Sage 300 Web Screens SDK

AR Invoice Entry Customization Tutorial

February 2018

The MIT License (MIT)

Copyright © 2018 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 4](#_Toc506833552)

[2. AR Invoice Entry Screen 5](#_Toc506833553)

[3. Creating the Customization 7](#_Toc506833554)

[3.1 Standalone Customization Wizard 7](#_Toc506833555)

[3.1.1 Step 1 – Add/Edit a Customization 7](#_Toc506833556)

[3.1.2 Step 2 – Add/Edit Screens 9](#_Toc506833557)

[3.1.3 Step 3 – Add/Edit Controls 11](#_Toc506833558)

[3.1.4 Step 4 – Generate Customization Files 16](#_Toc506833559)

[3.1.5 Step 5 – Generated Customization Files Complete 17](#_Toc506833560)

[4. JavaScript Modifications 18](#_Toc506833561)

[4.1 Control Initialization 18](#_Toc506833562)

[4.1.1 initButtons Function 18](#_Toc506833563)

[4.1.2 initTextbox Function 19](#_Toc506833564)

[4.1.3 initDropDownList Function 19](#_Toc506833565)

[4.1.4 initDatePicker Function 20](#_Toc506833566)

[4.1.5 initFinders Function 20](#_Toc506833567)

[4.1.6 initGrid Function 20](#_Toc506833568)

[4.2 init Function 21](#_Toc506833569)

[4.3 refreshGrid Function 22](#_Toc506833570)

[4.4 add Function 22](#_Toc506833571)

[4.5 Callback functions 24](#_Toc506833572)

[4.5.1 finderSuccess Callback 24](#_Toc506833573)

[4.5.2 populateDropdown Callback 24](#_Toc506833574)

[4.5.3 setGridDataSource Callback 25](#_Toc506833575)

[4.5.4 save Callback 25](#_Toc506833576)

[4.5.5 update Callback 25](#_Toc506833577)

[4.5.6 delete Callback 26](#_Toc506833578)

[5. Packaging the Customization 27](#_Toc506833579)

[5.1 Manifest.json Contents 27](#_Toc506833580)

[5.2 OrderEntry\_Settings.xml Contents 27](#_Toc506833581)

[6. Customized AR Invoice Entry Screen 29](#_Toc506833582)

1. Overview

This document is intended to serve as a tutorial for creating a Sage 300 Web Screen Customization for the AR Invoice Entry Screen. This customization sample is included in the Web SDK (samples\customization\Sample4\_AR\_Invoice\_Entry\_Customization).

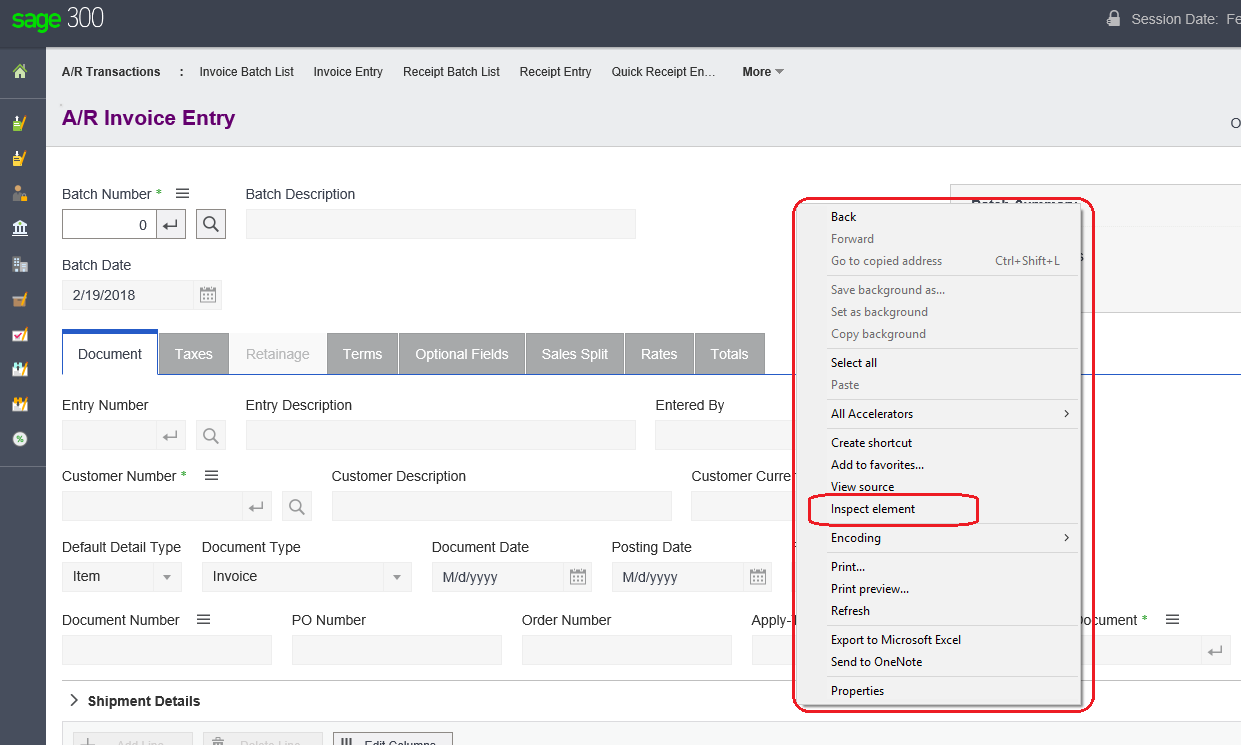
This customization will highlight the following topics:

* Adding a tab control and controls to the tab control
* Enhancing the display (via CSS) of some newly added controls
* Populating a dropdown list and grid
* Invoking Finder, Get, Save and Delete functions on the newly added controls

