Sage 300 Web Screens SDK

Web Screen Order Entry Customization Tutorial

April 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 5](#_Toc512425075)

[2. Order Entry Screen 6](#_Toc512425076)

[3. Creating the Customization 8](#_Toc512425077)

[3.1 Standalone Customization Wizard 8](#_Toc512425078)

[3.1.1 Step 1 – Add/Edit a Customization 8](#_Toc512425079)

[3.1.2 Step 2 – Add/Edit Screens 9](#_Toc512425080)

[3.1.3 Step 3 – Add/Edit Controls 10](#_Toc512425081)

[3.1.4 Step 4 – Generate Customization Files 19](#_Toc512425082)

[3.2 Plug-in Customization Wizard 20](#_Toc512425083)

[3.2.1 Step 1 – Create a New Project 20](#_Toc512425084)

[3.2.2 Step 2 – Generate a Customization Solution 21](#_Toc512425085)

[4. JavaScript Modifications 24](#_Toc512425086)

[4.1 Kendo Initialization 24](#_Toc512425087)

[4.1.1 initDropDownList Function 24](#_Toc512425088)

[4.1.2 initDatePicker Function 24](#_Toc512425089)

[4.2 Initialization 25](#_Toc512425090)

[4.2.1 Init Function 25](#_Toc512425091)

[4.2.2 initOtherControls Function 25](#_Toc512425092)

[4.2.3 initFinders Function 25](#_Toc512425093)

[4.2.4 initNumericTextBox Function 25](#_Toc512425094)

[4.2.5 Kendo UI Grid, Dropdown and other controls 26](#_Toc512425095)

[4.2.6 initDropdownList Function 26](#_Toc512425096)

[4.3 Intercept existing events and add custom logic 27](#_Toc512425097)

[4.3.1 ajaxSuccess Function 28](#_Toc512425098)

[4.3.2 ajaxComplete Function 29](#_Toc512425099)

[4.4 Add event handler and call back functions 30](#_Toc512425100)

[4.4.1 initButtons Function 30](#_Toc512425101)

[4.4.2 getCustomInfo Function 30](#_Toc512425102)

[4.4.3 Confirmation Dialog 30](#_Toc512425103)

[4.4.4 loadCustomerOrder Function 31](#_Toc512425104)

[4.4.5 Other Functions 31](#_Toc512425105)

[5. Add custom model, view model, and controller 32](#_Toc512425106)

[5.1 Add event handler and call back functions 32](#_Toc512425107)

[5.2 Add custom controller action methods 33](#_Toc512425108)

[6. Packaging the Customization 34](#_Toc512425109)

[6.1 Manifest.json Contents 35](#_Toc512425110)

[6.2 OrderEntry\_Settings.xml Contents 36](#_Toc512425111)

[7. Customized Screen 37](#_Toc512425112)

[7.1 Customized Order Entry screen 37](#_Toc512425113)

[8. Debugging 38](#_Toc512425114)

1. Overview

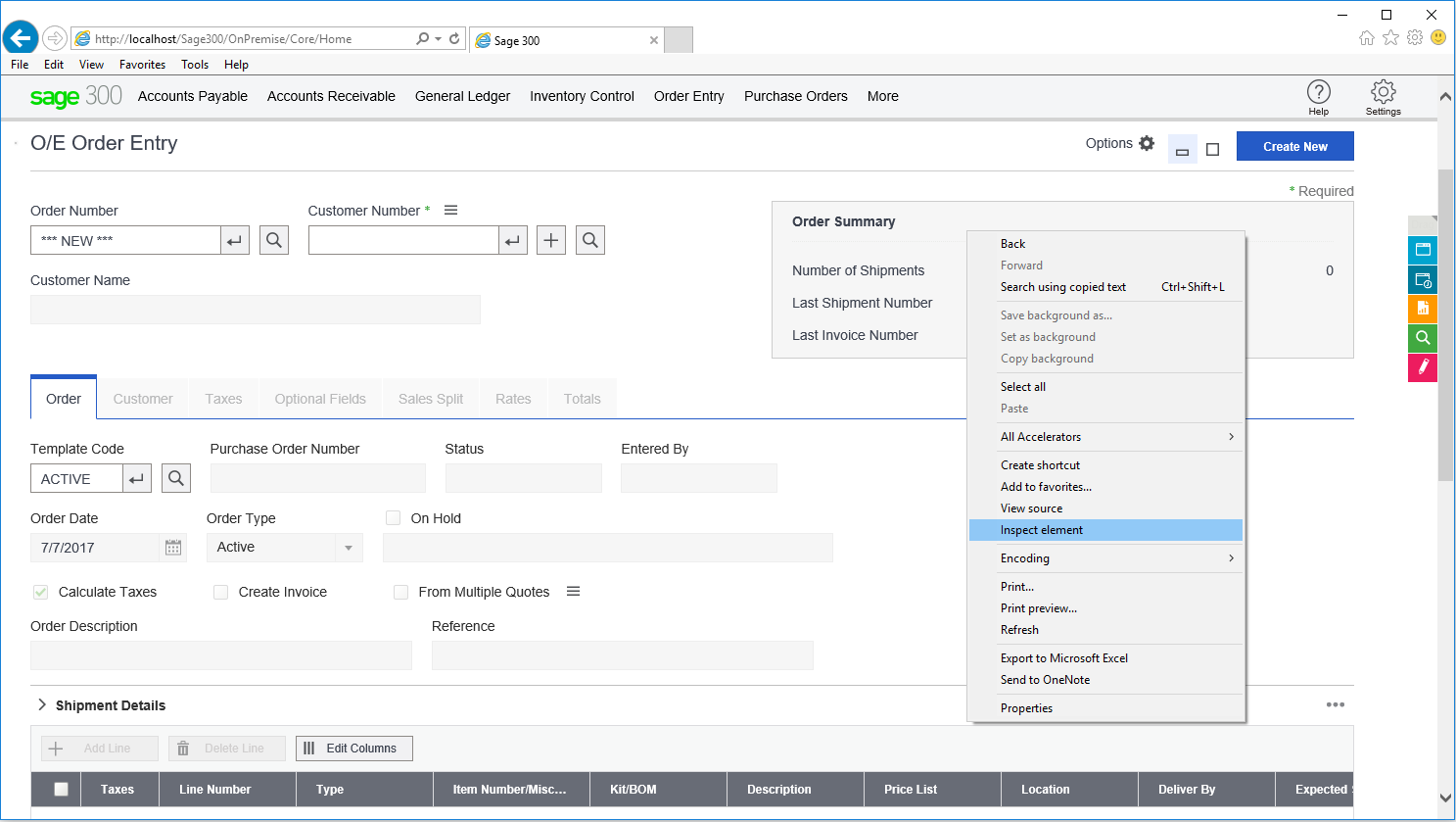
This document is intended to serve as a tutorial for creating a Sage 300 Web Screen Customization for the Order Entry Customization Sample included in the Web SDK (samples\customization\Sample1\_OE\_Order\_Entry\_Customization).

