Sage 300 Web Screens SDK

Grid Enhancement

August 2022

The MIT License (MIT)

Copyright © 2022 The Sage Group plc or its licensors. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Contents

[1. Overview 5](#_Toc456778699)

[2. Components 6](#_Toc456778700)

[2.1 GridPage.cshtml 6](#_Toc456778701)

[2.2 GridConfigViewModel.cs 6](#_Toc456778702)

[2.3 GridInfo.cs 6](#_Toc456778703)

[2.4 mergeGridConfiguration 6](#_Toc456778704)

[3. The Old Way 7](#_Toc456778705)

[3.1 Define grid preference variable in the view 7](#_Toc456778706)

[3.2 Define grid buttons in the view 7](#_Toc456778707)

[3.3 Define the actual grid 7](#_Toc456778708)

[3.4 Define the grid preference area 7](#_Toc456778709)

[3.5 If the grid is editable and requires editors 7](#_Toc456778710)

[3.6 The grid configuration has to manually create for every grid 8](#_Toc456778711)

[4. The New Way 10](#_Toc456778712)

[4.1 Put GridInfo attributes on model properties 10](#_Toc456778713)

[4.2 Define grid area in razor view 10](#_Toc456778714)

[4.3 Define the JavaScript object 10](#_Toc456778715)

[5. GridPage.cshtml 13](#_Toc456778716)

[6. GridConfigViewModel.cs 14](#_Toc456778717)

[6.1 Constructor Arguments 14](#_Toc456778718)

[6.2 Other Properties 14](#_Toc456778719)

[7. GridInfo.cs 16](#_Toc456778720)

[7.1 Constructor Arguments 16](#_Toc456778721)

[7.2 Other Properties 17](#_Toc456778722)

[7.3 Editor Types 17](#_Toc456778723)

[7.3.1 Text Editor 17](#_Toc456778724)

[7.3.2 Calendar Editor 17](#_Toc456778725)

[7.3.3 Finder Editor 17](#_Toc456778726)

[7.3.4 Dropdown Editor 18](#_Toc456778727)

[7.3.5 NumericTextBox Editor 18](#_Toc456778728)

[8. JavaScript 19](#_Toc456778729)

[9. Examples 22](#_Toc456778730)

[9.1 Segment Codes 22](#_Toc456778731)

[9.2 Source Journal Profiles 22](#_Toc456778732)

1. Overview

The purpose of this document is to explain and describe the new framework to define grid controls within Sage300 web screens.

1. Components

There are new files required for this enhancement along with a new JavaScript routine in the Global.js file:

* 1. GridPage.cshtml

A partial razor view to define and produce the HTML and JS code to construct data grids

* 1. GridConfigViewModel.cs

A view model that contains information regarding the properties of the grid (e.g. the type, location of data source, action buttons visibility, etc.)

* 1. GridInfo.cs

An attribute class used to mark the business model properties for the grid. It will also provide information regarding how the value will be presented in the grid (e.g. order of the column, title name, editable or not, html style, etc.)

* 1. mergeGridConfiguration

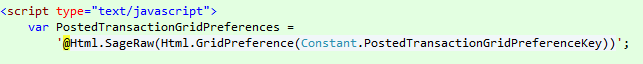
sg.utls.mergeGridConfiguration (propertiesArray, targetConfig, sourceConfig)

Merges the JSON properties as specified in the propertiesArray from targetConfig to sourceConfig. It will also copy any “additionalConfig” values from targetConfig to sourceConfig.

1. The Old Way

Here are the instructions on how a grid was defined in a web screen.

