Vanrise Coding Conventions

**Table of Contents**

[**1** **Introduction** 3](#_Toc15480376)

[**2** **General Guidelines** 4](#_Toc15480377)

[**2.1** **Naming conventions** 4](#_Toc15480378)

[**2.2** **Project Structure (Old & new Convention)** 4](#_Toc15480379)

[**2.3** **Grid** 5](#_Toc15480380)

[**2.3.1** **Client Grid** 5](#_Toc15480381)

[**2.3.2** **Server Grid** 6](#_Toc15480382)

[**2.3.3** **Drill Down** 7](#_Toc15480383)

[**2.4** **Search Pages** 7](#_Toc15480384)

[**2.5** **Editor** 7](#_Toc15480385)

[**2.6** **Selectors** 8](#_Toc15480386)

[**2.7** **Extensions** 8](#_Toc15480387)

[**2.8** **Loader uses and Loading Dependences** 9](#_Toc15480388)

[**2.9** **UI Alignments** 10](#_Toc15480389)

[**2.10** **Multi-layers UI communication** 10](#_Toc15480390)

[**2.11** **General Rules** 11](#_Toc15480391)

[**2.12** **Architecture conventions (Always start with single abstraction / use wrapper class for abstraction** 11](#_Toc15480392)

[**3** **Static Guidelines** 12](#_Toc15480393)

[**4** **Dynamic Guidelines** 12](#_Toc15480394)

[**4.1** **General** 13](#_Toc15480395)

[**4.1.1** **Definition – Runtime abstraction** 13](#_Toc15480396)

[**4.1.2** **Main grid** 13](#_Toc15480397)

[**4.2** **Settings** 13](#_Toc15480398)

[**4.3** **Invoice Type** 13](#_Toc15480399)

[**4.3.1** **Actions** 13](#_Toc15480400)

[**4.3.2** **Bulk Actions** 13](#_Toc15480401)

[**4.3.3** **Sub Views** 13](#_Toc15480402)

[**4.3.4** **File Parts** 13](#_Toc15480403)

[**4.3.5** **Serial Number Parts** 13](#_Toc15480404)

[**4.4** **Object Type Definition** 13](#_Toc15480405)

[**4.5** **Component Type** 13](#_Toc15480406)

[**4.6** **Generic Business Entity** 14](#_Toc15480407)

[**4.6.1** **Sub views** 14](#_Toc15480408)

[**4.6.2** **Actions** 14](#_Toc15480409)

[**4.6.3** **Handlers** 15](#_Toc15480410)

[**4.6.4** **Custom Actions** 15](#_Toc15480411)

[**4.6.5** **Bulk Actions** 15](#_Toc15480412)

**Table of Figures**

[Figure 1 Old Hierarchy 7](file:///C:\TFS\Vanrise\Code\Resources\Documents\Framework%20Guidelines.docx#_Toc15477676)

[Figure 2 New Hierarchy 7](file:///C:\TFS\Vanrise\Code\Resources\Documents\Framework%20Guidelines.docx#_Toc15477677)

[Figure 3 Grid Template 8](#_Toc15477678)

[Figure 4 Data Retrieval Function 9](#_Toc15477679)

[Figure 5 GetFiltered Controller 9](#_Toc15477680)

[Figure 6 GetFiltered Manager 9](#_Toc15477681)

[Figure 7 Search Grid Directive 10](#_Toc15477682)

[Figure 8 Selector Template 10](#_Toc15477683)

[Figure 9 Selector Implementation 10](#_Toc15477684)

[Figure 10 Selector Service Implementation 10](#_Toc15477685)

[Figure 11 Selector Controller Implementation 10](#_Toc15477686)

[Figure 12 Editor Runtime For Complex Editors 11](#_Toc15477687)

[Figure 13 Editor Runtime Class 11](#_Toc15477688)