This tutorial will walk you through the process required to create a customization for the Sage 300 Order Entry screen.

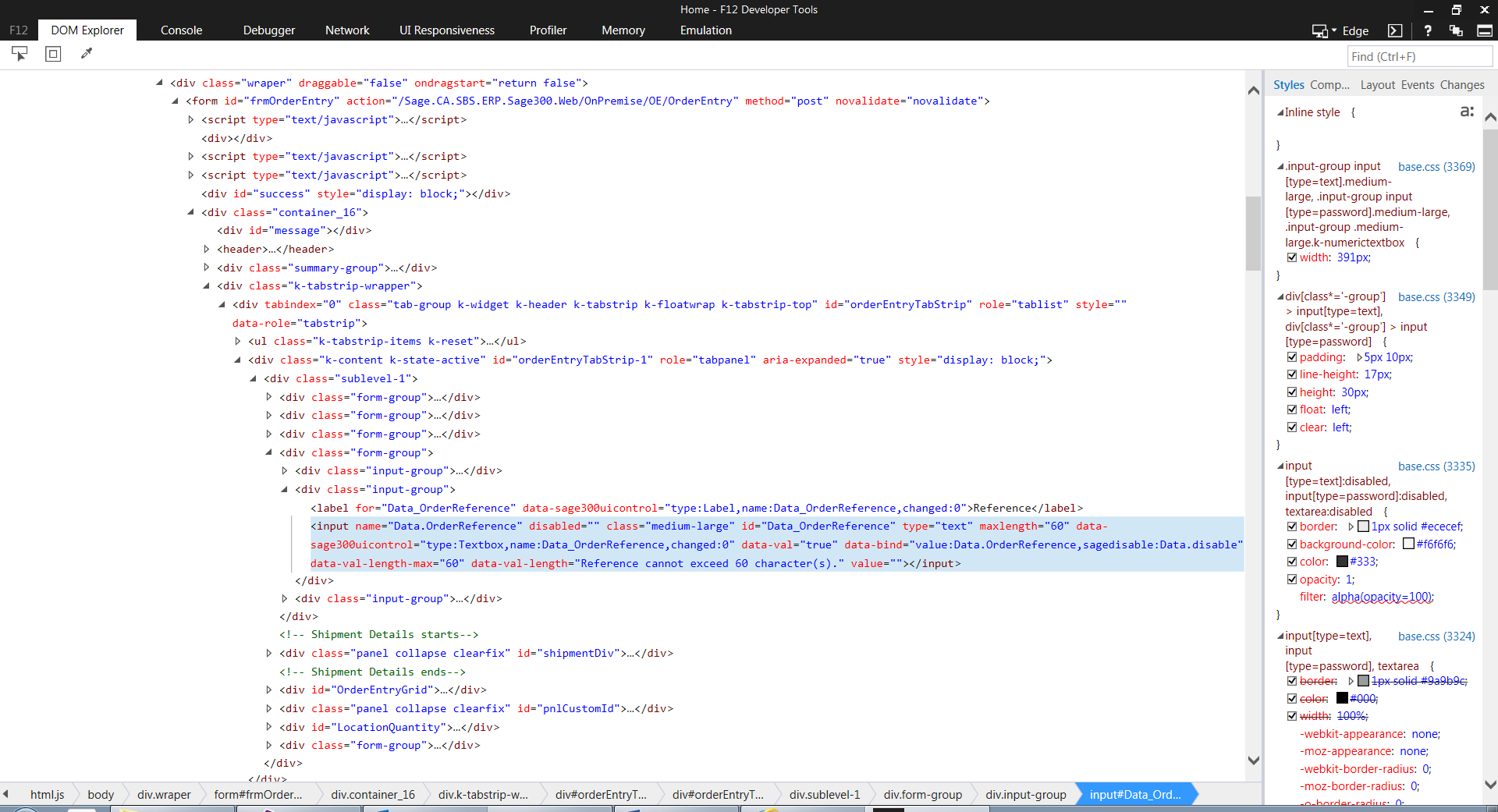
1. Order Entry Screen

Open the O/E Order Entry screen and use your browser’s **Inspect element** feature to find an existing element ID. This element ID is used to insert any new control on the Order Entry screen.

* To find the **Reference** textbox element ID, right click the textbox:



* Select the **Inspect element** context menu
* The F12 (Internet Explorer) window will be displayed with the DOM Explorer page:



* The Reference textbox element ID is Data\_OrderReference.

In the Order Entry sample (and this customization), we will add the following controls:

* A **textbox** after the Data\_OrderReference control and it will be bound to the to the Data.CustomDescription field
* A **button** before the btnPost control
* A **panel** containing several controls after the OrderEntryGrid control
* A **tab** containing several controls

To do this, we will run the Customization Wizard (standalone) now that we know what screen we want to customize and the location for the new controls.

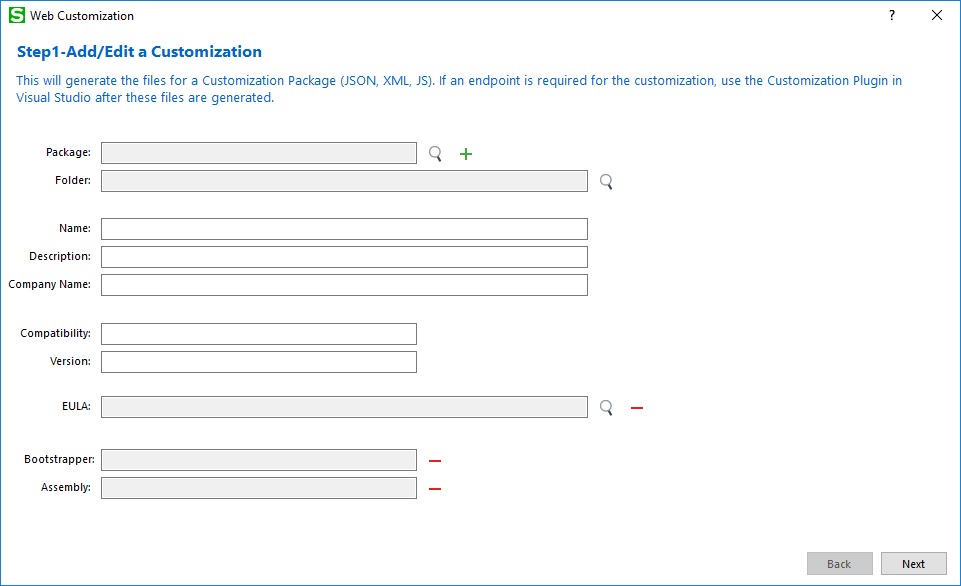
1. Creating the Customization
   1. Standalone Customization Wizard

Creates the JSON, XML and JavaScript files for the Order Entry Customization.

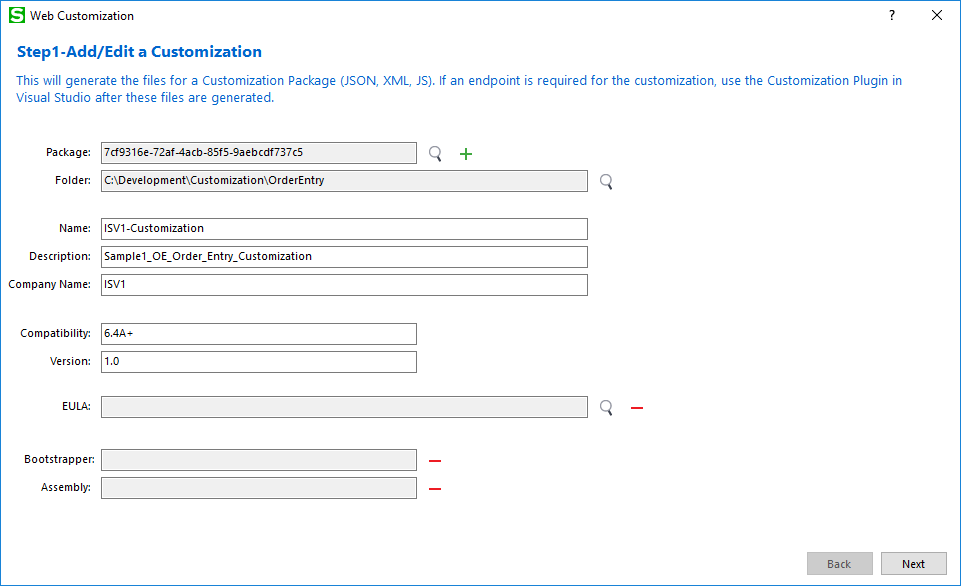
Locate the standalone customization wizard (located in the SDK’s bin folder (bin\Sage.CA.SBS.ERP.Sage300.CustomizationWizard.exe).