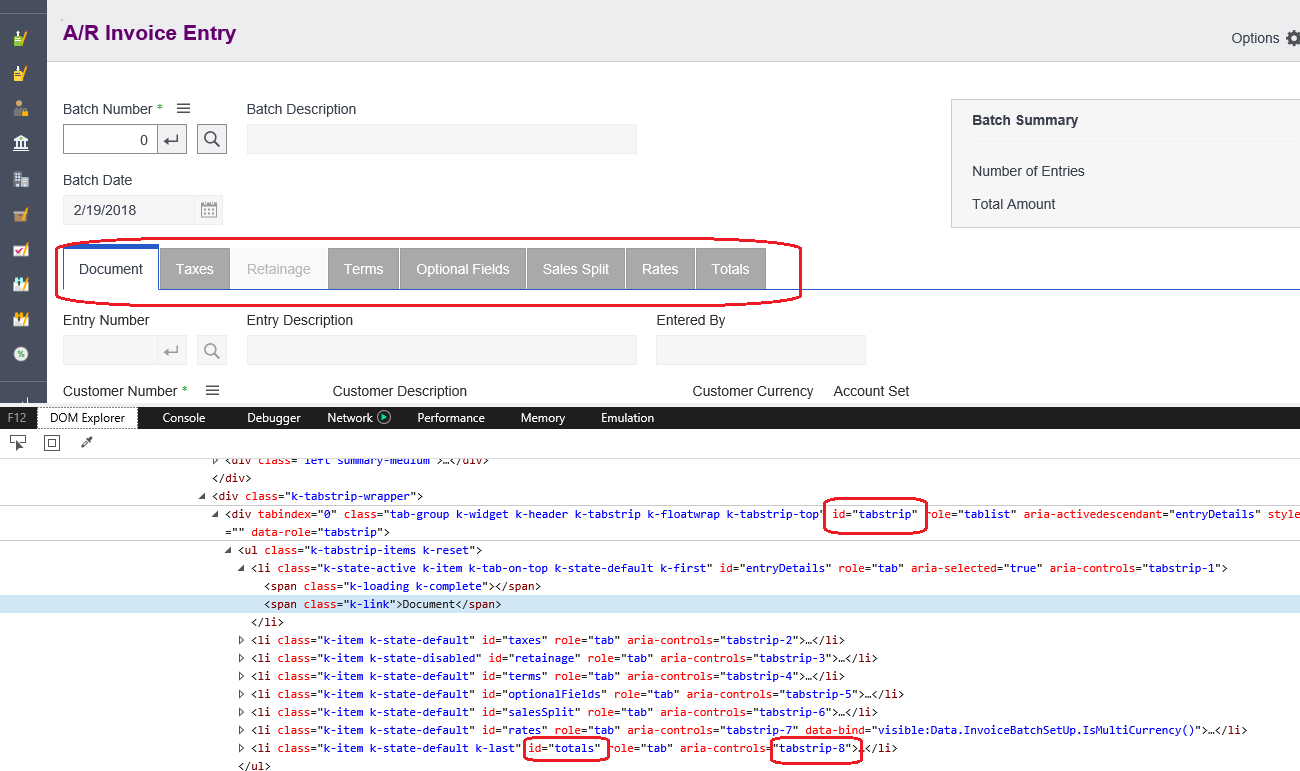
1. AR Invoice Entry Screen

The AR Invoice Entry screen is the web screen that will be customized in this tutorial. Since this customization will be adding a new tab page to the web screen, we must know the name of the existing tab control and where in the tab control we will place our new tab page.

To find the element ID of the tab control, bring up the AR Invoice Entry screen and use your browser’s **Inspect element** feature (right-click and select **Inspect element**).



To find the element ID for the tab control and tab page, right click the tab control and select **Inspect element**:



We can now inspect the element and determine that the tab control is named “**tabstrip**” the page we want to add our tab page after is “**tabstrip-8**”.

We now have enough information from the AR Invoice Entry screen is create the customization manifest and related files before writing any code.

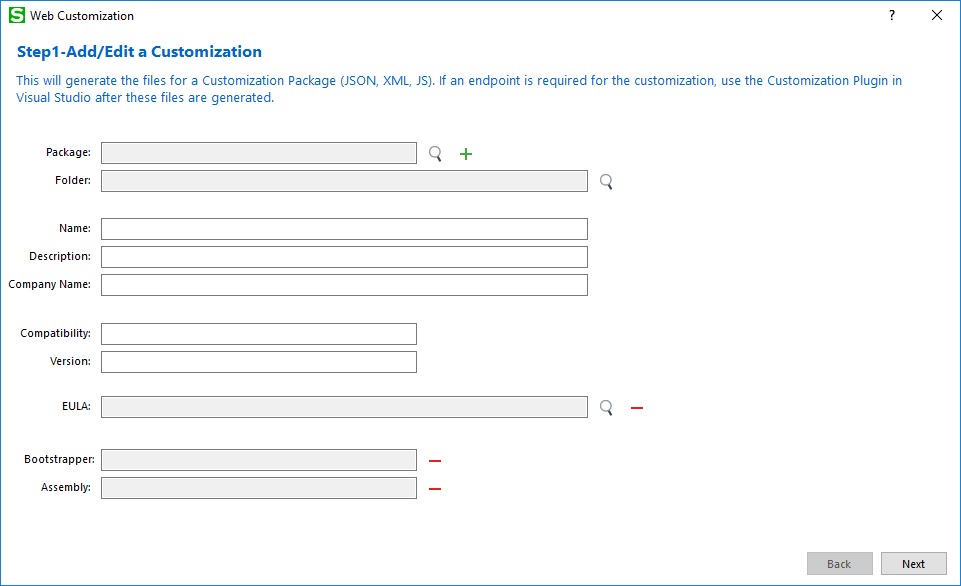
1. Creating the Customization
   1. Standalone Customization Wizard

Create the JSON, XML and JavaScript files for the AR Invoice Entry Customization.

