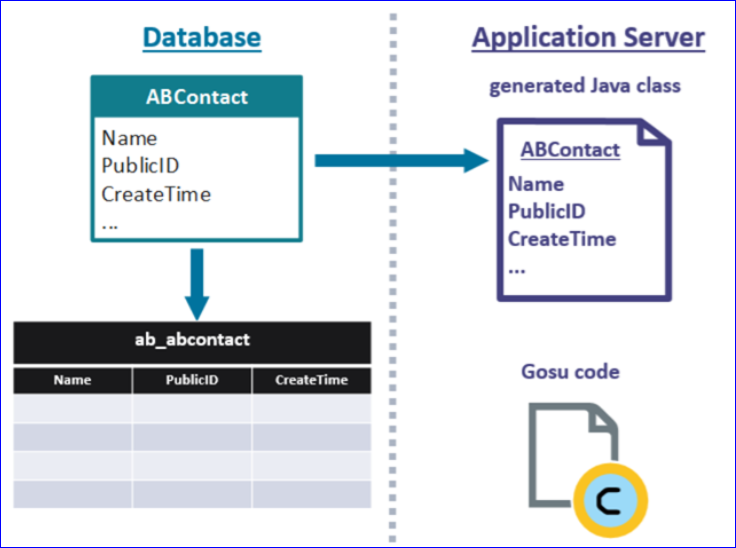
DataModel entities, PCF files and application logic to manage users.

DataModel Enttites and PCF files to create and assign activities to other users.



**DataModel**

GW DataModel is a set of XML formatted metadata definitions of entities and typelists:



DataModel Entities:

Entities are the high level business objects used by GW app, such as ABContact, Policy, Account, Invoice and Claim.

**ABContact → Entity → TableName Datatype**

Name ----------------------------------------------- varchar

Email Address-------------------------------------

Createtime----------------------------------------- datatime

Score ----------------------------------------------- integer

**AssignedUser ------------------------------------- TBD**

**Contact Tier ---------------------------------------- TBD**

Column --? It is a data field that stores a single value that does not reference any other object or table.

**Syntax: <coloumn>**

**Foreign Key fields:**

This field stores a reference to a related object in the data model.

**ABContact User**

Name Credential

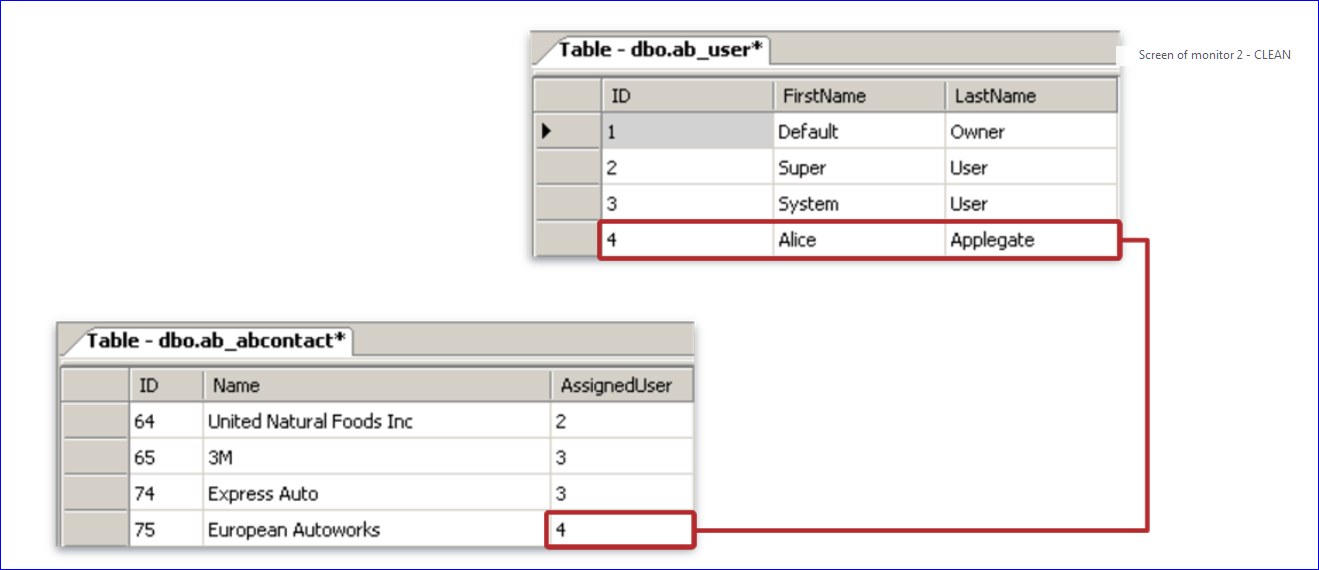
Email Address UserName

Createtime Roles

Score

**AssignedUser**

Contact Tier



**Array Fields:**

**ABContact ContactNote**

Name Subject

Email Address Body

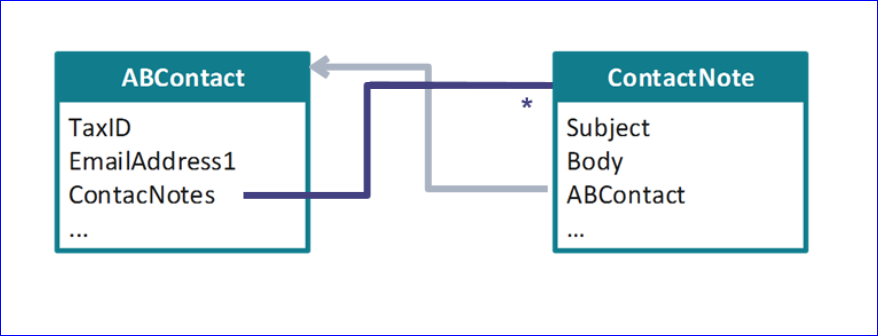
Createtime ABContact

Score

AssignedUser

Contact Tier

**ContactNotes**

****

**TypeLists and Typekey fields:**

**ABContact ContactTier (Typelist)**

Name Sliver

Email Address Gold

Createtime Platinum

Score Unknown

AssignedUser

**Contact Tier ---> TypekeyField.**

ContactNotes

**Commands**:

To start the Studio: **gwb studio**

To Open the Gradle: **gwb --gui**

To Start the Server: **gwb runServer**

URL To open the Training APP**:** [**http://localhost:8880/ab**](http://localhost:8880/ab)

