Sage 300 Web Screens SDK

Finder Architecture

May 2019

The MIT License (MIT)

Copyright © 2019 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. Introduction 4](#_Toc7620539)

[1.1 Too much code 4](#_Toc7620540)

[1.2 Too much complex and redundant code 4](#_Toc7620541)

[1.3 Customization 4](#_Toc7620542)

[1.4 Inefficient 4](#_Toc7620543)

[1.5 Dynamic Finders 5](#_Toc7620544)

[2. Workflow Diagram 6](#_Toc7620545)

[3. Overview 7](#_Toc7620546)

[3.1 New Finder Files 7](#_Toc7620547)

[3.1.1 …Common.Plugin.ViewFinder.js 7](#_Toc7620548)

[3.1.2 …Common.Plugin.ViewFinderProperties.js 7](#_Toc7620549)

[3.1.3 …Common.ViewFinderGrid.js 7](#_Toc7620550)

[4. Interface 8](#_Toc7620551)

[4.1 Old Interface 8](#_Toc7620552)

[4.2 New interface 8](#_Toc7620553)

[5. Examples 12](#_Toc7620554)

[5.1 AP Distribution Sets 12](#_Toc7620555)

[5.2 IC Price List Code 12](#_Toc7620556)

[5.3 AR Refund Entry 14](#_Toc7620557)

[5.4 PO Purchase Order Entry 14](#_Toc7620558)

[5.5 Employee Finder 15](#_Toc7620559)

[6. Future Considerations 17](#_Toc7620560)

1. Introduction

The Sage 300 Web Screen Finder implementation differs dramatically from the Visual Basic or desktop implementation. This has caused confusion and complexities for partners attempting to implement a finder from both a coding and customization perspective.

* 1. Too much code

There is simply too much code required to setup a finder in the web screens. The entity’s internal finder controller can range anywhere from 300 lines to over 4000 lines. And, that is just to define the controller. In the entity’s JavaScript file, there is setup code, event call- backs and logic dealing with the selected or returned record.

There are over 227 internal finder controllers for the Sage modules. And, third party modules further add to this number.

* 1. Too much complex and redundant code

As just stated, not only is there too much code, but the code that is present is redundant in that is must specify what columns to use, their sizes, descriptions and other such information that is obtainable via other means such as the MVC model.

* 1. Customization

It is possible for a third party to add a Sage finder to their module’s web screen. They must add Sage’s 6 assemblies for a module to their web project to implement the finder.

The customization mechanism for the web screens is different than that of the VB desktop.

However, a third party is unable to add one of their finders to a Sage web screen via the current customization process. Plus, a third party is not able to discover other third party finder ids for the purpose of adding their finder to a screen.

* 1. Inefficient

The web screen finders were developed to make use of the Controller, Service and Business Repository layers of the web screens as they have all the methods and structure for returning data from an Accpac Business View, mapping into the MVC model and returning that model. From a re-use standpoint, this is fantastic.

However, from an efficiency standpoint, this is very inefficient and costly. Let’s look at an example:

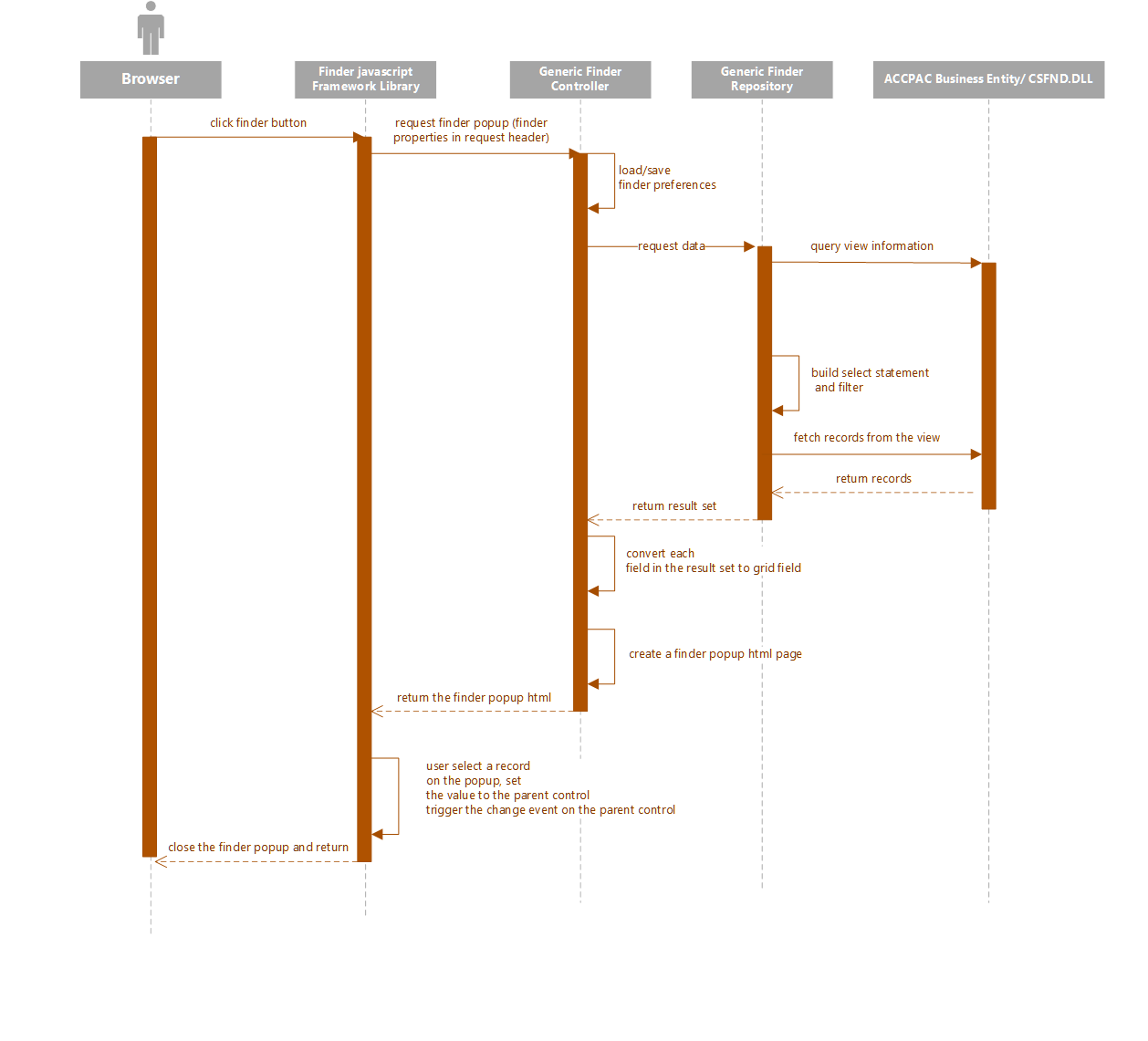
* Customer Finder
  + 24 or 25 default columns based upon whether multicurrency is enabled
    - Lots of other considerations based upon what screen is bringing up the Customer Finder (PO Entry, Order Entry, Shipments, Inquiries, etc.)
  + All columns must be specified and configured since the finder screen allows user to select non-default columns, etc.
    - And the Customer view has over 150 columns
  + Upon read of data from Accpac Business View, all 150+ columns must be mapped to the Customer MVC model even though only a subset are being displayed in the finder grid

So, as you can see, the invocation of a finder for AR Customer will have to instantiate a service layer, which instantiates a business repository layer, which performs initializations, caching and the like to read the Business View, which must map every field for every row returned in order to display the selected columns in the finder’s grid. And, upon selection, that Customer that was selected will need to be read properly once the screen is informed that a record was selected by the finder in which time it does the Service, Business Repository and mapping all again.

* 1. Dynamic Finders

Because of the framework used to implement web finders, it is not possible to simply or dynamically create a finder. This is a frequent request by partners.

