Sage 300 Web Screens SDK

Integrating the Navigation Control

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. Introduction 4](#_Toc101868688)

[2. The Razor View 5](#_Toc101868689)

[2.1 New Composite Helper SgFinderFor 5](#_Toc101868690)

[2.2 Textbox Helper for Finder 5](#_Toc101868691)

[3. The Behaviour JavaScript 6](#_Toc101868692)

[3.1 Add variable to store navigation action 6](#_Toc101868693)

[3.2 Bind the navigation handler in the InitButton routine 6](#_Toc101868694)

[3.3 Pass navigation action in the get routine 6](#_Toc101868695)

[3.4 Reset navigation action in the check is dirty function 7](#_Toc101868696)

[3.5 In the UI success object’s get function, set navigation action 8](#_Toc101868697)

[4. The Repository JavaScript 9](#_Toc101868698)

[4.1 In the get function, set the navigation action 9](#_Toc101868699)

[5. The Public Controller 10](#_Toc101868700)

[5.1 In the get method, set the navigation action 10](#_Toc101868701)

[6. The Internal Controller 11](#_Toc101868702)

[6.1 In the get method, set the navigation action (Flat Views) 11](#_Toc101868703)

[6.2 In the get method, set the navigation action (new Header/Detail Views) 13](#_Toc101868704)

[7. The Business Repository Interface 14](#_Toc101868705)

[7.1 In the get method, set the navigation action (new Header/Detail Views) 14](#_Toc101868706)

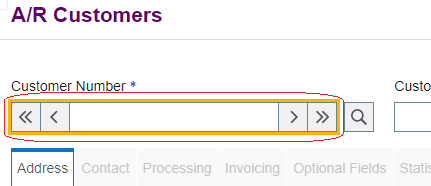
[8. The Business Repository 15](#_Toc101868707)

[8.1 In the get method, set the navigation action (new Header/Detail Views) 15](#_Toc101868708)

1. Introduction

Sage 300 2022 introduced an optional navigation component (first, previous, next, last) (also known as VCR buttons) for web screen fields associated with a finder.

Not all web screens require this functionality while the most relevant web screens have been modified to include this navigation component.



This document will address implementing this navigation component in the various C#, CSHTML, and JavaScript files.

1. The Razor View

The screen’s Razor View will be modified to set the “**includeNavigationButtons**” parameter to **true**.

* 1. New Composite Helper SgFinderFor

@Html.SgFinderFor(m => m.Data.DocumentNumber,

                  new { @sagevalue = "Data.DocumentNumber", @sagedisable = "InquiryOnly" },

                  new { @id = "txtDocumentNumber", @maxlength = "16", @class = "txt-upper" },

**includeNavigationButtons: true)**

* 1. Textbox Helper for Finder

@Html.KoSageTextBoxFor(model => model.Data.CustomerNumber, new { @sagevalue = "Data.CustomerNumber", @disable = "IsDisableFinder", @valueUpdate = "'input'" }, new { @class = "txt-upper default" }**, true**)

The new standard is to NOT display the Go or Load button with a finder. The composite helper SgFinderFor does not display the Go/Load button while the “old” or individual helpers may still specify this button. When implementing the Navigation Control, it is important to remove this button if it exists.

For example:

**@\***@Html.KoSageButton("btnLoadAccSetCode", null, new { @id = "btnLoad", @class = "icon btn-go", @tabindex = "-1" })**\*@**

1. The Behaviour JavaScript

The screen’s behaviour JavaScript will be modified to support the various navigation actions.

* 1. Add variable to store navigation action

var customerUI = {

**navigationAction: sg.utls.NavigationAction.None**,

…

* 1. Bind the navigation handler in the InitButton routine

initButtons: function () {

**// Navigation control support**

**sg.utls.bindingNavigationActions(customerUI, 'oldCustomerNumber',**

**'Data\_CustomerNumber', 0, 'customerUtility.checkIsDirty',**

**'customerUIData.getcustomerById');**

…

**bindingNavigationActions Parameters**

**customerUI** is the JavaScript UI Object

**oldCustomerNumber** is the key field object variable (to keep track of previous value)

**Data\_CustomerNumber** is the id of the textbox

**0** is the navigation group index if there are multiple navigation controls (default = 0)

**customerUtility.checkIsDirty** is the check is dirty function name (default = ‘checkIsDirty’)

**customerUIData.getcustomerById** is the get function name (default = ‘get’)

If there is a grid, the 7th parameter is the grid name (default = “”)