* 1. Define grid preference variable in the view



* 1. Define grid buttons in the view



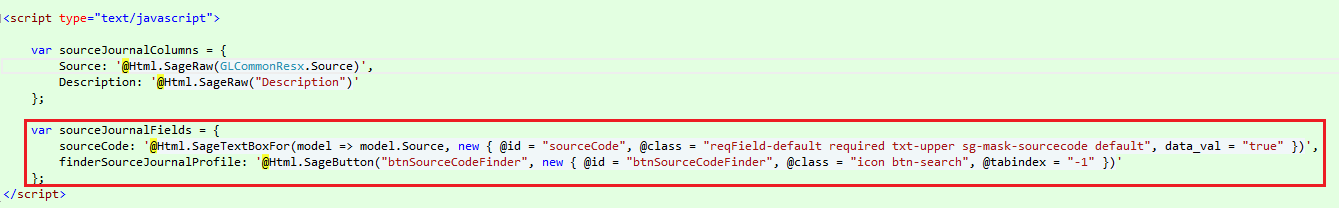
* 1. Define the actual grid



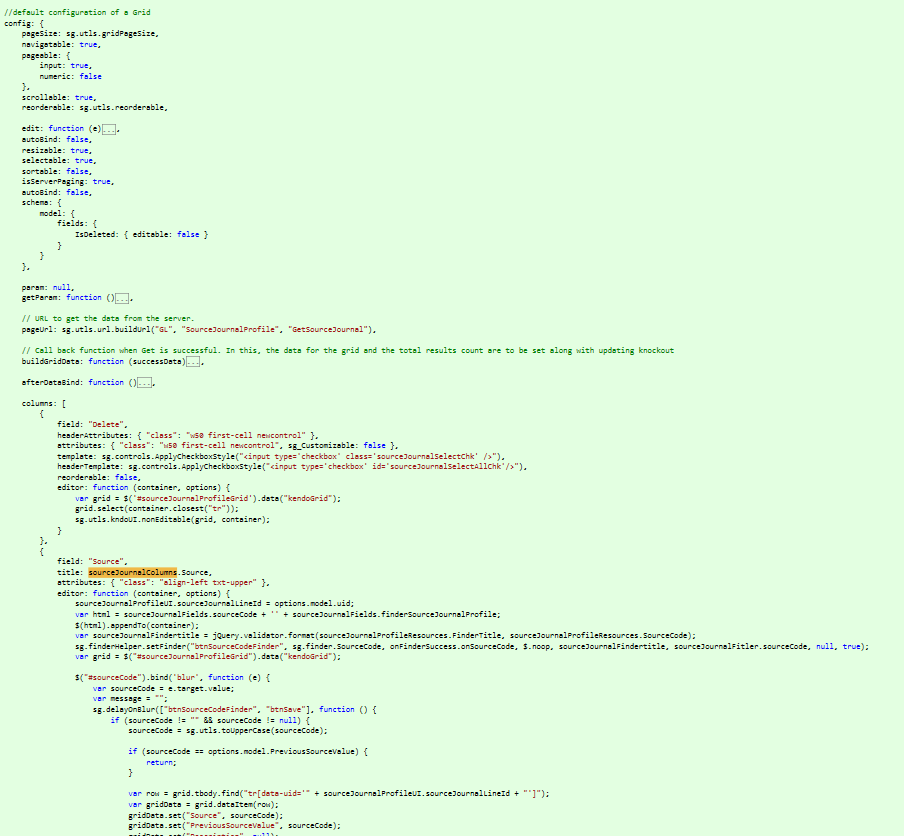
* 1. Define the grid preference area



* 1. If the grid is editable and requires editors



* 1. The grid configuration has to manually create for every grid



1. The New Way
   1. Put GridInfo attributes on model properties

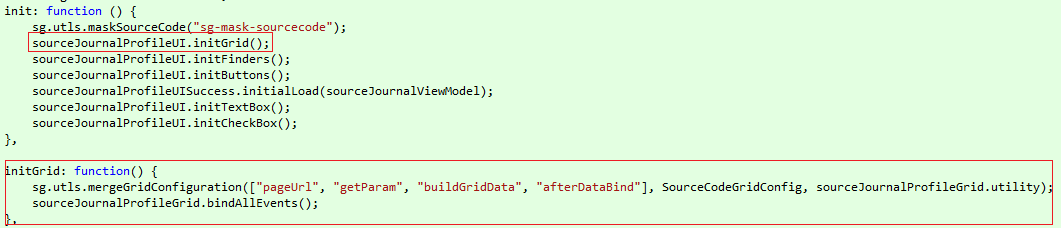


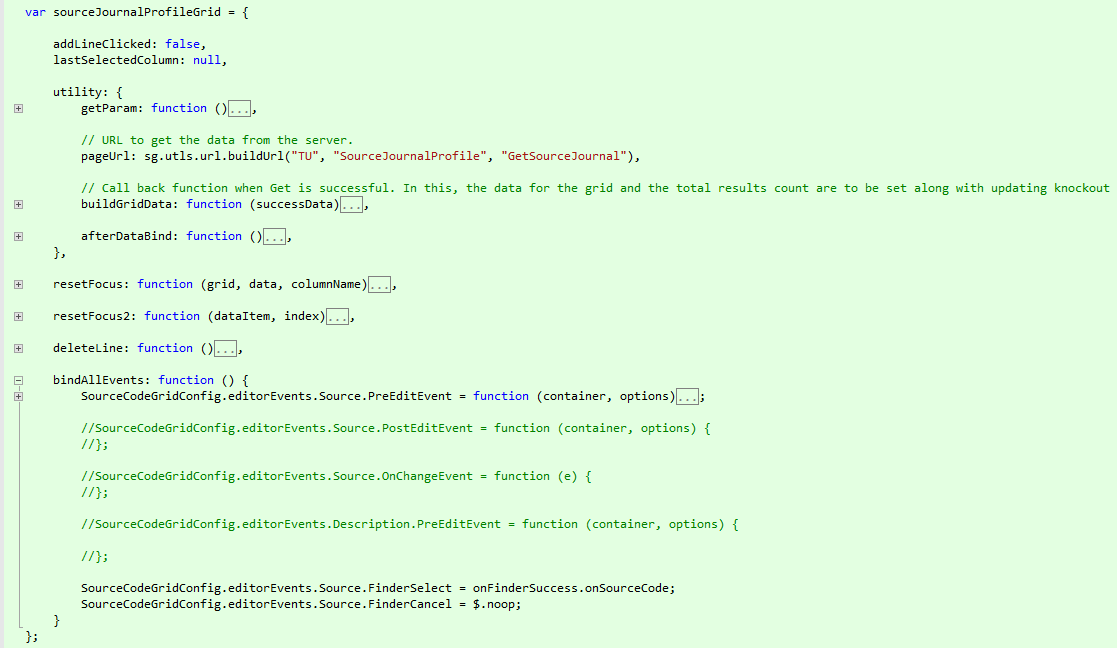
* 1. Define grid area in razor view



* 1. Define the JavaScript object

This defines how the grid will interact with the backend and extra actions on editor events.







1. GridPage.cshtml

This razor view page is responsible for generating all the HTML code of the Grid and also the configuration JavaScript object. The user should not need to interact with this file.

During parsing of the model, if the **StringLength** attribute is found along with **GridInfo**, the **maximumLength** will be used as the max length of the column.



1. GridConfigViewModel.cs

This is the view model corresponding to the **GridPage.cshtml**. It contains information regarding what kind of grid data should be used and the properties.



* 1. Constructor Arguments
* Type gridType
  + The type of business entity for the grid
* string dataSource
  + The JavaScript object that has the data for the grid
* object model
  + The model for the partial view page
* string gridKey
  + The key field of this grid used to uniquely identify each row of data
* (Optional) string gridPreferenceKey
  + If specified, it implies the grid is configurable and the setting will be saved across sessions
* (Optional) bool autoBind
  + Kendo grid setting which if set to false the grid will not bind to the data source during initialization. In this case, data binding will occur when the change event of the data source is fired. By default, the grid will bind to the data source specified in the configuration.
  1. Other Properties
* bool AddButtonVisibility
  + **true** to show the add button otherwise **false**
* bool DeleteButtonVisibility
  + **true** to show the delete button otherwise **false**
* bool EditButtonVisibility
  + **true** to show the edit button otherwise **false**
* bool HasDeleteColumn
  + **true** to show the checkboxes column for deletion otherwise **false**
* dynamic Controls
  + Contains information on how editors inside the grid will be created
  + Note: **users should not need to interact with this property**
  + Note: **must be called to prebuild all necessary controls**

1. GridInfo.cs

This is an attribute class used in business model. The properties which are going to be part of the grid will be mark by **GridInfo** attributes with the characteristics of the column.