**Entity under *metadata* will be with the .eti file type**

**Typelists under *metadata* will be with the .tti file type**

**Entity under *Extensions* folder will contains the file types of .eti and .etx**

**Typelists under *Extensions* folder will contains the file types of .tti and .ttx**

**Entity Requirement:**

Add the fields to ABContact.etx to capture additional details:

**Field Name**  **DataType**

1. WebAddress1\_Ext Varchar
2. FraudInvestigationNum1\_Ext Integer
3. LastCourtesyContact 1\_Ext Date
4. CustomerRating1\_Ext Decimal
5. IsStrategicPartner1\_Ext Boolean

Typelists Requirement:

You need to extend the CustomerServiceTier Typelist and add three new fields:

1. Platinum\_Ext
2. Gold\_Ext
3. Silver\_Ext

Local Machine Configuration:

Java

ApacheTomCat

SQL

How do we setup the TraningApp?

**Supertype (Subtypes) Entities**:

If there are multiple entities have a **common set of fields** then they could be organized into a hierarchy.

If the entities are in the same hierarchy, they will be using the concept of inheritance.

All the fields and methods of the supertype entities are inherited by subtype entities.

**ABContact**

**ABPerson ABCompany**

**ABPersonVendor ABPolicyPerson ABAdj**

**ABAttorney ABDoctor**

**ABContact**

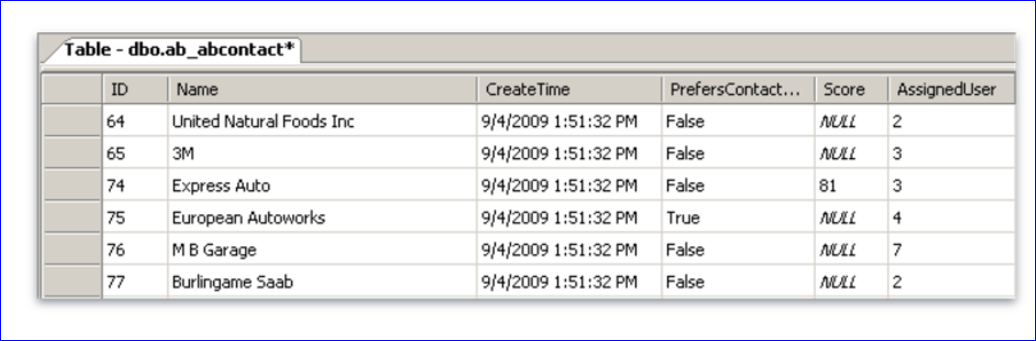
TaxID(SSN)

**EmailAddress**

**ABPerson** **ABCompany**

First Name Business Name

LastName

****

**Data Fields in the database:** Physical and Virtual

Data Model Defines physical fields as a columns in the database table.

**Gosu Code** defines the virtual fields and there is no physical database column.

**ABPerson**

**First Name**

**Middle Name**

**Last Name**

Full Name (**Virtual Field**)

**Gosu Code == First Name ++ Middle Name ++ Last Name**

**Data Dictionary**

The **DD** documents the entities and typelists in the guidewire application, Including both **base application** and **customer extension**.

To open the DD: …\build\dictionary\data\index.html → Recommended web browser.

**User Interface Architecture**

**→**  User Interface Architecture for GW Applications.

→ Diff kinds of PCF’s Files.

→ How to open and Edits PCF’s

→ Deploy PCF Files.

GW Application UI Framework.

**PCF Element**

**Widget** **Location**

→ **Widget**? It is a diaplayable element of the UI rendered into HTML.

→**Location?**It is navigable places in the application that a user or the application it self can navigate to.

**PCF**: Page Configuration Format. Is a object model in all the proprietary app framework used to create all guidewire end-user interfaces.

**PCF Element**

**Widget Location**

**Atomic Widget**

1. **Input**
2. **Cell**
3. **Button**

**Atomic Widget? →** these are the smallest building blocks of the UI.

Displays Individual data value

Executes individual action.

**PCF Element**

**Widget Location**

**Atomic Widget Container Widget**

1. **Input 1. Screen**
2. **Cell 2. List Detail Panel**
3. **Button 3. Card View Panel**

**4. Detail View Panel**

**5. List view Panel**

**6. Input Set**

**Primary Container and they are reusable to view that organizes the atomic widgets.**

**Secondary containers organizes the primary containers.**

**Screen? Is a top level container that organizes Secondary and Primary Containers.**

***Container widget*? Is a collection of atomic widgets and other container widgets. It organizes data and functionality into logical groups.**

**Inputset:**

* + **Input sets fit some attributes of the primary containers.**
  + **They can contain atomic widgets and organize them into logical groups.**

