Vanrise Coding Conventions

**Table of Contents**

[**1** **Introduction** 3](#_Toc16674360)

[**2** **General Guidelines** 4](#_Toc16674361)

[**2.1** **Naming conventions** 4](#_Toc16674362)

[**2.2** **Project Structure (Old & new Convention)** 4](#_Toc16674363)

[**2.3** **Grid** 5](#_Toc16674364)

[**2.3.1** **Client Grid** 5](#_Toc16674365)

[**2.3.2** **Server Grid** 6](#_Toc16674366)

[**2.3.3** **Drill Down** 7](#_Toc16674367)

[**2.4** **Search Pages** 7](#_Toc16674368)

[**2.5** **Editor** 7](#_Toc16674369)

[**2.6** **Selectors** 8](#_Toc16674370)

[**2.7** **Extensions** 9](#_Toc16674371)

[**2.8** **Loader uses and Loading Dependences** 9](#_Toc16674372)

[**2.9** **UI Alignments** 10](#_Toc16674373)

[**2.10** **Multi-layers UI communication** 10](#_Toc16674374)

[**2.11** **General Rules** 11](#_Toc16674375)

[**2.12** **Translation** 12](#_Toc16674376)

[**2.13** **Architecture conventions (Always start with single abstraction / use wrapper class for abstraction** 13](#_Toc16674377)

[**3** **Static Guidelines** 13](#_Toc16674378)

[**4** **Dynamic Guidelines** 14](#_Toc16674379)

[**4.1** **Definition – Runtime abstraction** 14](#_Toc16674380)

[**4.2** **Main grid** 14](#_Toc16674381)

[**5** **Invoice Type** 14](#_Toc16674382)

[**5.1** **Actions** 14](#_Toc16674383)

[**5.2** **Bulk Actions** 15](#_Toc16674384)

[**5.3** **Sub Views** 15](#_Toc16674385)

[**5.4** **File Parts** 15](#_Toc16674386)

[**5.5** **Serial Number Parts** 16](#_Toc16674387)

[**6** **Generic Business Entity** 16](#_Toc16674388)

[**6.1** **Sub views** 16](#_Toc16674389)

[**6.2** **Actions** 17](#_Toc16674390)

[**6.3** **Handlers** 17](#_Toc16674391)

[**6.4** **Custom Actions** 18](#_Toc16674392)

[**6.5** **Bulk Actions** 18](#_Toc16674393)

[**7** **Business Entity** 18](#_Toc16674394)

[**7.1** **Static Business Entity** 18](#_Toc16674395)

[**7.2** **Comment Business Entity** 18](#_Toc16674396)

[**7.3** **Lookup Business Entity** 18](#_Toc16674397)

[**7.4** **Account Business Entity** 18](#_Toc16674398)

[**8** **Other Modules** 18](#_Toc16674399)

[**8.1** **Settings** 18](#_Toc16674400)

[**8.2** **Object Type Definition** 18](#_Toc16674401)

[**8.3** **Component Type** 18](#_Toc16674402)

[**8.4** **Bulk Action Drafts Server Side** 19](#_Toc16674403)

[**8.5** **Excel Library** 19](#_Toc16674404)

**Table of Figures**

[Figure 1 Old Hierarchy 4](file:///C:\TFS\Vanrise\Code\Resources\Documents\Framework%20Guidelines.docx#_Toc16092903)

[Figure 2 New Hierarchy 5](file:///C:\TFS\Vanrise\Code\Resources\Documents\Framework%20Guidelines.docx#_Toc16092904)

[Figure 3 Grid Template 6](#_Toc16092905)

[Figure 4 Data Retrieval Function 6](#_Toc16092906)

[Figure 5 GetFiltered Controller 7](#_Toc16092907)

[Figure 6 GetFiltered Manager 7](#_Toc16092908)

[Figure 7 Search Grid Directive 7](#_Toc16092909)

[Figure 12 Editor Runtime For Complex Editors 8](#_Toc16092910)

[Figure 13 Editor Runtime Class 8](#_Toc16092911)

[Figure 8 Selector Template 8](#_Toc16092912)

[Figure 9 Selector Implementation 8](#_Toc16092913)

[Figure 10 Selector Service Implementation 8](#_Toc16092914)

[Figure 11 Selector Controller Implementation 9](#_Toc16092915)

# **Introduction**

The first step of making a framework we should think about maintainability, complexity and coding quality. To satisfy these three main concepts in programming we should follow a set of guidelines/conventions.