1. Workflow Diagram

****

1. Overview

The new finder has been completely re-imagined addressing the pain points discussed in the Introduction section:

* Minimal Code
* Elimination of complex and redundant code
* Customization
* Dynamic creation
* Efficient

It is now possible to invoke a finder with a single line of JavaScript code. However, most implementations will consist of several lines of code.

The existing web finder will continue working and will remain unless it is manually replaced

* 1. New Finder Files
     1. …Common.Plugin.ViewFinder.js

The main JavaScript class which contains the viewFinderHelper which is used to setup the finder on a screen.

* + 1. …Common.Plugin.ViewFinderProperties.js

The JavaScript class containing the Sage defined finders.

Third party Partners and ISVs will need to create a JavaScript library where they define their finders and then bundle this with their application.

Third party customizations can dynamically define their customizations or include them in the customization JavaScript.

The 2019.2 version will have a minimal amount of definitions and the number of definitions will be expanded in future releases.

* + 1. …Common.ViewFinderGrid.js

The JavaScript class supporting the finder screen.

1. Interface
   1. Old Interface

|  |
| --- |
| ***sg.setFinder****(id, searchFinder, onSelectCallBack, onCancelCallBack, title, filters, uid, postbackNotRequired, height, top)* |

Though the interface doesn’t look too complicated, a developer must implement the ***onSelectCallBack*** function to manually put the values to the destination control, trigger the change event, implement ***filters*** to define a filter, construct the finder ***title***, etc.   
  
And that is just the mandatory work on the client.

The brunt of the work lies on the server side. A developer is responsible for implementing a controller, a repository, a service, and registering all those interfaces and services in the bootstrapper.

* 1. New interface

A minimalistic approach is our guiding principle with the new finder. As the finder’s sole purpose is to lookup a record from an underlying business entity, our goal is to require as little information from a developer as possible.

For this reason, we have eliminated all the redundant elements from the old finder such as:

* + title
    - There is little reason to ask a developer to supply the title if we can derive it from the underlying business entity.
    - The new finder will build the title
  + onSelectCallBack
    - There is no reason to require a developer to grab a returned value from the finder to the place in the control on the UI and trigger subsequent actions.
    - The new finder simply requires the parent control and will can do this automatically when a record is selected.
  + Individual controller, repository, and service interface for each finder
    - The is no reason to have to create controllers, repositories and services if only finders are required.
    - The new finder eliminates this requirement.

But a minimalistic approach is not enough to design a good interface if the design is not flexible. Therefore, an extensible design pattern has been adopted in the new finder. This was achieved by adding properties to the finder on an as-needed basis to accommodate various business needs and/or deficiencies in some of the business entities.