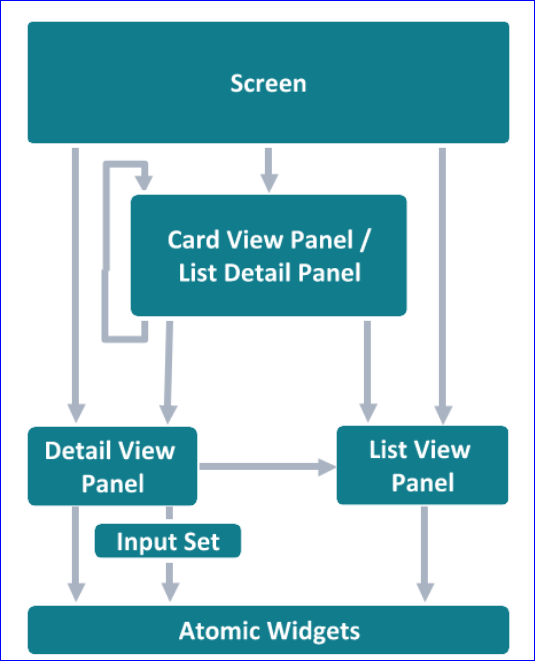
**BUT:**

**Input Sets can be Embeded into DV Panels.**

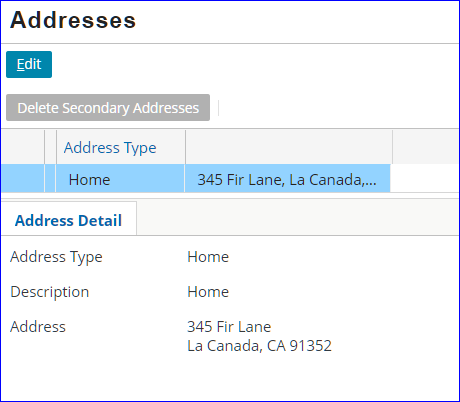
**Cannot be referenced by the Secondary views.**

**Cannot have a toolbar directly associated with them.**

**Container widget hierarchy:**

****

**List Detail View Panel**

****

**Locations**

**Location is a PCF where a User can navigate to.**

**Location can use the system permissions to control what user can see/access in the application.**

**PCF Element**

**Widget Location**

**Atomic Widget Container Widget 1.Location Group**

1. **Input 1. Screen 2.Page**
2. **Cell 2. List Detail Panel 3.Wizard**
3. **Button 3. Card View Panel 4.Popup**

**4. Detail View Panel 5.WorkSheet**

**5. List view Panel 6.Forward**

**6. Input Set 7.ExitPoint**

**Commands for PCF’s**

1. **To Open the Location info window**

**ALT + SHIFT + I**

**2. To Open a widget Inspector window**

**ALT + SHIFT + W**

**3. To Open the Page in the GW Studio**

**ALT + SHIFT + E**

**4. To Deploy the PCF**

**ALT + SHIFT + L**

**Atomic Widgets**

Atomic Widgets ia graphical user interface element that is no-Divisible.

**Input** and **Cell** widgets bind data values

Data Values often associated with the root objects, Query Objects or related objects.

DOT Notation is a GOSU Language syntax used to identify data objects and properties.

Dot Notation syntax

Referencing data field on given object

Object.fied

Example: We need to display Faxphone.

ABContact.FaxPhone

Referencing field at subtype level

(Object as subtype).field

(ABContact as ABPerson).DateofBirth

**DetailViews**

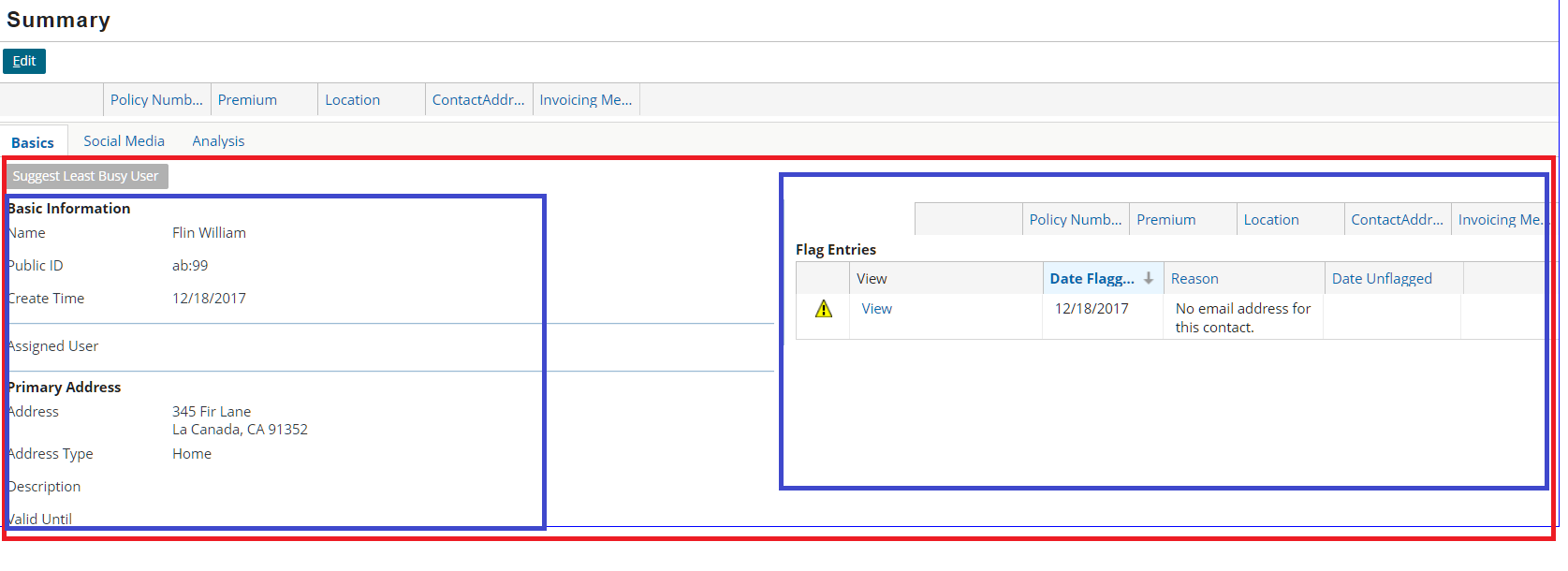
1. Functionality of Detail Views
2. Create a new Detail View
3. Reference a Detail View from the parent container
4. How to make detail view editable
5. Tollbar will be configured with the edit buttons.