These guidelines were put according to company vision and the way that make high productivity in small interval of time.

In addition, all conventions were put after discussions and years of experience in applying best practices to have such great framework.

# **General Guidelines**

## **Naming conventions**

Let us have an entity names: Country

1. Management pages: CountryManagement
2. Editors: CountryEditor
3. Selector: {solution}-{module}-country-selector
4. Grid: {solution}-{module} -country-grid
5. Abstract settings: {solution}-{module}-county-setting
6. API Controller: CountryContoller
7. Business Manager: CountryManager
8. IDataManager: ICountryDataManager
9. DataManager: CountryDataManager

Note: solution will be replaced by solution symbol ex: Tone: whs, Vanrise: vr, Retail: retail.

Module will be replaced by module name example: common, genericdata ... etc

## **Project Structure (Old & new Convention)**

Rules:

1. Don’t mix between old and new.
2. Any new project created should follow the new convention.
3. Old Hierarchy:

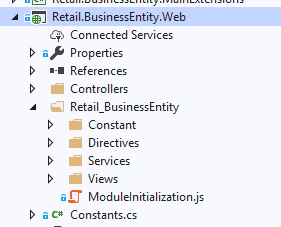


Figure Old Hierarchy

1. New Hierarchy

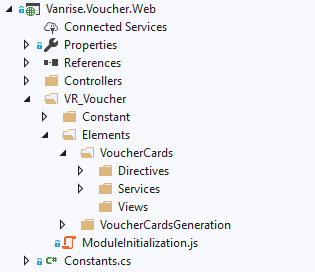


Figure New Hierarchy

Implementations

When we need to create a new implementation for an abstract class within same project it should be created in MainExtensions but if it is related for another project it should be in Business layer.

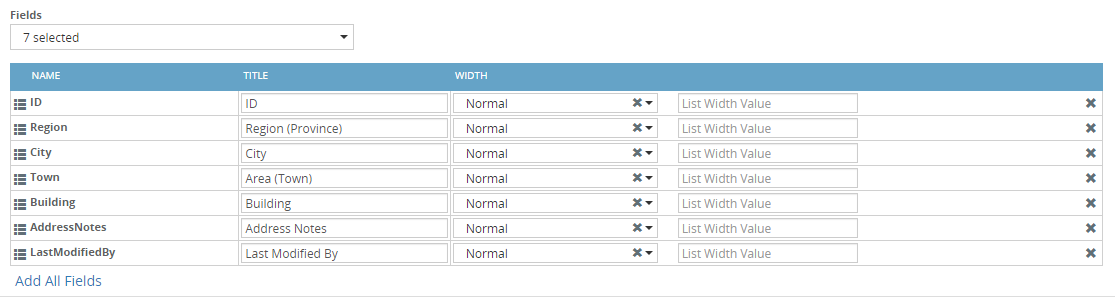
## **Grid**

### **Client Grid**

1. It’s required to put the dataItem of the grid in an object to collect the clean object.

Ex: $scope.datasource.push({Entity: dataItem});

1. It’s better to put the fields editable in the columns instead of popups for better user experience.



1. In case having settings, it’s better to use drill down instead of popups for better user experience.

### **Server Grid**

1. When we want to develop a static grid, the grid implemented as a directive with load method.
2. Grid Directive Template



Figure Grid Template

1. Grid Server Side Loading by using GetFiltered



Figure Data Retrieval Function



Figure GetFiltered Controller



Figure GetFiltered Manager

### **Drill Down**

Drilldowns should be implemented in the form of registration to allow adding different drill downs from several modules.

## **Search Pages**

Any drilldown should be considered as search page so we need to build two directives one for grid and the second for search and we use the grid inside it as following:

Search Directive



Figure Search Grid Directive

## **Editor**

While implementing complex editor we should avoid multiple calls to get the entity needed. The best approach to implement such behavior if creating editorruntime class to get it before loading the editor.

Ex:



Figure Editor Runtime For Complex Editors



Figure Editor Runtime Class

## **Selectors**

1. It’s required to use vr-columns in the selectors directive and pass normal-col-num attribute from outside.



Figure Selector Template

1. Selectors load its data by get api method using a function on the controller named Get{Entities}Info that takes a filter as parameter.