* 1. Pass navigation action in the get routine

var customerUIData = {

getcustomerById: function () {

sg.utls.clearValidations("frmCustomer");

var id = customerUI.navigationAction == 0 ? customerUI.customerNumber : $("#Data\_CustomerNumber").val().toUpperCase();

customerRepository.getById(id**, customerUI.navigationAction**);

},

…

* 1. Reset navigation action in the check is dirty function

checkIsDirty: function (yesFuncionToCall, noFunctionToCall) {

var vm = customerUI.customerViewModel;

if ((vm.IsKoCustomerDirty && vm.IsKoCustomerDirty.isDirty() && customerUI.customerNumber && customerUI.oldCustomerNumber ) || (vm.IsKoCommentDirty.isDirty() || vm.IsKoContactFormDirty.isDirty() || vm.IsKoStatisticsDirty.isDirty())) {

sg.utls.showKendoConfirmationDialog(

function () { // Yes

yesFuncionToCall.call();

},

**function () { // No**

**if (customerUI.navigationAction = sg.utls.NavigationAction.None) {**

**noFunctionToCall.call();**

**} else {**

**sg.utls.resetNaviControlStatusFocus(customerUI, 0, true);**

**}**

},

$.validator.format(globalResource.SaveConfirm, customerResources.CustomerNumber, customerUI.oldCustomerNumber.toUpperCase()));

} else {

yesFuncionToCall.call();

}

},

**resetNaviControlStatusFocus Parameters**

**customerUI** is the JavaScript UI Object

**0** is the navigation group index if there are multiple navigation controls (default = 0)

**true** is the reset flag (default = false)

If there is a grid, the 4th parameter is the grid name(default = “”)

* 1. In the UI success object’s get function, set navigation action

var customerUISuccess = {

onGetByIdCompleted: function (result) {

…

**// Navigation control support**

**if (customerUI.navigationAction !== sg.utls.NavigationAction.None) {**

**sg.utls.naviControlStatusFocus(customerUI, 'oldCustomerNumber', 'Data\_CustomerNumber');**

**customerUI.oldCustomerNumber = customerUI.customerNumber;**

**}**

…

},

**naviControlStatusFocus Parameters**

**customerUI** is the JavaScript UI Object

**oldCustomerNumber** is the key field object variable (to keep track of previous value)

**Data\_CustomerNumber** is the id of the textbox

If there are multiple navigation controls, the 4th parameter is the group index (default = 0)

If there is a grid, the 5th parameter is the grid name (default = “”)

1. The Repository JavaScript

The screen’s repository JavaScript will be modified to support the various navigation actions.

* 1. In the get function, set the navigation action

var customerRepository = customerRepository || {};

customerRepository = {

// Ajax call to get the selected Customer data

getById: function (id**, navigationAction**) {

var data = { 'id': id**, 'navigationAction': navigationAction** };

customerAjax.post("GetById", data, customerUISuccess.onGetByIdCompleted);

},

…

1. The Public Controller

The screen’s public controller will be modified to support the various navigation actions.

* 1. In the get method, set the navigation action

/// <summary> Get Customer by id </summary>

/// <param name="id">customer number</param>

**/// <param name="navigationAction">Navigation Action</param>**

/// <returns>Customer</returns>

[HttpPost]

public virtual JsonNetResult GetById(string id**, NavigationAction navigationAction = NavigationAction.None**)

{

try

{

return JsonNet(ControllerInternal.GetById(id**, navigationAction**));

}

catch (BusinessException businessException)

{

return JsonNet(BuildErrorModelBase(CommonResx.GetFailedMessage, businessException, ARCommonResx.Customer));

}

}

1. The Internal Controller

The screen’s internal controller will be modified to support the various navigation actions.