* They are the primary containers and it organizes the widgets.
* Allows the users to
  + View the existing record
  + Create new record
  + Edit a record

Input columns help organize atomic widgets to display a related information.

Screens and Secondary view can reference.

Detail view Must at least contain a column and can contain as many columns as necessary to the user in the most effective way.

****

**Two types of detail views in the studio**

* **Detail view widget.**
* **Detail view PCF File.**

**Detail View Widget Detail View PCF file**

Not Reusable Reusable

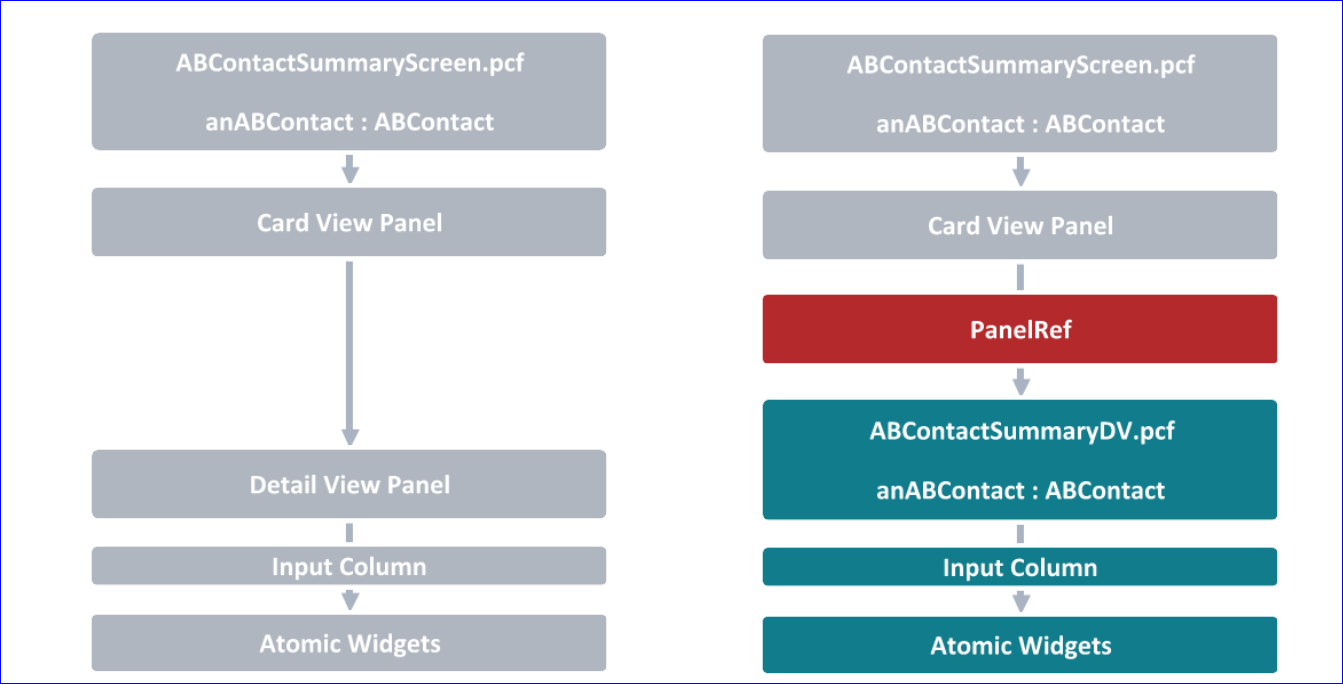
Inherits parent root object Takes a root object

Created as a DV widget in the Created as a Seperate

Parent Container PCF File PCF’s File the Filename

Ends with **DV**

**Similarities** :It can exercise and control Over all elements.



**Toolbar:**

It is a horizontal bar containing buttons and other useful widgets. It can be used at the top (Reflected at the bottom) of a screen or before a Panel.

It Usually contains Edit Buttons.

Toolbar can be directly added to a Screen, Panel Ref or Listview Input

Toolbars can also be associated with the Detail View Panels, Card view Panels and List Details Panels and List View Panel. However there is no direct placement or reference with the input set.

**Locations**

How can we navigate in the user interface

Diff types of locations.

Modifying widgets and navigating to the given locations.