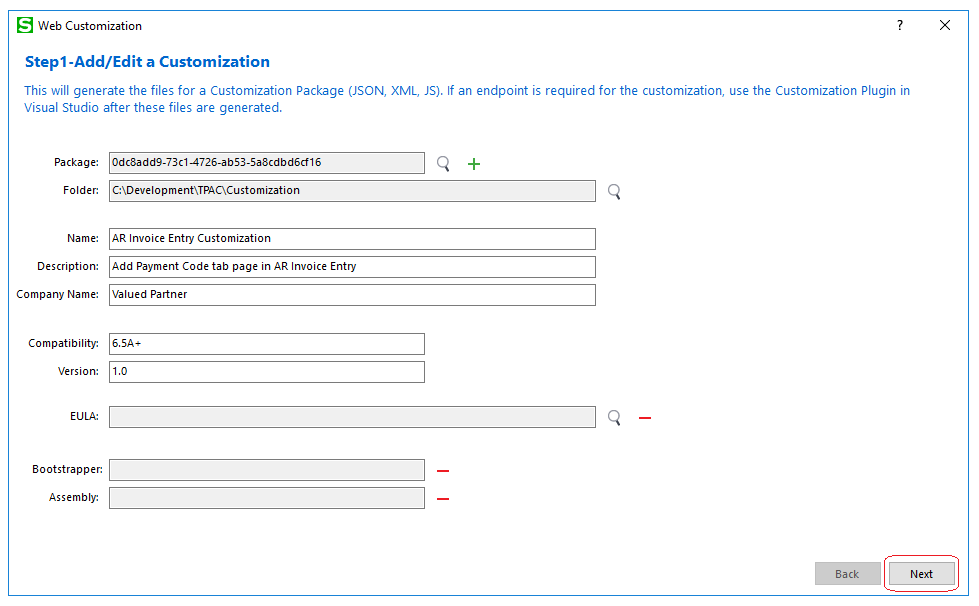
Locate the standalone customization wizard (located in the SDK’s bin folder (bin\Sage.CA.SBS.ERP.Sage300.CustomizationWizard.exe).

* + 1. Step 1 – Add/Edit a Customization

Create a new customization for the AR Invoice Entry screen.

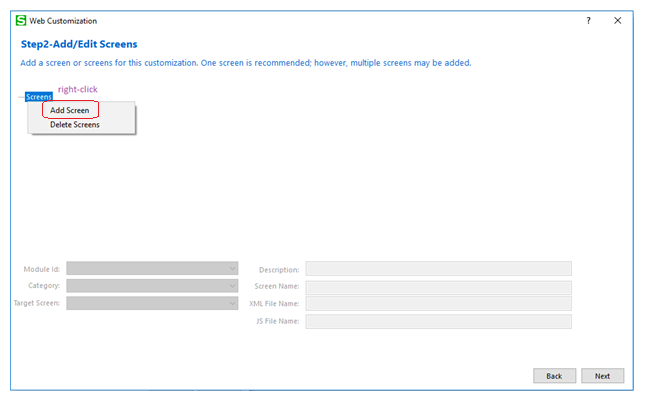


1. Click the green plus button to generate a GUID for the new customization.
2. Populate the remaining fields
   1. Select a folder name by clicking the magnifying glass of the Folder field to indicate where this customization will reside.
   2. Enter **AR Invoice Entry Customization** for the Name field.
   3. Enter **Add Payment Code tab page in AR Invoice Entry** for the Description field
   4. Enter **Valued Partner** for the Company Name field
   5. Verify **6.5A+** for the Compatibility field
   6. Verify **1.0** for the Version field

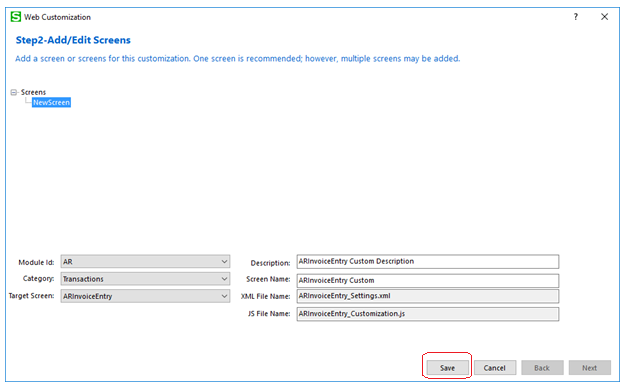


1. Click **Next** to proceed.
   * 1. Step 2 – Add/Edit Screens

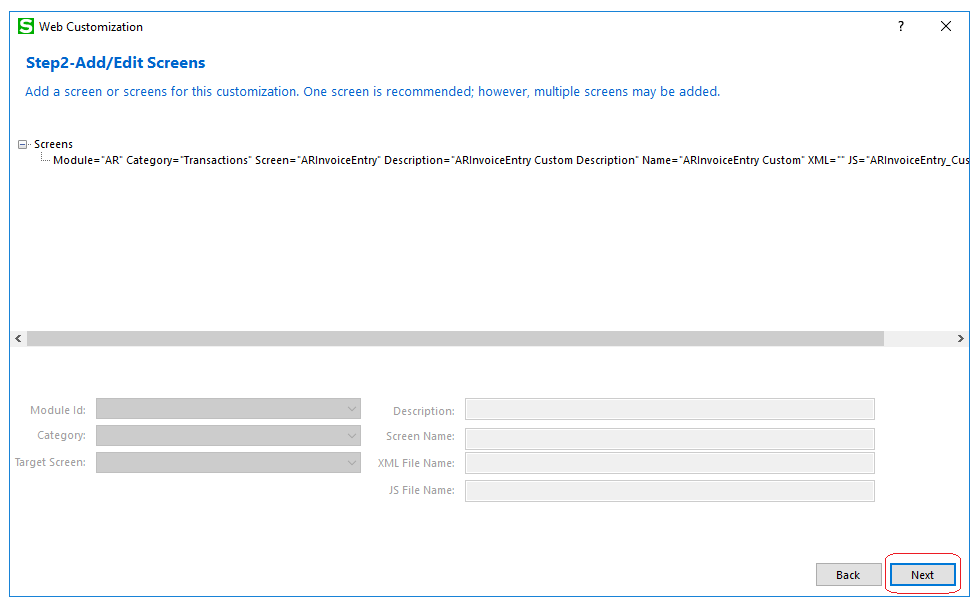
Add the AR Invoice Entry screen for the customization.