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

Create a new Order Entry Customization.

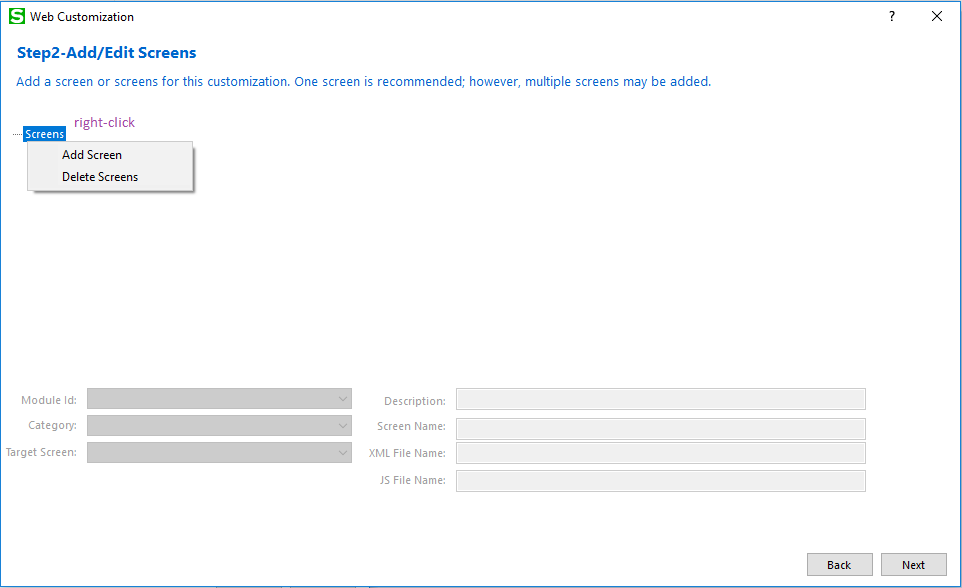


1. Click the green plus button to generate a GUID for a new customization.
2. Populate the fields
   1. Select a folder where this customization will reside.
   2. Enter **ISV1-Customization** for the Name field.
   3. Enter **Sample1\_OE\_Order\_Entry\_Customization** for the Description field
   4. Enter **ISV1** for the Company Name field
   5. Enter **6.4A+** for the Compatibility field
   6. Enter **1.0** for the Version field