# **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:

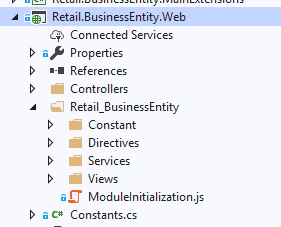


Figure Old Hierarchy

1. New Hierarchy

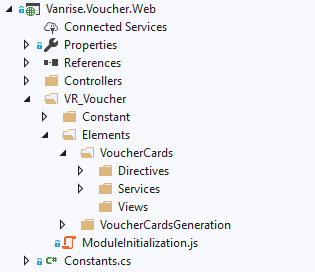


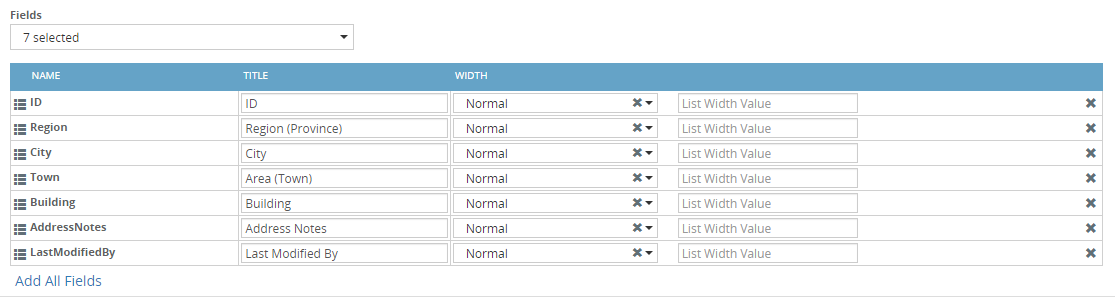
Figure New Hierarchy

## **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.
2. 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

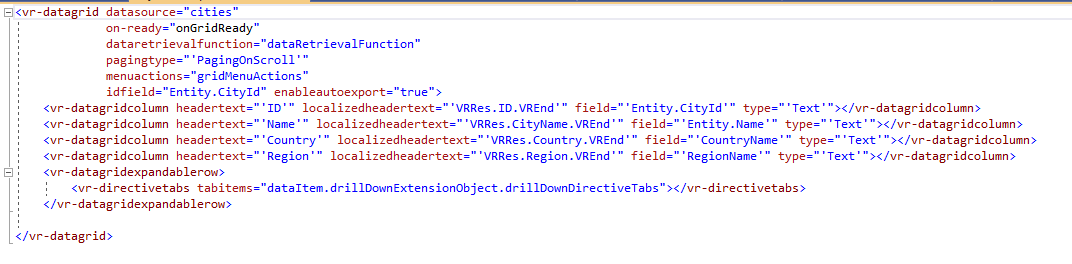


Figure Grid Template

1. Grid Server Side Loading by using GetFiltered

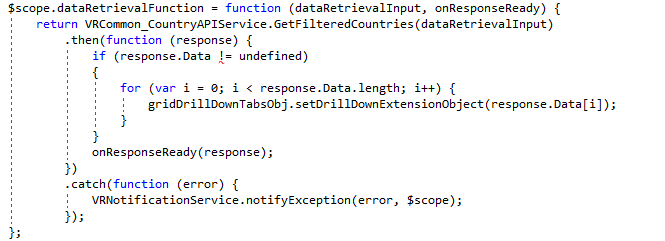


Figure Data Retrieval Function

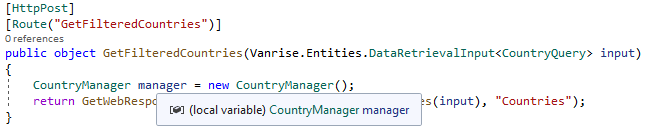


Figure GetFiltered Controller

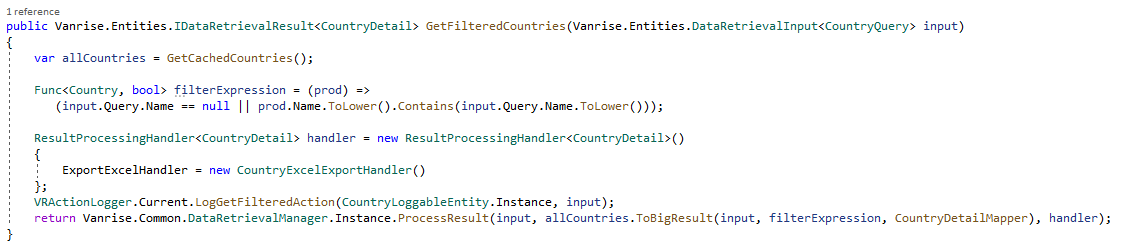


Figure GetFiltered Manager

### **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:

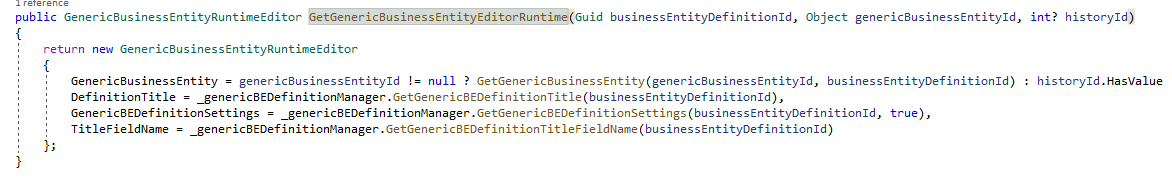


Figure Editor Runtime For Complex Editors

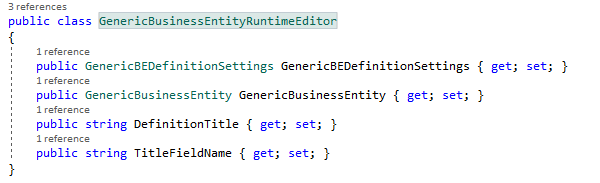


Figure Editor Runtime Class

## **Selectors**

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