* 1. Constructor Arguments
* int sequence
  + The order sequence of the column
* Type resourceType
  + The resource type to be used for the title
* string resourceName
  + The resource property name to be used for the title
* (Optional) bool editable
  + **true** if the column is editable otherwise **false** (default is **true**)
* (Optional) GridEditorNum editorType
  + The type of editor (default to **None**)
* (Optional) string templateSource
  + The template used for column formatting
  + Currently there are 2 predefined in **GridInfoTemplateLib** class
* (Optional) Type[] editorResourceTypes
  + Array of resource types to be used for editor title (default to **null**)
* (Optional) string[] editorResourceNames
  + Array of resource property names to be used for editor title (default to **null**)
* string propertyName
  + Will be assigned with the current property name automatically
  + Should not be specified
  1. Other Properties
* string Field
  + Name of the field, if not set, model property name will be used
* bool Hidden
  + **true** to hide column otherwise **false** (default to **false**)
* string Style
  + CSS style of column
* string EditorHtmlClass
  + CSS class of the field editor

The follow properties are group by the type of editor of the column

* 1. Editor Types
     1. Text Editor
* bool IsAlphaNumericEditor
  + **true** if field allows alpha numeric characters otherwise **false**
    1. Calendar Editor

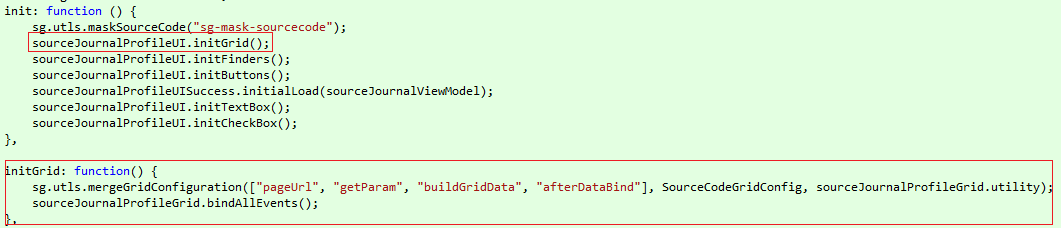
(Suggestion) set templateSource to **GridInfoTemplateLib.FormattedDate**

* + 1. Finder Editor
* string FinderName
  + Name of the finder
* string FinderTitle
  + String of finder title
  + If not set, it will default to “Select <finder type>”. However, if editorResourceTypes and editorResourceNames are defined, the first type and name will be use as the string template and the result will be use as arguments.
* string FinderSourceId
  + The id field of the finder window
    1. Dropdown Editor
* string DropDownSourceField
  + The field name in the view model that contains the list of dropdown values
    1. NumericTextBox Editor
* string NumericDecimalField
  + The field name in the view model that contains the number of decimals
* int NumericDecimalMaxLength
  + The max length of the numeric field, if not defined, it will be default to **16**