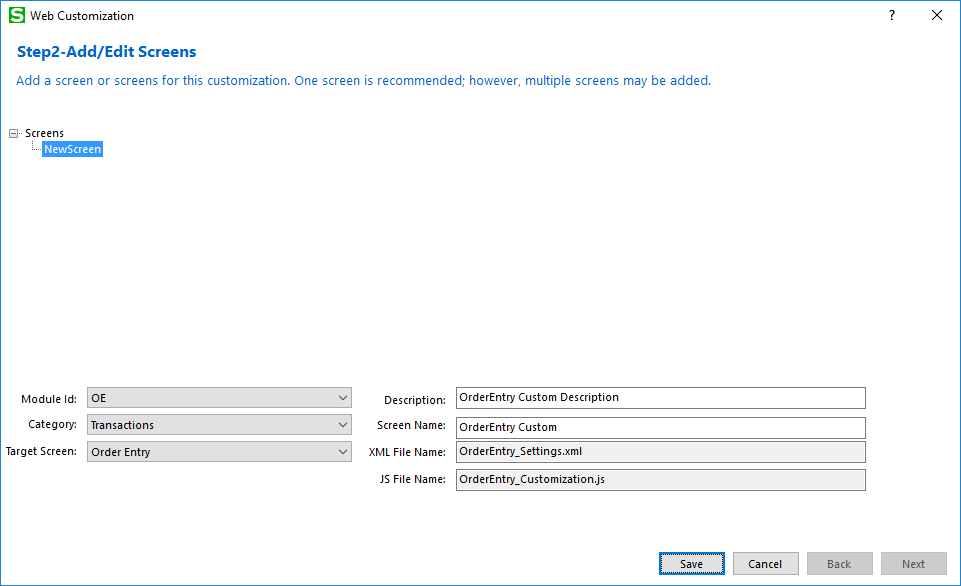


1. Click **Next** to proceed.
   * 1. Step 2 – Add/Edit Screens

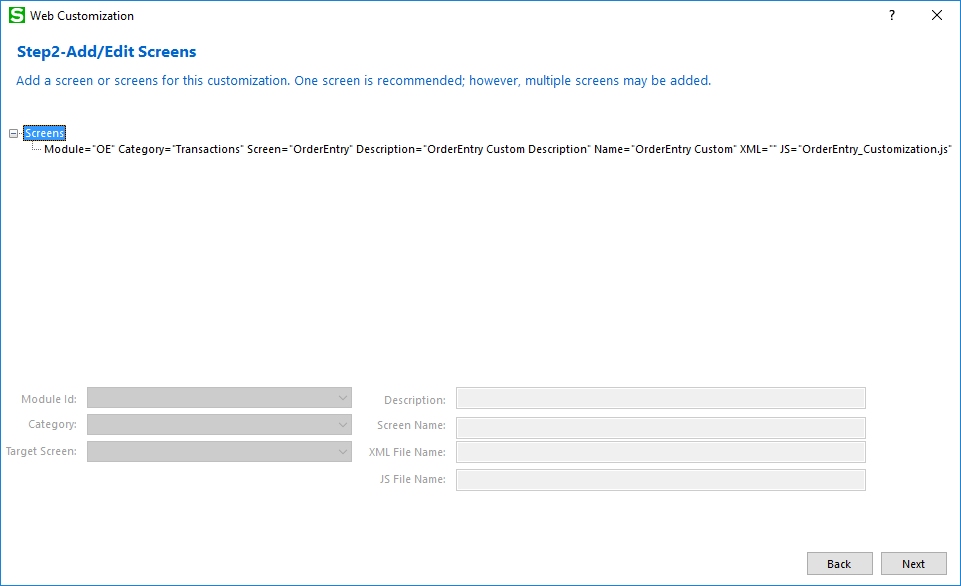
Add the Order Entry screen for the customization.



1. Right click the Screens node for options and select **Add Screen**
2. Select **OE** for the Module Id field
3. Select **Transactions** for the Category field
4. Select **Order Entry** for the Target Screen field
5. Enter **OrderEntry\_Custom Description** for the Description field
6. Enter **OrderEntry\_Custom** for the Screen Name field



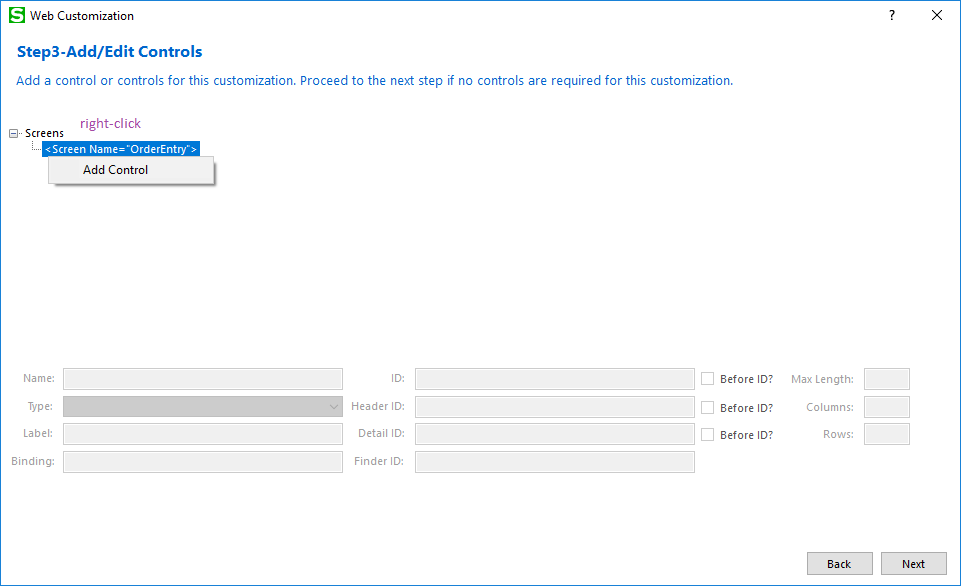
1. Click **Save** to save the screen



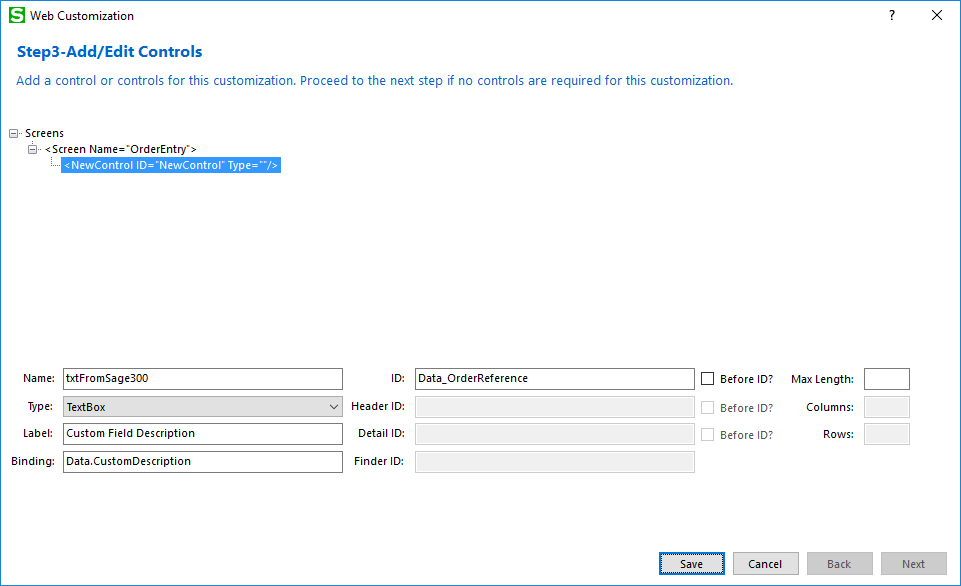
1. Click **Next** to proceed
   * 1. Step 3 – Add/Edit Controls

Add the new controls for the customization.