|  |  |  |
| --- | --- | --- |
| ***sg.viewFinderHelper.setViewFinder***(***id***, ***parent***, ***finderProperties***, ***onCancelCallback***, ***height***, ***top***) | | |
| **Description** | Set the finder for use | |
| **Parameters** | ***id*** | (Required) The id of the finder button |
|  | ***parent*** | (Required) The parent can either be the id of the parent control (textbox associated with finder) or a callback function.  If it is the parent control, the finder will set the selected value from the finder into this control and then will trigger the ‘change’ event, which forces the screen to execute the standard and existing logic to retrieve the record for screen population.  If it is a callback function, the finder will invoke the function if a record is selected and passes the selected value(s) to the function. The developer is now responsible for what the screen does |
|  | ***finderProperties*** | (Required) The finderProperties can either be a callback function or an object that defines the finder properties.  The following properties are available:   * **viewID** (Required)   + Sets the ROTO ID for the finder use (i.e. AP0006) * **viewOrder** (Required)   + Sets which index to use on the business entity (i.e. 0) * **displayFieldNames** (Required)   + Sets the fields to be displayed   + These fields will also appear in the dropdown list of the popup for further filtering. (i.e. ["ACCTSET", "TEXTDESC", "SWACTV", "DATEINACTV", "DATELASTMN", "CURRCODE"]) * **returnFieldNames** (Required)   + Sets the field(s) to be returned (i.e. ["ACCTSET"]) * **initKeyValues** (Optional)   + Sets the initial values for the key field(s). If omitted, the finder search from the beginning of the table. (i.e. [“USA”] ) * **filter** (Optional)   + Sets a hidden filter where the finder will further restrict the search scope in additional to the filter specified in the finder screen (i.e. “DELETE = 0”) * **parentValAsInitKey** (Optional)   + Default to true   + When set to true, the value in the parent control will be used as the initial key/search value.     - This only works for a single key business entity and the parent parameter should be a control, not a callback function. * **optionalFieldBindings** (Optional)   + When set, optional fields will be included in the filter (i.e. “AP0407,AP0500[0]” * **ReinterpretInitKeyValues** (Optional)   + Default to true   + When set to true     - Sets the initial key values to the ACCPAC business entity     - Gets the value back     - Uses the returned value to construct the filter to calculate page count. * **url** (Optional)   + Sets the path to override the default entry point.   + The URL array contains three elements: module, controller, and action. (i.e. url: ["CS", "TaxGroupViewFinder", "Find"]).   + Used when the finder requires logic before initial load (i.e. showing/hiding the multicurrency column depending on the user’s company) |
|  | ***onCancelCallback*** | (Optional) A callback function if the developer wishes to be notified when the cancel button is selected |
|  | ***height*** | (Optional) The developer may specify the height of the finder screen |
|  | ***top*** | (Optional) the developer may specify the top location for the finder screen |
| **Returns** | None | |
| **Example** | See Example section | |

1. Examples

The following section will illustrate a several different implementations that illustrate the flexibility in implementation

* 1. AP Distribution Sets

initFinders: function () {

// Setup finder with simple binding to parent control and manually creating configuration

sg.viewFinderHelper.setViewFinder("btnFinderDistributionSet", "txtDistributionSet",

{

viewID: "AP0009",

viewOrder: 0,

displayFieldNames: ["DISTSET", "TEXTDESC", "SWACTV", "DATEINACTV"],

returnFieldNames: ["DISTSET"],

filter: null,

initKeyValues: [],

parentValAsInitKey: true

});

},

* 1. IC Price List Code

/\*\*

\* @name initFinders

\* @description Initialize the finder(s)

\*/

initFinders: function () {

// Gets the predefined configuration for the IC Price Codes Finder

// All display fields, return field, view id, etc. have already been setup

var props = sg.viewFinderProperties.IC.PriceListCodes;

// Identifies the finder button on the screen

var buttonId = "btnPriceListCodefinder";

// Identifies the callback routine on success since the screen wants control

var onOkCallback = onFinderSuccess.priceListCode;

// Identifies the callback routine on a cancel since the screen wants to be notified

var onCancelCallback = onFinderCancel.priceListCode;

// Creates a filter for the finder

var filter = sg.finderHelper.createDefaultFunction("txtPriceListCode", "PriceListCodeName", sg.finderOperator.StartsWith);

// Creates a finderProperties object for the finder based upon definition and modifications to definition

var initFinder = function (viewFinder) {

viewFinder.viewID = props.viewID;

viewFinder.viewOrder = props.viewOrder;

viewFinder.displayFieldNames = props.displayFieldNames;

viewFinder.returnFieldNames = props.returnFieldNames;

viewFinder.filter = filter;

};

// Sets up the finder for the Price Code screen

sg.viewFinderHelper.setViewFinder(buttonId, onOkCallback, initFinder, onCancelCallback);

// On Finder success bind the data to appropriate controls

var onFinderSuccess = {

pricelist Code: function (result) {

// Set the Price List Code as a filter value

if (result != null) {

var selectedCode = result.PRICELIST;

priceListCodeUI.ModelData.Data.PriceListCodeName(selectedCode);

priceListCodeUtility.checkIsDirty(priceListCodeUIData.getPriceListCodeById, selectedCode);