Figure Selector Implementation



Figure Selector Service Implementation



Figure Selector Controller Implementation

1. It’s better to expose these functions to be used in selectors

* selectItem :used as api.selectItem(itm) where itm is a  specific object from the selector datasource.
* selectFirstItem: used as api.selectFirstItem().
* selectIfSingleItem: used as api.selectIfSingleItem ().

## **Extensions**

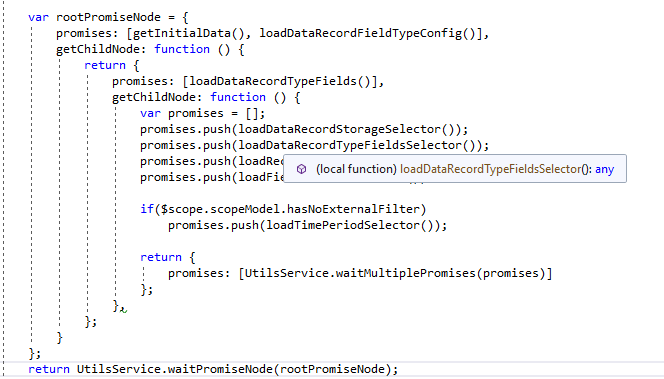
Extensons are the implementations of abstract classes, if these implementations are used in UI and the user choose between them so these should follow the following steps:

1. Each class should implement the abstract class and should override a ConfigId property.
2. Each class should have a directive that return its type
3. The implemention for all directives should be places in ExtensionConfiguration table and for same abstract class we have only one configtype.

## **Loader uses and Loading Dependences**

Loader is the most important thing in the application to prevent user from taking any action before we finish loading the pages.

1. Always use PromiseNode



1. Dependent Selectors
2. Loading when using directive wrapper

## **UI Alignments**

We should be care about the alignment of all controls in the page or editor, they should be aligned and same size and the trick behind the alignments is the way we use vr-row and vr-columns.

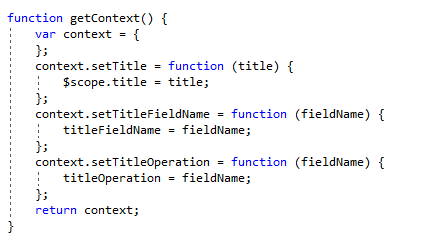
|  |  |
| --- | --- |
| Wrong behavior ✘ | Correct behavior ✓ |
|  |  |

## **Multi-layers UI communication**

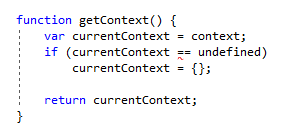
The most important thing in multi-layer UI for example we have two or three pop-ups and we need to let the third one communicate with the first one we need to create a context as an object that have only functions van be passed for all editor and each editor will call the function needed.

In addition, in each directive or editor we need to create another context that contain a copy from the previous one plus the additional functions to be passed for any child control.

1. In main editor:



1. In sub editor



## **General Rules**

The below rules should be applied instead you have a specific permission from your supervisor.

1. Copy – Past is forbidden.
2. Inject only needed services in the controller.
3. Not allowed to use settimeout function.
4. Not allowed to use watchers in business directives.
5. Don’t use “vr-loader” or “vr-disable” on directive wrapper, try to use a span outside the directive wrapper.
6. Row must not be created directly inside another row, you can create a row, then a column, and then another row in the column.
7. While Using vr-columns we should use **width** attribute instead of **colnum** when applicable.
8. When creating a column inside another column, haschildcolumns must be added to the parent column.
9. We should always pass and wrapper object as payload for the directive, instead of passing only the data to give the ability for any future modification.
10. It’s required to use scopeModel Object in all js controllers and directives.
11. Think of increasing the width of vr-modal if you find that one or more of your modal components can take more space than expected.
12. All  vr-datagridcolumn must have widthfactor attribute to ensure that will take enough space to render correctly.
13. Make sure to use widthfactor attribute as integer value and not as a decimal.

e.g:

**a- Valid usage**

<vr-datagridcolumn headertext="'Name'" field="'Name'" widthfactor="8" disablesorting="true"></vr-datagridcolumn>

**b- Invalid usage**

<vr-datagridcolumn headertext="'Name'" field="'Name'" widthfactor="0.8" disablesorting="true"></vr-datagridcolumn>

1. When Using  vr-columns base directive make sure that **colnum** attribute is an integer between 1 and 12  and not one of values ("normal" ,"small"  ,"large"  ,"fullrow","1/2row" ,"1/4row" ,"3/4row" ,"1/3row" ,"2/3row") only **width** attribute can be passed that way.
2. Never use **vr-row** tag container without following it by using  **vr-columns**.