New Control 1: The textbox after the Data\_OrderReference control and it will be bound to the to the Data.CustomDescription field

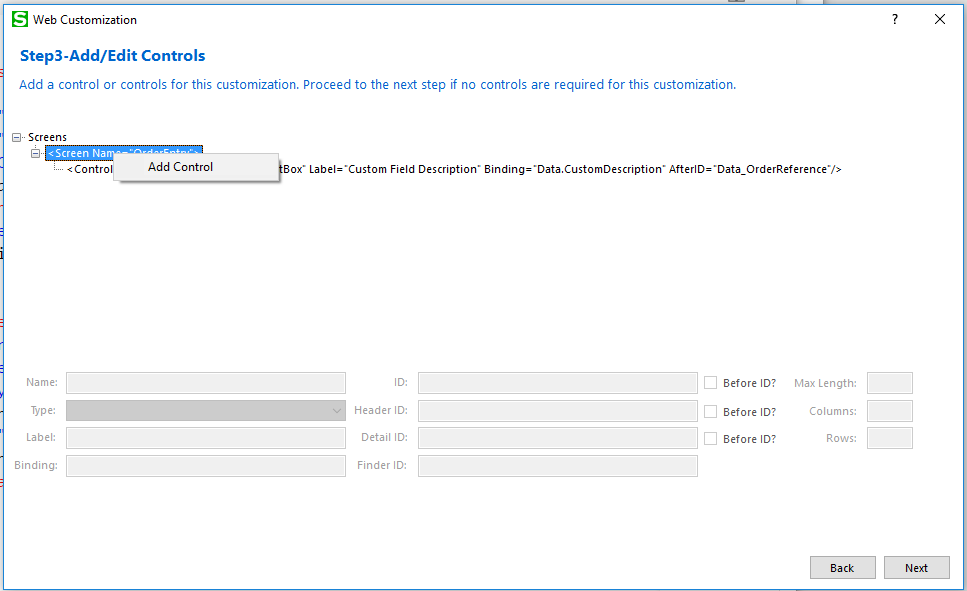


1. Right click the Screen for **OrderEntry** and select **Add Control**
2. Enter **txtFromSage300** for the Name field
3. Select **Textbox** for the Type field
4. Enter **Custom Field Description** for the Label field
5. Enter **Data.CustomDescription** for the binding field
6. Enter **Data\_OrderReference** for the ID field