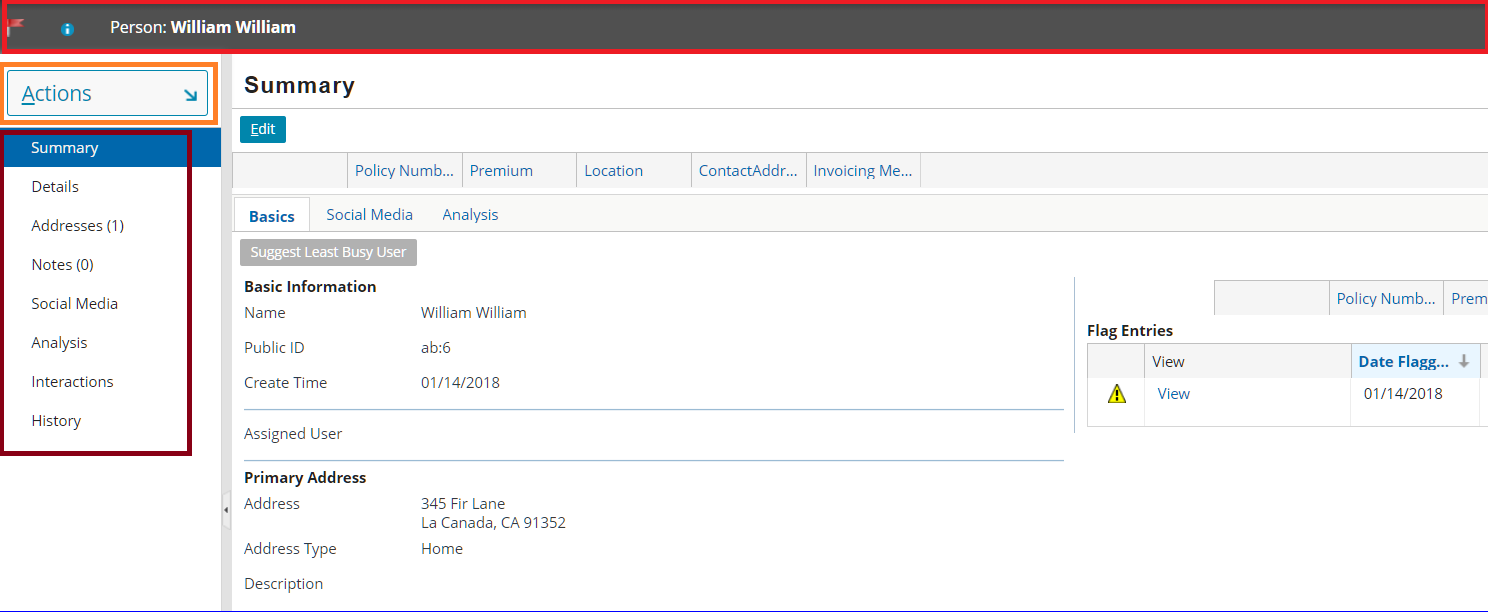
Location is a PCF element that user can navigate to diff types of PCF’s

**1.Page**: A page contains a single Screen.

Pages are used exclusively within location groups.

**2.Location Group**:

Location Group is a collection of pages used to view or modify data.



**3.Wizard**:

Wizard is an ordered collection of Screens used to execute a complex business process.

**4.Popups**:

A popup contains a single screen and returns the user to the previous screen or location once the popup is closed.

**5.WorkSheet:**

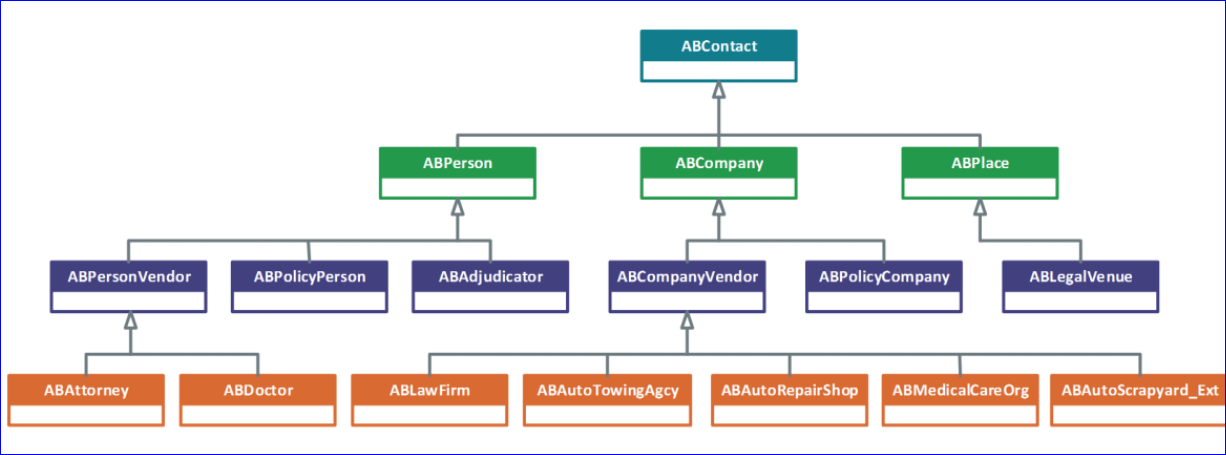
Worksheet contains a single screen rendered in the workspace frame.

**6.Forward:**

It contains a logic to execute before navigating to another location.

**7.Exit Points:**

It points to the URL outside of the guidewire application.



**Entry Point Syntax**

ABPersonDetailsPage(anABContact : ABContact)

**Action Syntax:**

**ABCompanyDetailsPage.*go*((anABContact as ABPerson).Employer)**

**DestinationPCF**:ABContactCompanyDV

**SourcePCF**: ABContactDetailPersonDV

**ListViews**

Functionalities of the LV

Creating a new List View

Create and modify a row iterator, row and cell widgets

Referencing a list view from the parent container.

List view panel is a Container widget that allows the user to view info about multiple objects at same time.

It is often referred as LV

User has ability to edit the data in the LV

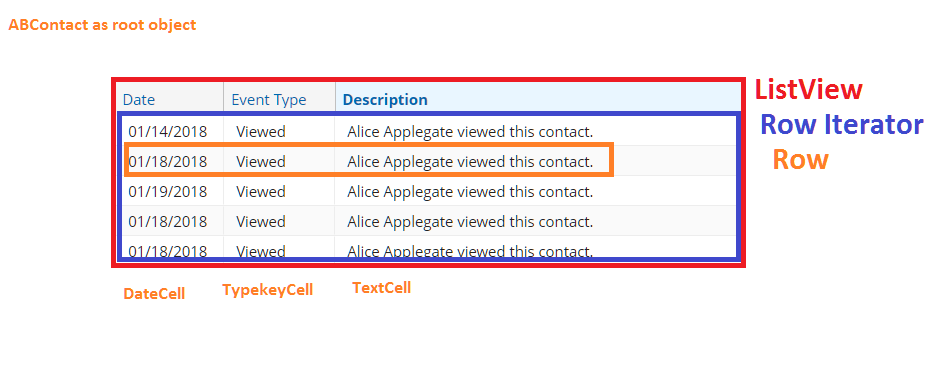
Structure of the List View:

LV it takes a single root obj that has a array of entities to process.