}

}

};

// On cancel Finder the focus should go to appropriate control

var onFinderCancel = {

priceListCode: function () {

sg.controls.Focus($("#txtPriceListCode"));

$("#message").empty();

}

};

* 1. AR Refund Entry

// Get customer and document from the model

var customerNumber = jobDetailsUI.jobDetailsModelData.Data.CustomerNumber();

var documentNumber = jobDetailsUI.jobDetailsModelData.Data.DocumentNumber();

// Gets the predefined configuration for the AR Open Document Details Finder

var finderProperties = sg.viewFinderProperties.AR.OpenDocumentDetails;

// Sets the initKeyValues and filter properties

finderProperties["initKeyValues"] = [customerNumber, documentNumber, $("#txtOriginalLineNumber").val()];

finderProperties["filter"] = jQuery.validator.format(finderProperties.filterTemplate,

customerNumber, documentNumber);

// Setup the finder

sg.viewFinderHelper.setViewFinder("btnOriginalLineNumber",

sg.utls.grid.onFinderSuccess.bind(finderProperties.returnFieldNames[0]), finderProperties,

sg.utls.grid.onFinderCancel);

* 1. PO Purchase Order Entry

// Setup the finder

sg.viewFinderHelper.setViewFinder("btnFinderVendorNo",

finderSuccess.vendorFinder,

sg.viewFinderProperties.AP.Vendor,

finderCancel.vendorFinder,

sg.finderHelper.createDefaultFunction("txtVendorNumber", "VendorNumber"));

// Success

vendorFinder: function (data) {

if (data != null) {

var vendorNumber = data.VENDORID;

purchaseOrderEntryUI.purchaseOrderEntryModel.Data.Vendor(vendorNumber);

purchaseOrderEntryUI.headerEventTypeEnum = purchaseOrderEntryEnum.HeaderEventType.Vendor;

purchaseOrderEntryRepository.RefreshHeader(purchaseOrderEntryUI.purchaseOrderEntryModel.Data.Vendor(),

purchaseOrderEntryEnum.HeaderEventType.Vendor);

sg.controls.Focus($("#txtTemplate"));

}

},

// Cancel

vendorFinder: function () {

sg.controls.Focus($("#txtVendorNumber"));

},

* 1. Employee Finder

// Employee Finder

var employeeProperties = payrollType === PayrollTypes.Canada ? sg.viewFinderProperties.PR.CAEmployee : sg.viewFinderProperties.PR.USEmployee;

// Setup finder

sg.viewFinderHelper.setViewFinder("btnEmployeeNumberFinder",

function (result) { //OnSuccess

if (result) {

employeeTimecardUI.employeeTimecardModel.Data.Employee(result[employeeProperties.returnFieldNames[0]]);

employeeTimecardUI.employeeTimecardModel.EmployeeName(result[employeeProperties.returnFieldNames[1]]);

if (employeeTimecardUI.employeeTimecardModel.Data.EndDate()) { //Try retrieving the timecard

employeeTimecardUI.checkIsDirty();

}

}

},

employeeProperties,

$.noop);

},

1. Future Considerations

* Optional Field page count is always 0.
  + This is a preexisting defect in the CSFND view that has yet to be addressed. The desktop finder doesn’t need to show page count if optional field is included so it always returns 0.
* Ability to launch a finder programmatically instead of clicking a button.
  + This is a request from partners.
* Create a composite edit control that can be customized to show label, edit box, tab button, and finder button.
  + This will greatly simplify the razor views for a UI as well as improving customization experience.
  + Currently, to show a typical edit control on a web page (like the one below), a developer has to define 4 different ids for each element, and write 4 lines of Razor view code. This not only makes the code more complex, it also makes customization harder as the customization has to manage 4 separate components instead of just one for this control.  
      
    As the new finder treats the parent control (the edit box) as an intrinsic part of a finder, it should much easier to implement a composite edit control using the new finder.