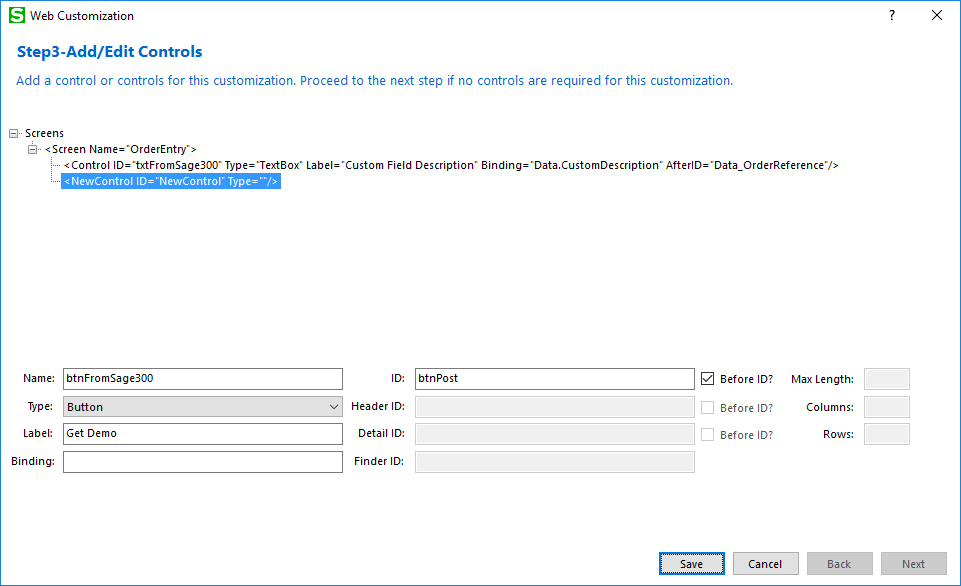


1. Click **Save** to save the control

New Control 2: The **button** before the btnPost control

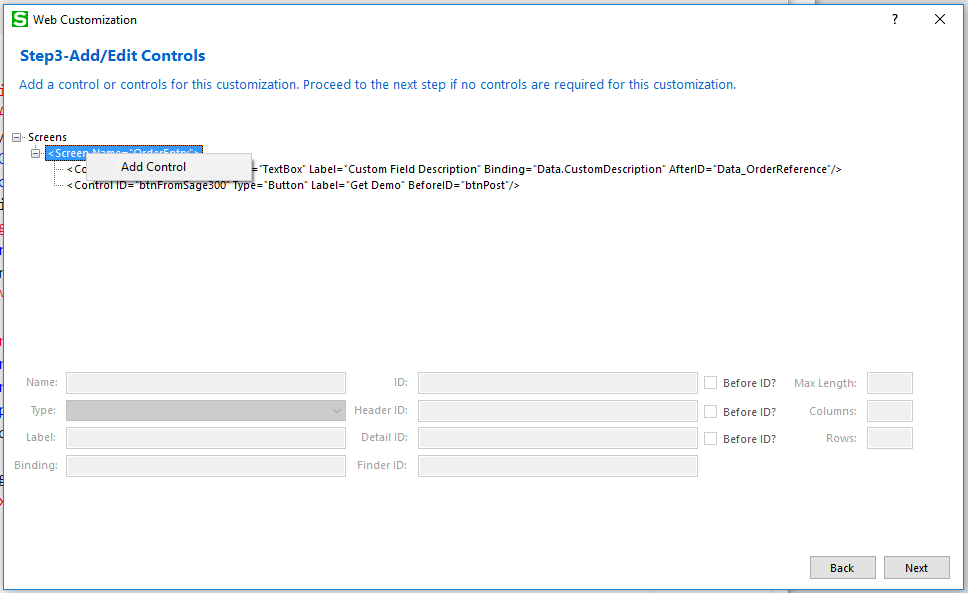


1. Right click the Screen for **OrderEntry** and select **Add Control**
2. Enter **btnFromSage300** for the Name field
3. Select **Button** for the Type field
4. Enter **Get Demo** for the Label field
5. Enter **btnPost** for the ID field
6. **Check** the Before ID field

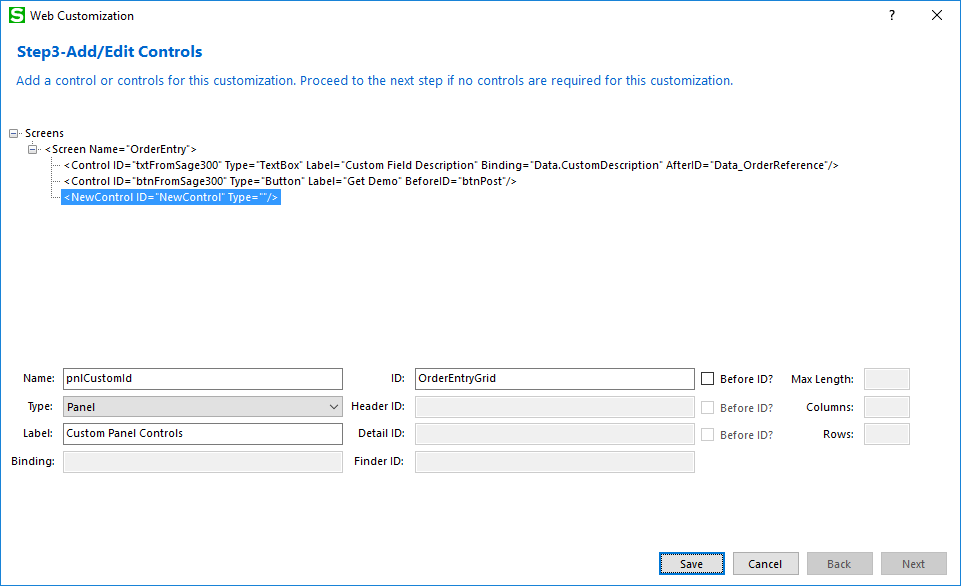


1. Click **Save** to save the control

New Control 3: The **panel** containing several controls after the OrderEntryGrid control



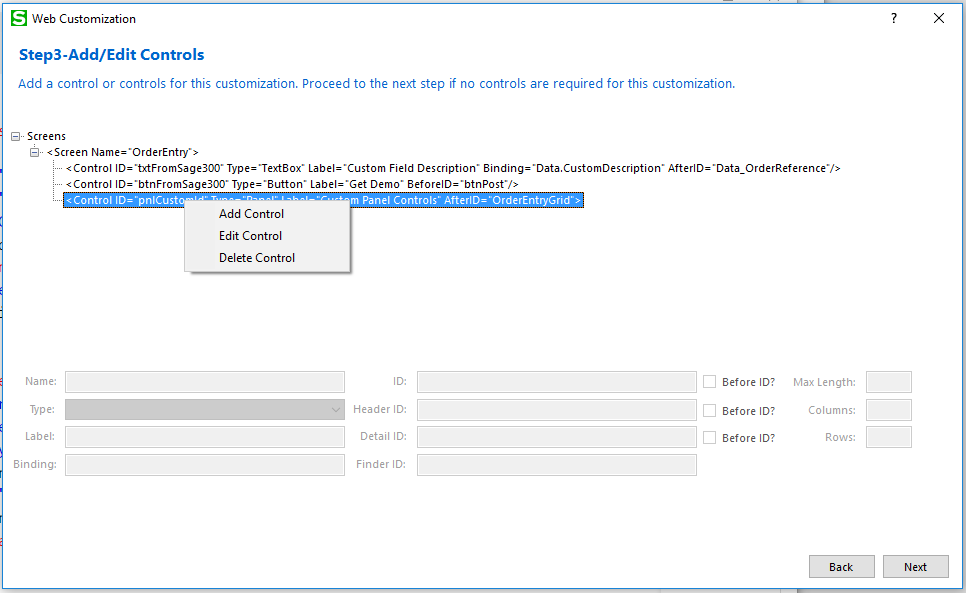
1. Right click the Screen for **OrderEntry** and select **Add Control**
2. Enter **pnlCustomId** for the Name field
3. Select **Panel** for the Type field
4. Enter **Custom Panel Controls** for the Label field
5. Enter **OrderEntryGrid** for the ID field