**Wrong implementation**

<vr-row>

         <vr-common-document-grid on-ready="scopeModal.onDocumentGridReady"></vr-common-document-grid>

      </vr-row>

**Right implementation**

<vr-row>

       <vr-columns width="fullrow">

          <vr-common-document-grid on-ready="scopeModal.onDocumentGridReady"></vr-common-document-grid>

           </vr-columns>

</vr-row>

## **Translation**

We should care about translation in static and generic modules.

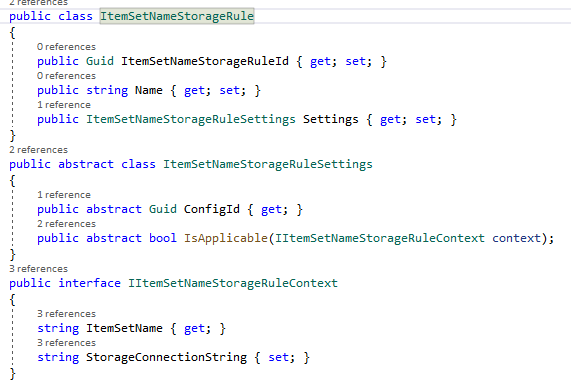
1. In static module its easy to implement just by adding translation attribute for all controls and give them the key needed as following “VRRes.{Module}.{ItemKey}.VREnd”.
2. In dynamic module like invoice, analytic or generic business entity we translate all entity server side so we should implement the apply translation method for any new abstract implementation.

## **Architecture conventions (Always start with single abstraction**

When we have abstract class, we should use a wrapper class to handle any new modification needed.

Note: every method in abstract class should take a context as parameter which is an interface having set and get properties to be used in overridable methods.

Ex:



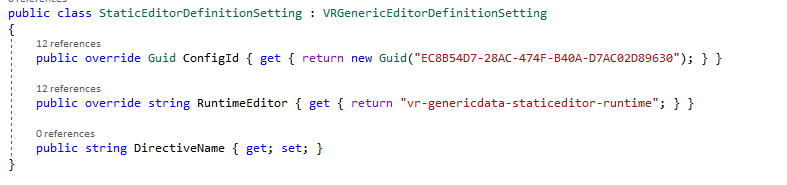
# **Static Guidelines**

In this section we will talk about the guidelines that help us to build a static page following the best practices and the latest conventions used.

# **Dynamic Guidelines**

## **Definition – Runtime abstraction**

1. When we want to use an abstract class for definition and runtime purpose, we should put the runtime directive as a property in the definition class.



## **Main grid**

1. Developing such grid required a definition entity having the settings for grid columns, these columns should be evaluated server side before loading the grid. (ref: Check Generic BE Grid).

# **Invoice Type**

## **Actions**



## **Bulk Actions**



## **Sub Views**



## **File Parts**



## **Serial Number Parts**



# **Generic Business Entity**

## **Sub views**

The implementation of sub views should be in the same way as registration, but in dynamic behavior we should return the valid sub views on the data Item entity that allow us to find all sub views needed to be shown.

This allow us to hide and show any sub view not match with a specific record under certain condition.



## **Actions**





## **Handlers**

1. Before Save Handler
2. After Save Handler
3. Before GetFiltered Handler

## **Custom Actions**

Same as Actions but another abstract class.

## **Bulk Actions**

# **Business Entity**

## **Static Business Entity**

To create static business entity, we should use the static business entity class “StaticBEDefinitionSettings” and put it manual in BusinessEntityDefinition table.

## **Comment Business Entity**

## **Lookup Business Entity**

## **Account Business Entity**

# **Other Modules**

## **Settings**

Steps to create new setting:

1. We should make a new directive to be placed in common editor and contains the data.



1. Setting Should be created manually in database.



Note: the flag IsTechnical set to true (1) just in case this setting for technical purpose and the clients cannot take any action for it.

1. The data of the settings will be filled in data field and should implement a SettingData Class like the following:



1. One config cs file should be created per module to return this setting and it should be called ConfigManager:



## **Object Type Definition**

## **Component Type**

To create a component type we should create a class implement VRComponentType class and pass another class for Setting in the generic. Ex:



Then at the end put the directive in extension configuration.

## **Bulk Action Drafts Server Side**

## **Excel Library**