Row iterator processes the related array and produces a row for each entity instance.

Row contains one or more cell widgets

Cell Displays an individual field of an entity.



Row iterator:

There 4 required properties:

**Value** : anABContact.HistoryEntries

**valueType**: HistoryEntry[]

**elementName**: currenthistoryEntry → points to the obj in the array which is associated with the row.

**Editable:** false -- whether the cells can be edited or not.

**LV is divided into two types:**

**List view widget Listview PCF File**

**Differences**

Not reusable reusable

Inherits parent root obj Takes a root obj

Created as a LV widget in the parent Created as a separate

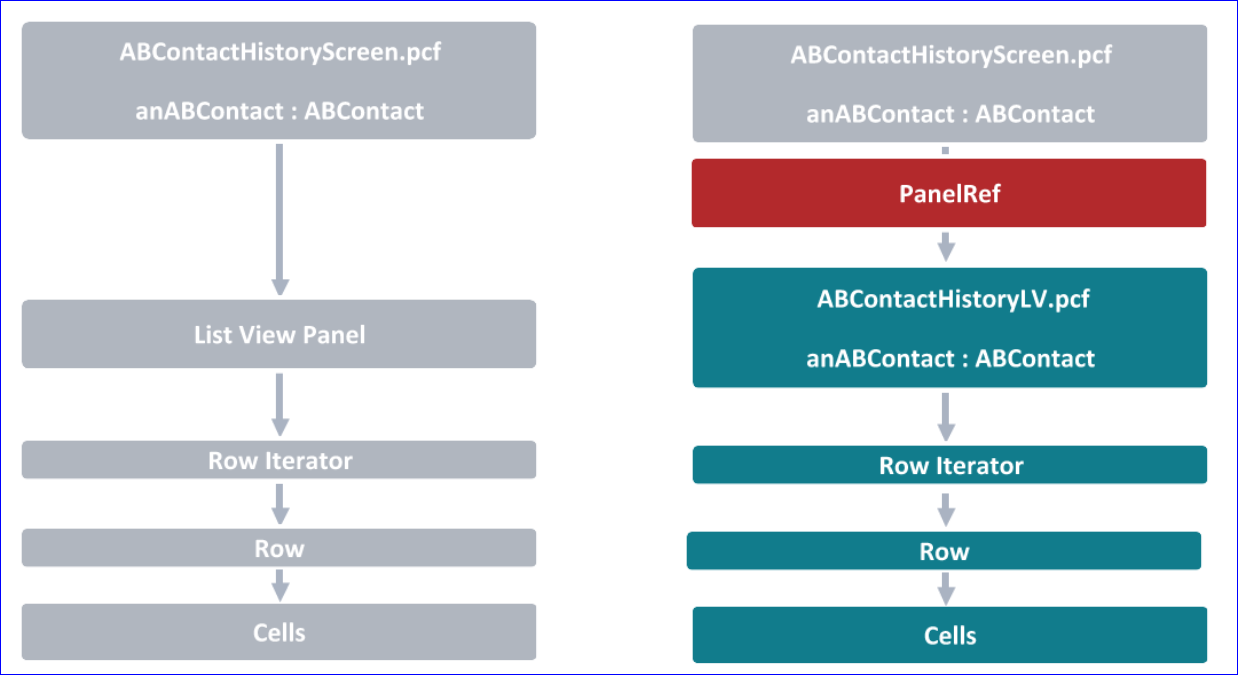
Container PCF file PCF file and file name

Ends with “LV”

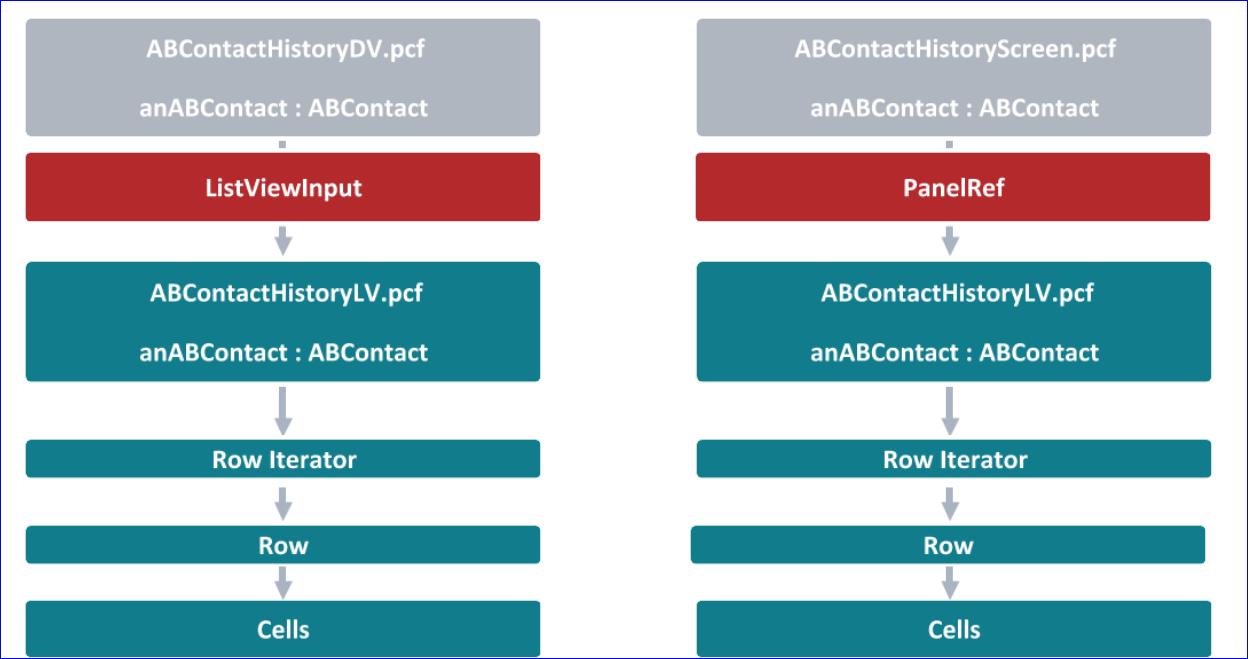
**Similarities**

It can exercise control over all elements.