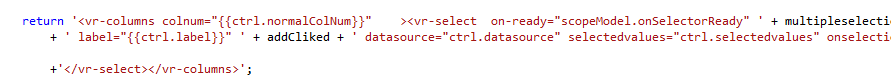


Figure Selector Template

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.

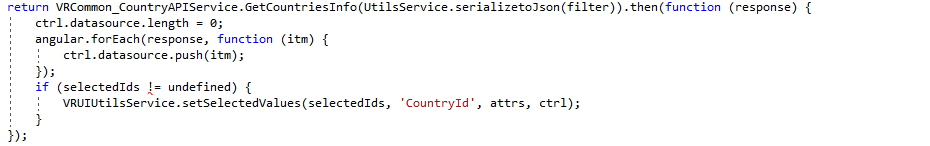


Figure Selector Implementation

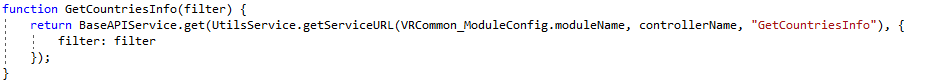


Figure Selector Service Implementation

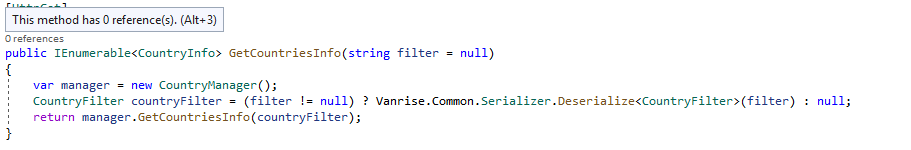


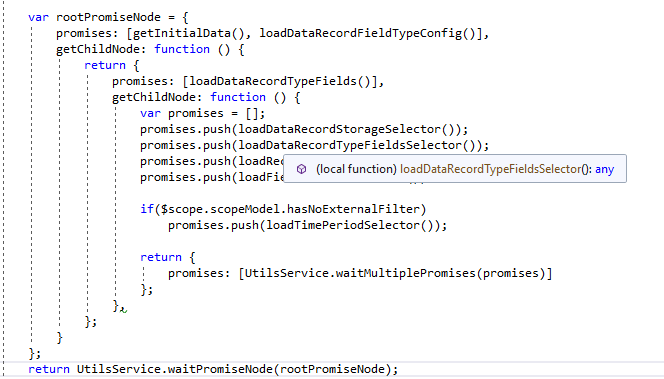
Figure Selector Controller Implementation

## **Extensions**

## **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.

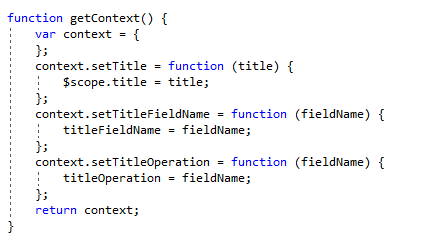
|  |  |
| --- | --- |
| Wrong behavior ✘ | Correct behavior ✓ |
|  |  |

## **Multi-layers UI communication**

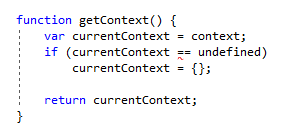
The most important thing in multi-layer UI for example we have two or three pop-ups and we need to let the third one communicate with the first one we need to create a context as an object that have only functions van be passed for all editor and each editor will call the function needed.