* 1. In the get method, set the navigation action (Flat Views)

/// <summary> Get Customer data specific to customer number </summary>

/// <param name="id">Customer Number</param>

**/// <param name="navigationAction">Navigation Action</param>**

/// <returns>Customer View Model</returns>

internal CustomerViewModel<THeader> GetById(string id**, NavigationAction navigationAction = NavigationAction.None**)

{

var customerViewModel = new CustomerViewModel<THeader>();

SessionHelper.Remove(\_commentCacheKey);

SessionHelper.Remove(\_optFieldCacheKey);

SessionHelper.Remove(\_contactFormCacheKey);

Expression<Func<THeader, bool>> filter = null;

if (!string.IsNullOrEmpty(id))

{

filter = param => param.CustomerNumber == id;

}

**var customerModel = base.GetById(id, navigationAction, filter);**

**//var customerModel = Service.GetById(id);**

The **base.GetById** routine in the **BaseExportImportControllerInternal** class will now invoke the appropriate method in the service/repository classes. See base class routine below:

/// <summary>

/// Get entity record set by navigation action and filter

/// </summary>

/// <param name="id">Key field value</param>

/// <param name="navigationAction">Navigation action</param>

/// <param name="filter">Filter applied to view for select record</param>

/// <param name="firstOrDefault">Navigate to firstOrDefault when key is empty</param>

/// <returns></returns>

public virtual T GetById(string id, NavigationAction navigationAction = NavigationAction.None, Expression<Func<T, bool>> filter = null, bool firstOrDefault = false)

{

T data = default(T);

switch (navigationAction)

{

case NavigationAction.None:

data = Service.GetById(id);

break;

case NavigationAction.First:

data = Service.FirstOrDefault();

break;

case NavigationAction.Previous:

data = string.IsNullOrEmpty(id) ? (firstOrDefault ? Service.FirstOrDefault() : null) : Service.Previous(filter);

break;

case NavigationAction.Next:

data = string.IsNullOrEmpty(id) ? Service.FirstOrDefault() : Service.Next(filter);

break;

case NavigationAction.Last:

data = Service.FirstOrDefault(null, new OrderBy { SortDirection = SortDirection.Descending });

break;

}

return data;

}

* 1. In the get method, set the navigation action (new Header/Detail Views)

A new pattern has been established for header – detail screens that go straight to a repository (like the new PJC screens). Therefore, modify the internal controller’s get method like the following example:

/// <summary>

/// Get a Charge

/// </summary>

/// <param name="id">Id for Charge</param>

**/// <param name="navigationAction">Navigation Action</param>**

/// <returns>JSON object for Charge</returns>

internal ChargesViewModel GetById(string id**, NavigationAction navigationAction = NavigationAction.None**)

{

var data = \_repository.GetById(id**, navigationAction**);

var userMessage = new UserMessage(data);

return GetViewModel(data, userMessage);

}

1. The Business Repository Interface

The screen’s business repository interface will be modified to support the various navigation actions ***if the screen is a new header – detail screen that goes directly to the repository (like the new PJC screens) as opposed to the service***.

* 1. In the get method, set the navigation action (new Header/Detail Views)

/// <summary>

/// Get a Charge record by id

/// </summary>

/// <typeparam name="TKey"></typeparam>

/// <param name="id">primary key value</param>

**/// <param name="navigationAction">Navigation Action</param>**

/// <returns>ChargeModel</returns>

Charge GetById<TKey>(TKey id**, NavigationAction navigationAction = NavigationAction.None**);

1. The Business Repository

The screen’s business repository will be modified to support the various navigation actions ***if the screen is a new header – detail screen that goes directly to the repository (like the new PJC screens) as opposed to the service***.

* 1. In the get method, set the navigation action (new Header/Detail Views)

/// <summary>

/// Get Charge based on the primary key

/// </summary>

/// <typeparam name="TKey">data type of the primary key</typeparam>

/// <param name="id">key value</param>

**/// <param name="navigationAction">Navigation Action</param>**

/// <returns>Charge</returns>

public Charge GetById<TKey>(TKey id**, NavigationAction navigationAction = NavigationAction.None**)

{

CheckRights(\_chargeEntity, SecurityType.Inquire);

\_chargeEntity.Order = 1;

var hasIdValue = (id != null && !string.IsNullOrEmpty(id.ToString()));

if (hasIdValue)

{

\_chargeEntity.SetValue(Charge.Fields.ChargeNumber, id.ToString(), true);

if (navigationAction == NavigationAction.None)

{

if (!\_chargeEntity.Read(false))

{

CreateNewRecord();

\_chargeEntity.SetValue(Charge.Fields.ChargeNumber, id.ToString(), true);

return \_chargeMapper.Map(\_chargeEntity);

}

}

}

else

{

\_chargeEntity.Top();

}

**NavigateEntity(\_chargeEntity, navigationAction, id?.ToString());**

return !\_chargeEntity.Read(false) ? null : \_chargeMapper.Map(\_chargeEntity);

}