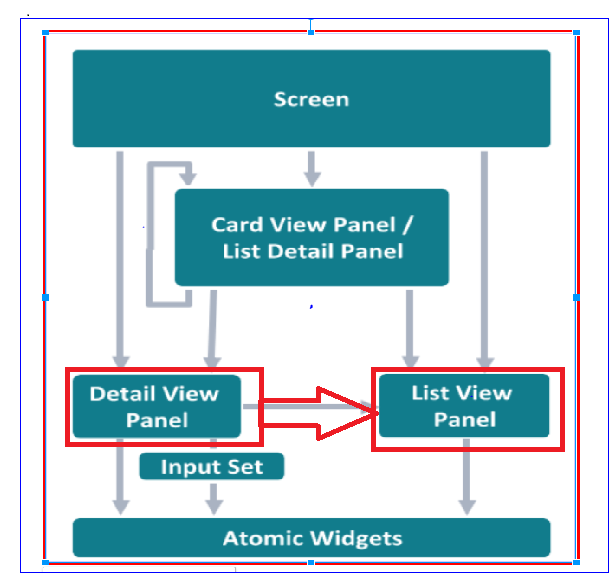
Inline Widget PCF File



Listview Input PanelRef



For the Listview Input refer the below Snippet:

****

**Toolbar and Pagination**

A toolbar is horizontal bar containing buttons and other useful widgets.

It can be added directly in the Screen, PanelRef and ListviewInput

Every LV Panel must have associated toolbar.

Application will automatically displays buttons to support the pagination.

Creating an Entity:

Make sure the new entity is Exportable

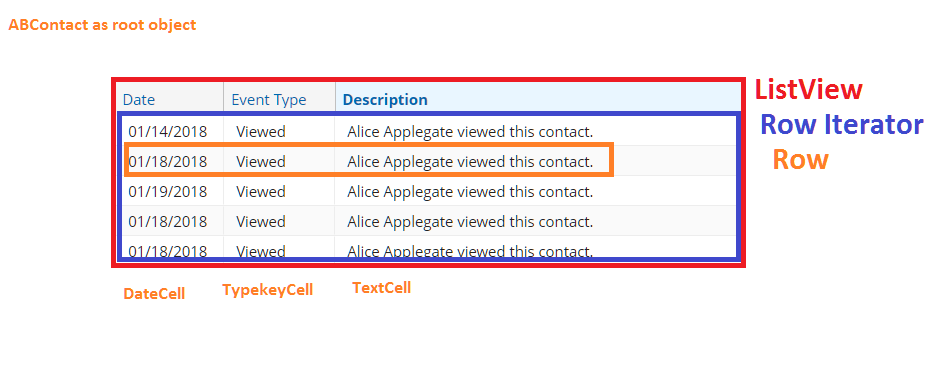
Make sure you have foriegn key associated to the ABContact Entity.

Make sure you have the Arraykey from ABConact Entity to the newly created entity.

Associated User Cell:

We use AltUserCell widget for the Associated User.

Editing an List View:



**Iterator Buttons** widget will two buttons : Add and Remove.

**Edit Buttons**: Edit, Update, Cancel

Add: This is used to add a new rows with the Iterator Buttons.

Remove: This is used to Remove the existing rows from the LV.

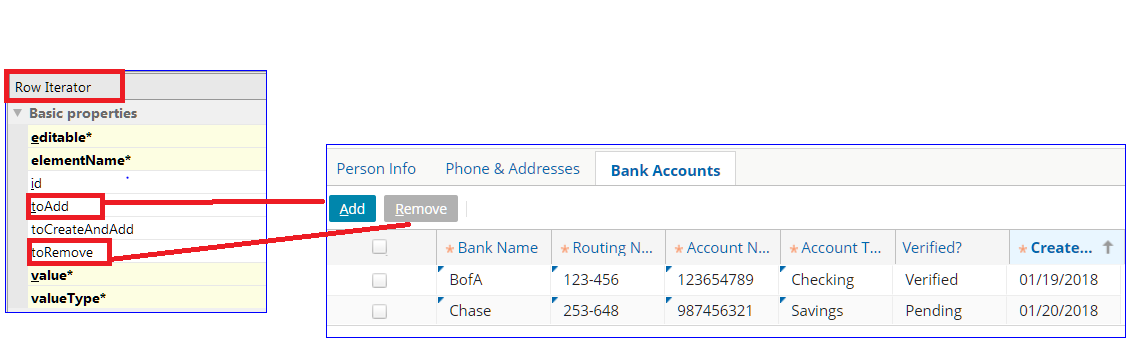
Toolbar:

Best place to place the edit buttons is the Screen Level toolbar

Best place to place the Iterator Buttons is the toolbar closest to the List view.