1. Right click the **Screens** node for options and select **Add Screen**
2. Select **AR** for the Module Id field
3. Select **Transactions** for the Category field
4. Select **ARInvoiceEntry** for the Target Screen field
5. Verify **ARInvoiceEntry Custom Description** for the Description field
6. Enter **ARInvoiceEntry Custom** for the Screen Name field



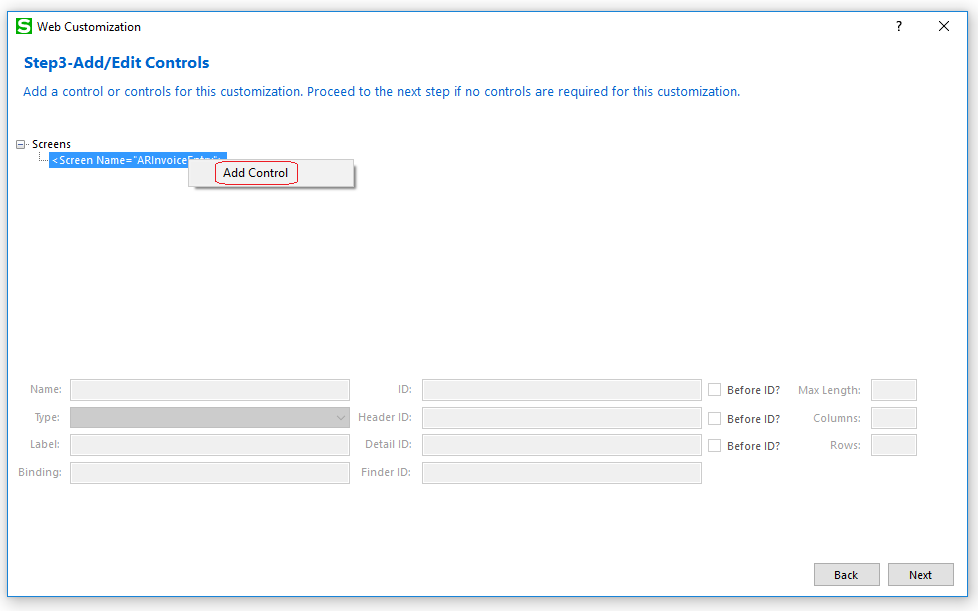
1. Click **Save** to save the screen



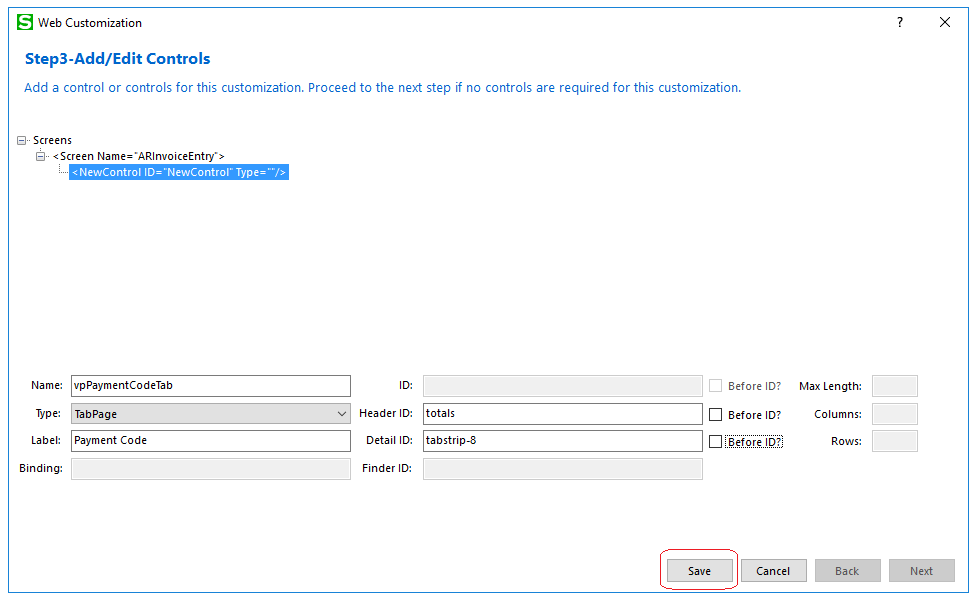
1. Click **Next** to proceed
   * 1. Step 3 – Add/Edit Controls

Add the new controls for the customization.

New Control 1: The tab page will be added after the **totals** tab page discovered when inspecting the AR Invoice Entry screen



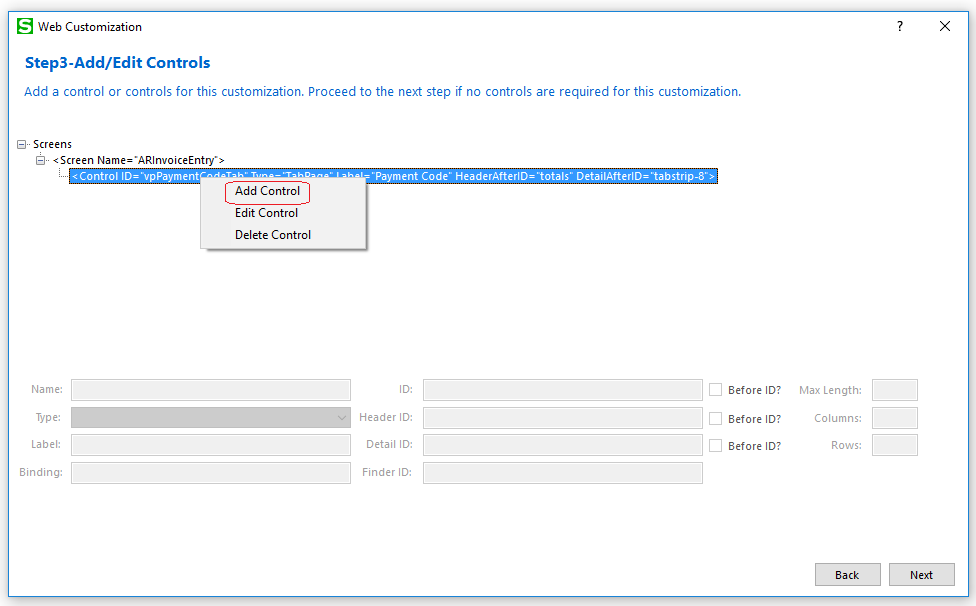
1. Right click the screen for **ARInvoiceEntry** and select **Add Control**
2. Enter **vpPaymentCodeTab** for the Name field
3. Select **TabPage** for the Type field
4. Enter **Payment Code** for the Label field
5. Enter **totals** for the Header ID field
6. Enter **tabstrip-8** for the Detail ID field