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.

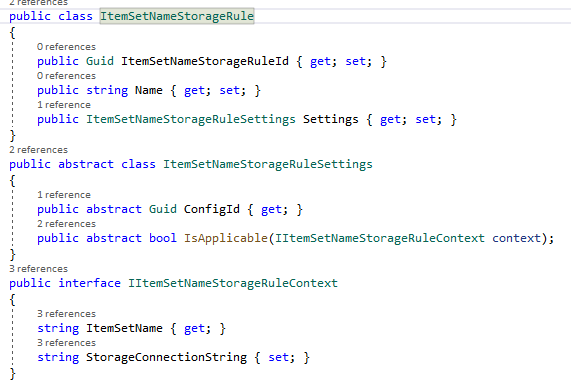
1. Not allowed to use settimeout function.
2. Not allowed to use watchers in business directives.
3. Don’t use “vr-loader” or “vr-disable” on directive wrapper, try to use a span outside the directive wrapper.

## **Architecture conventions (Always start with single abstraction / use wrapper class for 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.

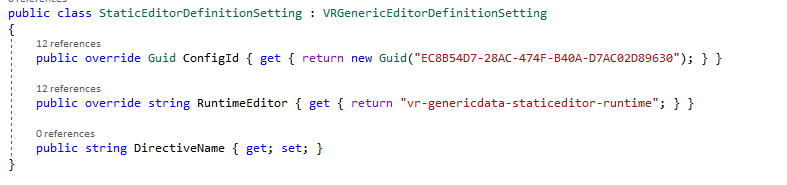
# **Dynamic Guidelines**

## **General**

In this section we will talk about the general conventions applicable in all dynamic or generic components.

### **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).

## **Settings**

## **Invoice Type**

### **Actions**

### **Bulk Actions**

### **Sub Views**

### **File Parts**

### **Serial Number Parts**

## **Object Type Definition**

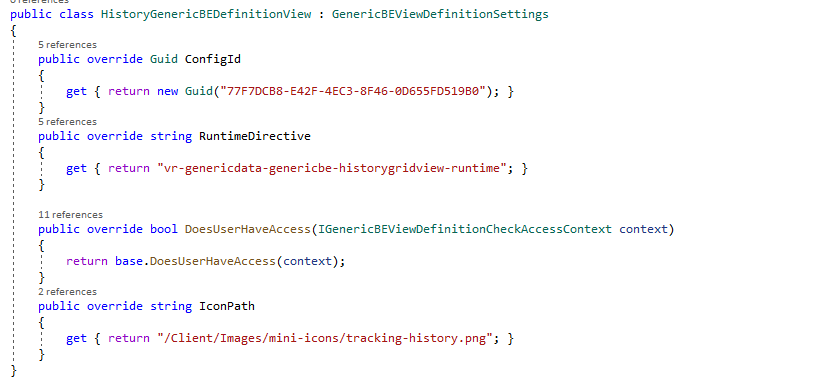
## **Component Type**

## **Generic Business Entity**

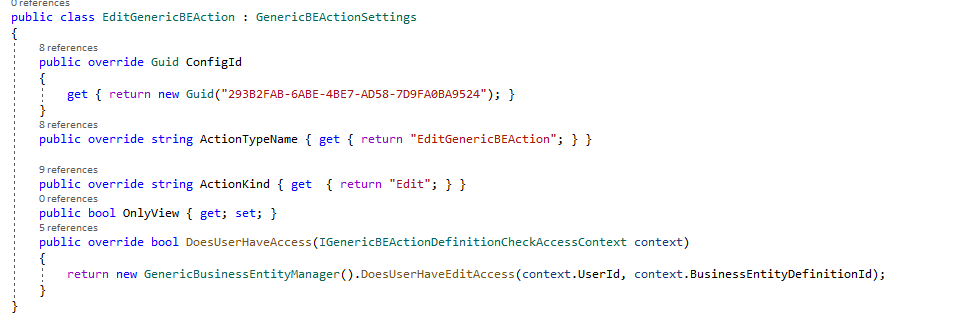
### **Sub views**

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**