Iterator Buttons:

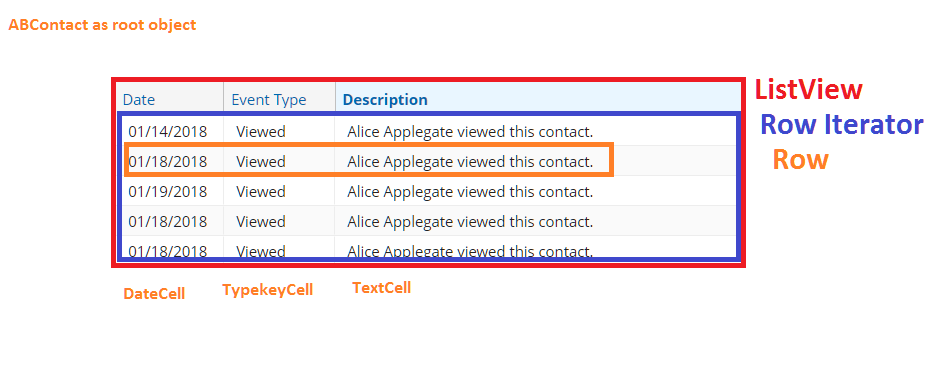
The properties in the Row Iterator will have capability to add the remove the rows from the LV.



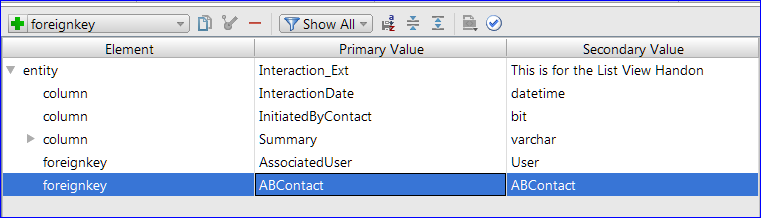
toAdd -- It will take action to add a new row when clicked.

toRemove - it will take action to remove a row when it is clicked.

**Editable Hierarchy:**



1. ListView Panel is editable.
2. Row Iterator is editable.
3. Row is editable.
4. Cell is editable.



**Popup’s**

Popup contains a single screen and returns the user to the previous location once the popup is closed.

OOTB -- > Automatically has Return to <previous location> link

Every Popup should contain a entry point.

An Entry point is a reference used by widgets to navigate to given location. An entry point specifies the location name and parameters required to render the location. Every location must have at least one entry point.

Important Note: Every parameter defined in the entry point must have a matching variable pair (Same Name and Same Type)

**Syntax for entry point**: NameofThePCFfile(parameter : Type)

**Syntax for the Action property from DV to popup is**

Popupfilename(variable name : variable Type)

**LV** : Popupfilename(rowIterater Elementname)

**Validation**

**Session Objectives:**

Field-Level Validation Configuration in the UI and in the data model.

Input mask differs from the regular expression.

Entity field validator

Validation Rule behavior

Validation Rules that raise warnings and errors.

**Pattern Matching:**

Pattern Matching is a field-level validation techinque.

Verifies that the user entered the data in the correct format for that field.

Example: Phone number format.

SSN.

**Pattern Matching** technique can be implemented in two tiers.

1. Data model.
2. UI tier

Main components of pattern matching are:

Regular Expression.

Input mask (optional)

**Pattern matching - regular expression**:

A regular expression defines an abstract pattern.

**Example**:

EmailAddress: .+@.+\..+

Routing Numer: [0-9a-zA-Z]{3} - [0-9]{3}

**Pattern Matching - input masks**

Appears as a watermark in the field and it also provides over tool tip.

Pattern Matching - Practices Comparison.

Data Model User Interface

**Pros**:

App enforces compliance in the UI Dyanmic values: regex

And through API. input mask can change on

Support full localization. Business logic.

Robust extensibility.

**Cons**: **Cons**

Static vales only: Only a few widget supportit

Regex and input mask cannot be Must be configured per

Changed based on the business logic. widget

Implemented as a reusable field No Custom error message

Validator. Option.

Extensibility is a big concern

Ternary operator Syntax:

Condition? Return if condition true : return if condition false.

Example: Contact == Policy Compnay ? True : false

Business Requirement:

Policy number should contain 2 Chars and 7 Digits. There should be hypen in between first two chars and next 7 digits.

There are three possible outcomes:

1.Reject with error

The action is prevented and a message is displayed to the user in the in user interface.

2.Reject with warning

The actions is permitted, but the user is warned of some potential issue.

3. No rejection

The object is valid and the action is permitted.

**Syntax**

reject(errorLevel, errorMessage, warningLevel, warningMessage)

Above syntax is used to display warning/error message and No field highlighting required.

rejectField(strRelativeFieldPath, errorLevel, errorMessage, warningLevel, warningMessage)

Code:

**package** rules.Validation.ABContactValidationRules\_dir

**uses** gw.api.locale.DisplayKey

@gw.rules.RuleName(**"Married Tax Filing status"**)

**internal class** MarriedTaxFilingstatus {

**static function** doCondition(aBContact : entity.ABContact) : **boolean** {

*/\*start00rule\*/*

**return** aBContact **typeis** ABPerson

**and**

(aBContact.**TaxFilingStatus** == TaxFilingStatusType.***TC\_MARRIED\_JOINT***

**or**

aBContact.**TaxFilingStatus** == TaxFilingStatusType.***TC\_MARRIED\_SEPARATE***

)

**and**

aBContact.**MaritalStatus** != MaritalStatus.***TC\_MARRIED***

*/\*end00rule\*/*

}

**static function** doAction(aBContact : entity.ABContact, actions : gw.rules.Action) {

*/\*start00rule\*/*

*//aBContact.rejectField("TaxFilingStatus",ValidationLevel.TC\_LOADSAVE, DisplayKey.get("MarriedTaxFilingStatus"),null,null)*

aBContact.rejectField(**"TaxFilingStatus"**,**null**, **null** ,ValidationLevel.***TC\_LOADSAVE***,DisplayKey.*get*(**"MarriedTaxFilingStatus"**))

*/\*end00rule\*/*

}

}