****

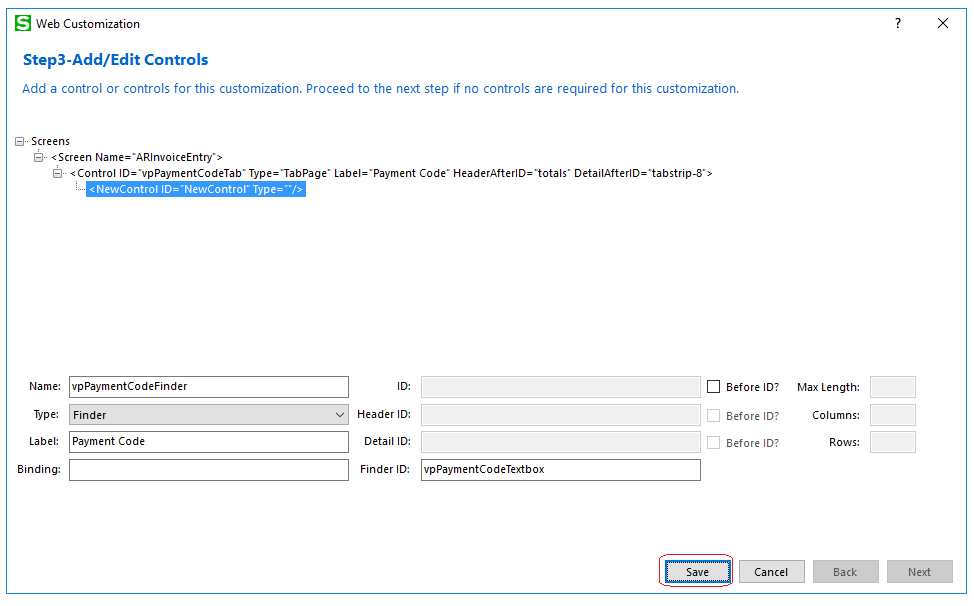
1. Click **Save** to save the control

New Control 2: Add controls to the newly added **Tab Page.**

**Note:** The controls on a new container control do not need to specify a location.



1. Right click the control for **vpPaymentCodeTab** and select **Add Control**
2. Enter **vpPaymentCodeFinder** for the Name field
3. Select **Finder** for the Type field
4. Enter **Payment Code** for the Label field
5. Enter **vpPaymentCodeTextbox** for the Finder ID field

****

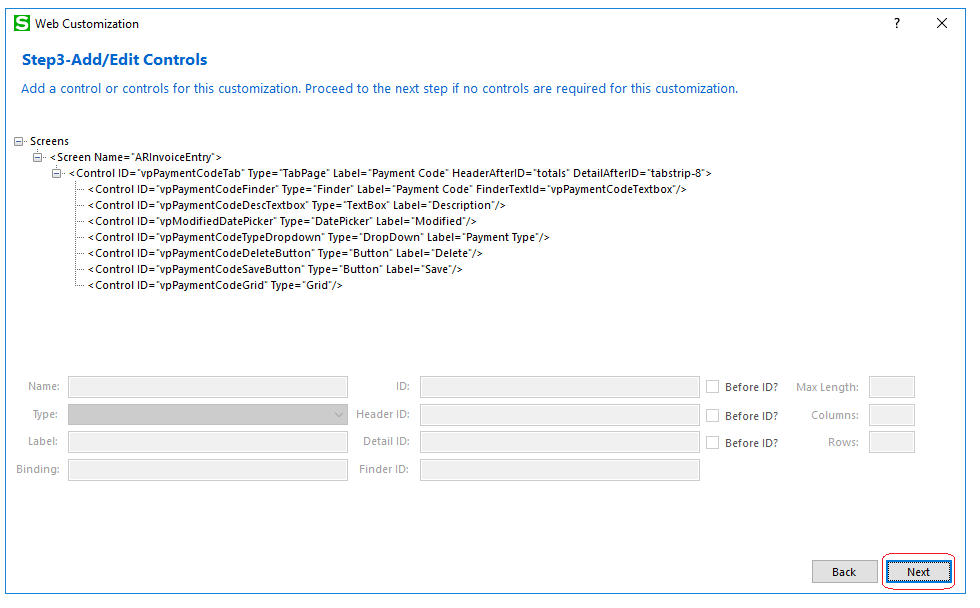
1. Click **Save** to save the control

New Controls 3-8: Add remaining controls to the newly added **Tab Page.**

The previous step just added a control to the newly created tab page. The remaining controls will be added to the tab page in the same way by Right clicking the control for **vpPaymentCodeTab** and select **Add Control.**

The controls are listed here and Click **Save** for each control to be added to the tab page.