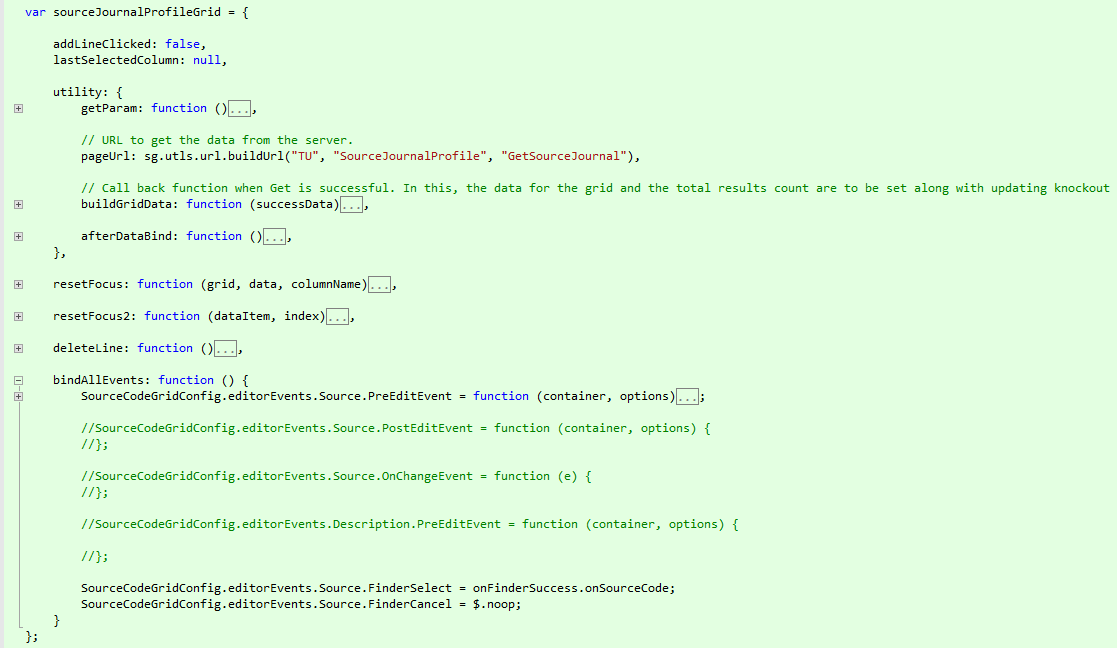
1. JavaScript

The final step of defining a grid control is to specify the remaining grid properties in order for the grid to work properly.

There are 2 calls at a minimum that JavaScript code will need to make.



* sg.utls.mergeGridConfiguration(<array>, <sourceGridConfig>, <otherGridConfig>)
  + <array> - should contains at least four strings, "pageUrl", "getParam", "buildGridData", "afterDataBind".
  + These are the name of the properties/functions that will be copied from <otherGridConfig> to <sourceGridConfig>
    - pageUrl
      * The string of URL remote server
    - getParam
      * A function to be called to get the parameter before making the server call
    - buildGridData
      * A function that input as the data return from the server and should return a JavaScript object with properties totalResultsCount (total number of result beyond current page) and data (data for the current page)
    - afterDataBind
      * A function will be called after the data is bound. It can perform hiding of columns, setting checkbox states, or anything necessary.
  + <sourceGridConfig>
    - The auto-generated JavaScript object that defines the grid. By default, it should have the name <BusinessModel>Grid (e.g. SourceCodeGridConfig)
  + <otherGridConfig>
    - A user defined JavaScript object that has the above 4 properties defined plus an optional **additionalConfig** property.
      * additionalConfig
        + If defined, it will copy or overwrite the properties defined in the sourceGridConfig



* When a field is defined using any one of the editor types, user has options to define PreEditEvent, PostEditEvent, OnChangeEvent, FinderSelect, and FinderCancel. Users at this point, should assign the handler of this events for each type of properties. The formation should be <sourceGridConfig>.editorEvents.<propertyName>.<eventType> For example: ***SourceCodeGridConfig.editorEvents.Source.FinderSelect = onFinderSuccess.onSourceCode;***
  + PreEditEvent
    - This event is called before the editor is created to perform any state checking or actions. It should return **true**, if the creation of the editor should continue, or **false**, the editor will not be created.
  + PostEditEvent
    - This event is called after the editor is created.
    - Note: this function is called regardless the value return from PreEditEvent.
  + OnChangeEvent
    - This event is called when the value of the editor is changed.
  + FinderSelect
    - This event is called when the editor is a finder and user has selected a value.
  + FinderCancel
    - This event is called when the editor is a finder and user cancels the finder.
  + FinderFilter
    - This event is called when the editor is a finder and user opens the finder from the editor.

1. Examples

There are two sample projects in the Sage 300 Web SDK which have been refactored to use this grid enhancement. Refer to these projects to explore the implementation and debug the solution to see the running behavior.

* 1. Segment Codes
  2. Source Journal Profiles