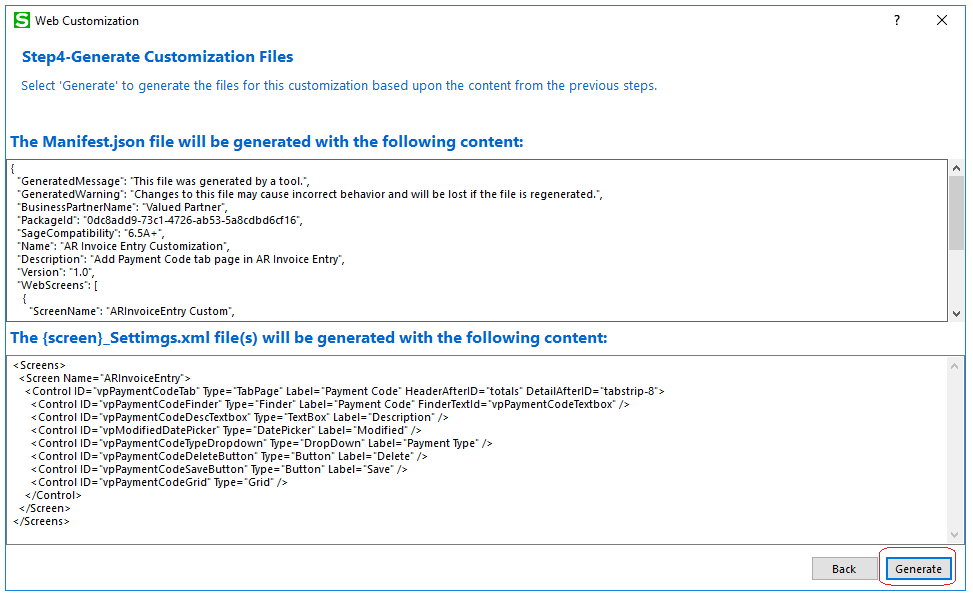
* Name="vpPaymentCodeDescTextbox" Type="TextBox" Label="Description"
* Name="vpModifiedDatePicker" Type ="DatePicker" Label="Modified"
* Name="vpPaymentCodeTypeDropdown" Type ="DropDown" Label="Payment Type"
* Name="vpPaymentCodeDeleteButton" Type="Button" Label="Delete"
* Name="vpPaymentCodeSaveButton" Type="Button" Label="Save"
* Name="vpPaymentCodeGrid" Type="Grid"



All controls have been added, Click **Next** to proceed.

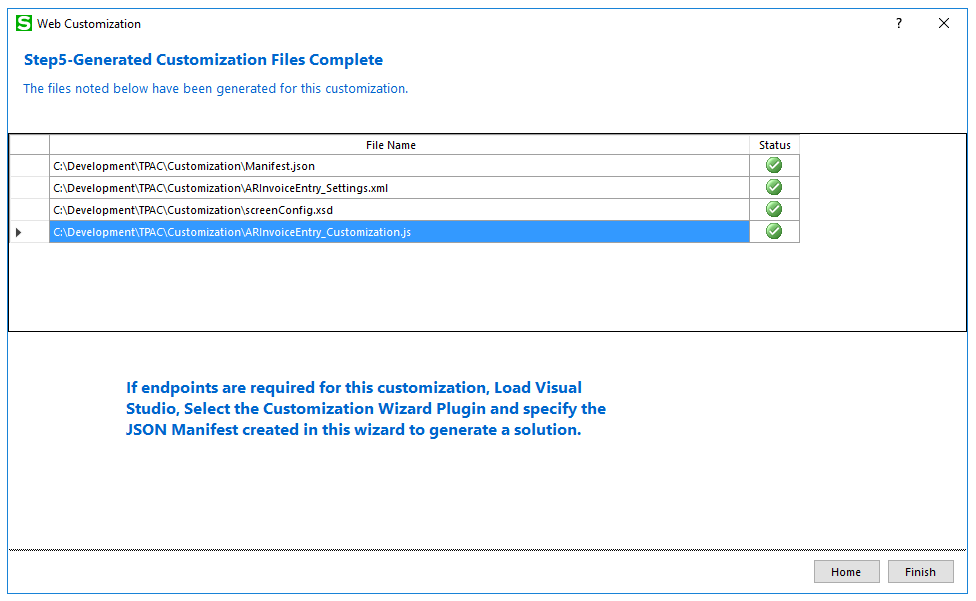
* + 1. Step 4 – Generate Customization Files

Preview the manifest before creation of customization.



1. If there are any corrections to be made, press **Back** to return to the previous step otherwise press **Generate** to generate the customization files.
   * 1. Step 5 – Generated Customization Files Complete

Files have been generated!



* Manifest
  + **Manifest.json**
* Control Settings file
  + **ARInvoiceEntry\_Settings.xml**
* JavaScript file
  + **ARInvoiceEntry\_Customization.js**
* XML Schema file
  + **ScreenConfig.xsd**

This customization does not require an endpoint and we will now start adding JavaScript code to our generated JavaScript template.

1. JavaScript Modifications

The standalone customization wizard generated the JavaScript for this customization.

However, the generated file is only a template and now requires modifications specific to this customization.

Load the **ARInvoiceEntry\_Customization.js** in the editor of your choice to perform the modifications in the following sections.

* 1. Control Initialization

Some Kendo UI controls require initialization in JavaScript.

* + 1. initButtons Function

Add the following code snippet in the **initButtons** function to add some styling and the click events for the Save and Delete buttons:

// This is an example of adding some space after the newly added Delete button

$("#vpPaymentCodeDeleteButton").css('margin-right', 20);

// Initialize the Delete button's click event

$("#vpPaymentCodeDeleteButton").on('click', function (e) {

// Get value fom Payment Code textbox

var value = $("#vpPaymentCodeTextbox").val();

// If a value is in the textbox, issue a call to the delete method

if (value) {

// Assign url (module, controller, action) to delete

var url = sg.utls.url.buildUrl("AR", "PaymentCode", "Delete");

// Assign model for parameter(s) of Delete action

var model = { id: value };

// Post delete and specify call back

sg.utls.ajaxPost(url, model, ValuedPartnerARInvoiceEntryCustomizationUICallback.delete);

}

})

// Initialize the Save button's click event

$("#vpPaymentCodeSaveButton").on('click', function (e) {

// Get value fom Payment Code textbox

var value = $("#vpPaymentCodeTextbox").val();

if (!value) {

return;

}

// Locate the payment code grid

var grid = $("#vpPaymentCodeGrid").data("kendoGrid");

// Get list of payment codes in grid

var list = grid.dataSource.data();

// Filter the list based upon value in present in the textbox

var filteredList = list.filter(function (item) { return item.PaymentCode == value; });

// Determine if the Save will perform an add or update

if (filteredList.length == 0) {

// Not in list and therefore a new code

ValuedPartnerARInvoiceEntryCustomizationUI.add();

} else {

// For update, we will get record first for concurrency check (ETag)

// Assign url (module, controller, action) to get

var url = sg.utls.url.buildUrl("AR", "PaymentCode", "Get");

// Assign model for parameter(s) of Get action

var model = { id: value };

// Post get and specify call back

sg.utls.ajaxPost(url, model, ValuedPartnerARInvoiceEntryCustomizationUICallback.update);

}

})

* + 1. initTextbox Function

Add the following code snippet in the **initTextbox** function to add some styling:

// This is an example of changing the styling on the textbox

$("#vpPaymentCodeTextbox").removeClass("smaller").addClass("default");

// This is an example of starting a new line

$("<br/>").insertBefore($("#vpModifiedDatePicker").parent());

* + 1. initDropDownList Function

Add the following code snippet in the **initDropDownList** function to initialize the control:

// Initialize the dropdown list

$("#vpPaymentCodeTypeDropdown").kendoDropDownList({

dataTextField: "Text",

dataValueField: "Value",

change: function (e) {

var idx = this.selectedIndex;

$("#vpPaymentCodeTypeDropdown").val(this.value());

}

});

* + 1. initDatePicker Function

Add the following code snippet in the **initDatePicker** function to initialize the control:

// Initialize the date picker

$("#vpModifiedDatePicker").kendoDatePicker({});

$('#vpModifiedDatePicker').data("kendoDatePicker").value(new Date());

* + 1. initFinders Function

Add the following code snippet to the **initFinders** function to initialize the finder control:

// Init the finder

sg.finderHelper.setFinder("vpPaymentCodeFinder", sg.finder.PaymentCode,

ValuedPartnerARInvoiceEntryCustomizationUICallback.finderSuccess, null, "Select Payment Code",

sg.finderHelper.createDefaultFunction("vpPaymentCodeTextbox", "PaymentCode", sg.finderOperator.StartsWith), null, true);

* + 1. initGrid Function

Add the following code snippet to the **initGrid** function to initialize the grid control and event when a row is selected:

// Initialize the grid

$("#vpPaymentCodeGrid").kendoGrid({

dataSource: [],

height: 300,

resizable: true,

selectable: true,

reorderable: true,

editable: false,

change: onSelectChange,

columns: [

{

field: "PaymentCode",

title: "Payment Code",

width: "100px"

},

{

field: "Description",

title: "Description",

width: "160px"

},

{

field: "PaymentTypeString",

title: "Payment Type",

width: "120px"

},

{

field: "StatusString",

title: "Status",

width: "120px"

}

]

});

// Establish the OnSelectChange function

function onSelectChange(e) {

// Get the grid control

var grid = $("#vpPaymentCodeGrid").data("kendoGrid");

// Get the selected row

var row = grid.dataItem(grid.select());

// If a row was selected, push values into controls since this is not data bound

if (row) {

$("#vpPaymentCodeTextbox").val(row.PaymentCode);

$("#vpPaymentCodeDescTextbox").val(row.Description);

$('#vpPaymentCodeTypeDropdown').data("kendoDropDownList").text(row.PaymentTypeString);

}

}

* 1. init Function

The **init** function is a routine that initialize Kendo UI controls when the JavaScript is loaded. It is also a place to perform additional initializations as needed.

Add the following code snippet to the **init** function (after the line that invokes the **initOtherControls** function) to populate the dropdown list with payment codes and to refresh the grid:

// Issuing a dummy call to get the Payment Code view model

// in order to populate the dropdown of payment codes

// Assign url (module, controller, action) to get

var url = sg.utls.url.buildUrl("AR", "PaymentCode", "Get");

// Assign model for parameter(s) of Get action

var model = { id: "CASH" };

sg.utls.ajaxPost(url, model, ValuedPartnerARInvoiceEntryCustomizationUICallback.populateDropdown);

// Refresh the grid

ValuedPartnerARInvoiceEntryCustomizationUI.refreshGrid();

* 1. refreshGrid Function

This routine will refresh the grid by borrowing the finder’s **RefreshGrid** method. This is an example of populating a grid based upon how the finder grid is populated.

Add the **refreshGrid** function to the **ValuedPartnerARInvoiceEntryCustomizationUI** object after the **initOtherControls** function:

// Refresh the grid utilizing the finder's RefreshGrid method

refreshGrid: function () {

// Assign url (module, controller, action)

var url = sg.utls.url.buildUrl("Core", "Find", "RefreshGrid");

// Assign the parameters

var postData = { finderOptions: { AdvancedFilter: null, PageNumber: 0, PageSize: 30,

SearchFinder: sg.finder.PaymentCode, SimpleFilter: null } };

// Post and specify call back

sg.utls.ajaxPost(url, postData, ValuedPartnerARInvoiceEntryCustomizationUICallback.setGridDataSource);

},

* 1. add Function

This routine will add a new Payment Code.

Add the **add** function to the **ValuedPartnerARInvoiceEntryCustomizationUI** object after the **refreshGrid** function:

// Add a new code that does not exist in the grid

add: function(){

// Assign url (module, controller, action)

var url = sg.utls.url.buildUrl("AR", "PaymentCode", "Add");

// Assign the parameters

var model = {

DateLastMaintained: new Date(),

Description: $("#vpPaymentCodeDescTextbox").val(),

DisplayIndex: 0,

InactiveDate: new Date(1970, 1, 1),

PaymentCode: $("#vpPaymentCodeTextbox").val(),

PaymentType: $('#vpPaymentCodeTypeDropdown').data("kendoDropDownList").value(),

Status: 1

}

// Post and specify call back

sg.utls.ajaxPost(url, model, ValuedPartnerARInvoiceEntryCustomizationUICallback.save);

},

* 1. Callback functions

The **ValuedPartnerARInvoiceEntryCustomizationUICallback** object is where we will define our call back functions.

* + 1. finderSuccess Callback

Add the following callback to the **ValuedPartnerARInvoiceEntryCustomizationUICallback** object:

// Populate fields based on finder selection, if any

finderSuccess: function (data) {

// Data is not null if a value is selected in the finder

if (data) {

// Populate controls

$("#vpPaymentCodeTextbox").val(data.PaymentCode);

$("#vpPaymentCodeDescTextbox").val(data.Description);

$('#vpPaymentCodeTypeDropdown').data("kendoDropDownList").text(data.PaymentTypeString);

// Set focus to textbox

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

}

},

* + 1. populateDropdown Callback

Add the following callback to the **ValuedPartnerARInvoiceEntryCustomizationUICallback** object:

// Populate the dropdown list from view model

populateDropdown: function (viewModel) {

// GetPaymentType of view model is a dynamic list

var list = viewModel.GetPaymentType;

// Get dropdown control

var dropdown = $("#vpPaymentCodeTypeDropdown").data("kendoDropDownList");

// Set list to the datasource of dropdown

dropdown.setDataSource(list);

// Select the first item

dropdown.select(0);

},

* + 1. setGridDataSource Callback

Add the following callback to the **ValuedPartnerARInvoiceEntryCustomizationUICallback** object:

// Callback from refreshGrid routine

setGridDataSource: function (data) {

// Get grid control

var grid = $("#vpPaymentCodeGrid").data("kendoGrid");

// Assign to datasource of grid

grid.dataSource.data(data.Data);

},

* + 1. save Callback

Add the following callback to the **ValuedPartnerARInvoiceEntryCustomizationUICallback** object:

// Callback from the add routine

save: function (data) {

// If the add was a success

if (data.UserMessage.IsSuccess) {

// Refresh the grid to see the new addition

ValuedPartnerARInvoiceEntryCustomizationUI.refreshGrid();

}

// Show the success message

sg.utls.showMessage(data);

},

* + 1. update Callback

Add the following callback to the **ValuedPartnerARInvoiceEntryCustomizationUICallback** object:

// Callback from save click where it was determined that this is

// an existing record. Thus, it performed a Get and this is the callback from

// that Get with the record from the database (in case someone else just updated)

update: function(viewModel){

// Assign to parameters object for Save call

var model = viewModel.Data;

// Assign values from controls

model.DateLastMaintained = new Date();

model.Description = $("#vpPaymentCodeDescTextbox").val();

model.PaymentType = $('#vpPaymentCodeTypeDropdown').data("kendoDropDownList").value();

// Assign url (module, controller, action)

var url = sg.utls.url.buildUrl("AR", "PaymentCode", "Save");

// Post and set callback

sg.utls.ajaxPost(url, model, ValuedPartnerARInvoiceEntryCustomizationUICallback.save);

},

* + 1. delete Callback

Add the following callback to the **ValuedPartnerARInvoiceEntryCustomizationUICallback** object:

// Callback from delete

delete: function (data) {

// If delete was a success

if (data.UserMessage.IsSuccess) {

// Clear controls

$("#vpPaymentCodeTextbox").val(""),

$("#vpPaymentCodeDescTextbox").val(""),

$('#vpModifiedDatePicker').data("kendoDatePicker").value(new Date());

$("#vpPaymentCodeTypeDropdown").data("kendoDropDownList").select(0);

// Refresh the grid

ValuedPartnerARInvoiceEntryCustomizationUI.refreshGrid();

}

// Show message of delete action

sg.utls.showMessage(data);

}

1. Packaging the Customization

Now that the customization is complete, it is time to package the customization files which simply entails zipping the files.

This customization package will be consumed in the Customization Admin screen. This screen will manage the import of the customization files, assignment to the applicable companies, etc.

The customization package must include for this customization:

* The **Manifest.json** file
* The **ARInvoiceEntry\_Settings.xml** file
* The **screenConfig.xsd** file
* The **ARInvoiceEntry\_Customization.js** file
  1. Manifest.json Contents

{

"GeneratedMessage": "This file was generated by a tool.",

"GeneratedWarning": "Changes to this file may cause incorrect behavior and will be lost if the file is regenerated.",

"BusinessPartnerName": "Valued Partner",

"PackageId": "0dc8add9-73c1-4726-ab53-5a8cdbd6cf16",

"SageCompatibility": "6.5A+",

"Name": "AR Invoice Entry Customization",

"Description": "Add Payment Code tab page in AR Invoice Entry",

"Version": "1.0",

"WebScreens": [

{

"ScreenName": "ARInvoiceEntry Custom",

"ScreenDescription": "ARInvoiceEntry Custom Description",

"Category": "Transactions",

"TargetScreen": "ARInvoiceEntry",

"ControlsConfiguration": "ARInvoiceEntry\_Settings.xml",

"ControlsBehavior": "ARInvoiceEntry\_Customization.js",

"Module": "AR"

}

]

}

* 1. OrderEntry\_Settings.xml Contents

<?xml version="1.0" encoding="utf-8"?>

<!-- This file was generated by a tool. -->

<!-- Changes to this file may cause incorrect behavior and will be lost if the file is regenerated. -->

<Screen Name="ARInvoiceEntry" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="screenConfig.xsd">

<Control ID="vpPaymentCodeTab" Type="TabPage" Label="Payment Code" HeaderAfterID="totals" DetailAfterID="tabstrip-8">

<Control ID="vpPaymentCodeFinder" Type="Finder" Label="Payment Code" FinderTextId="vpPaymentCodeTextbox" />

<Control ID="vpPaymentCodeDescTextbox" Type="TextBox" Label="Description" />

<Control ID="vpModifiedDatePicker" Type="DatePicker" Label="Modified" />

<Control ID="vpPaymentCodeTypeDropdown" Type="DropDown" Label="Payment Type" />

<Control ID="vpPaymentCodeDeleteButton" Type="Button" Label="Delete" />

<Control ID="vpPaymentCodeSaveButton" Type="Button" Label="Save" />

<Control ID="vpPaymentCodeGrid" Type="Grid" />

</Control>

</Screen>

1. Customized AR Invoice Entry Screen

When the AR Invoice Entry screen is opened, the customization is applied and the requested controls are now visible in the screen.