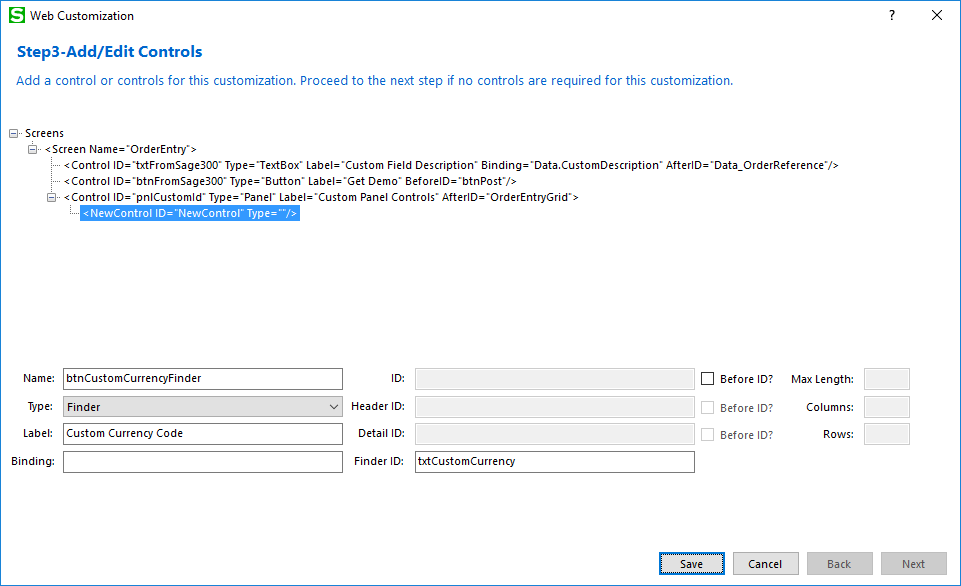
1. Click **Save** to save the control

New Control 4: Add controls to the newly added **panel.**

**Note:** Controls will be added in the order they appear on the screen and thus will not need to specify the location relevant to a Sage control as the panel control has already done this.



1. Right click the Control for **pnlCustomId** and select **Add Control**
2. Enter **btnCustomCurrencyFinder** for the Name field
3. Select **Finder** for the Type field
4. Enter **Custom Currency Code** for the Label field
5. Enter **txtCustomCurrency** for the Finder ID field



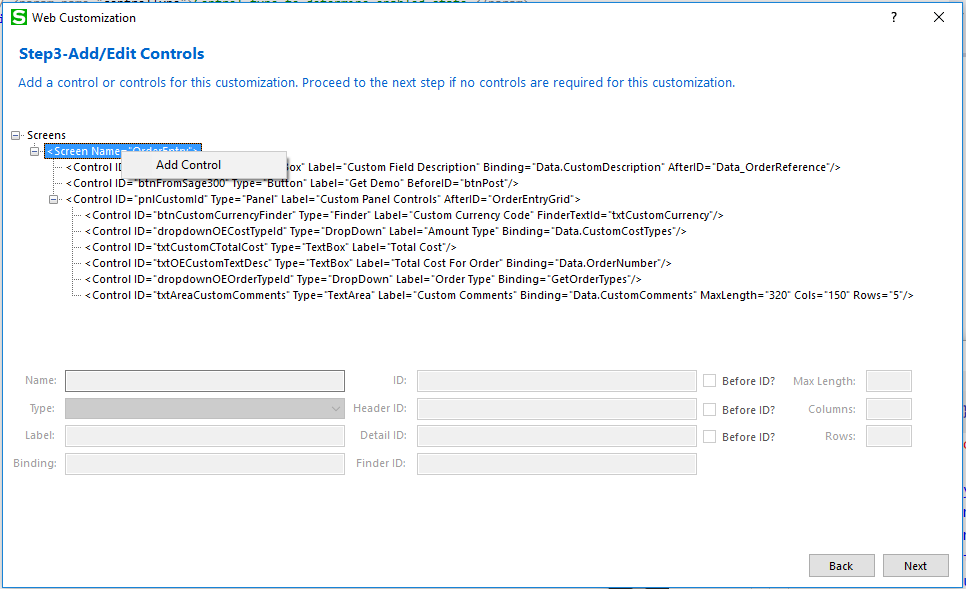
1. Click **Save** to save the control

New Controls 5-9: Add remaining controls to the newly added **panel.**

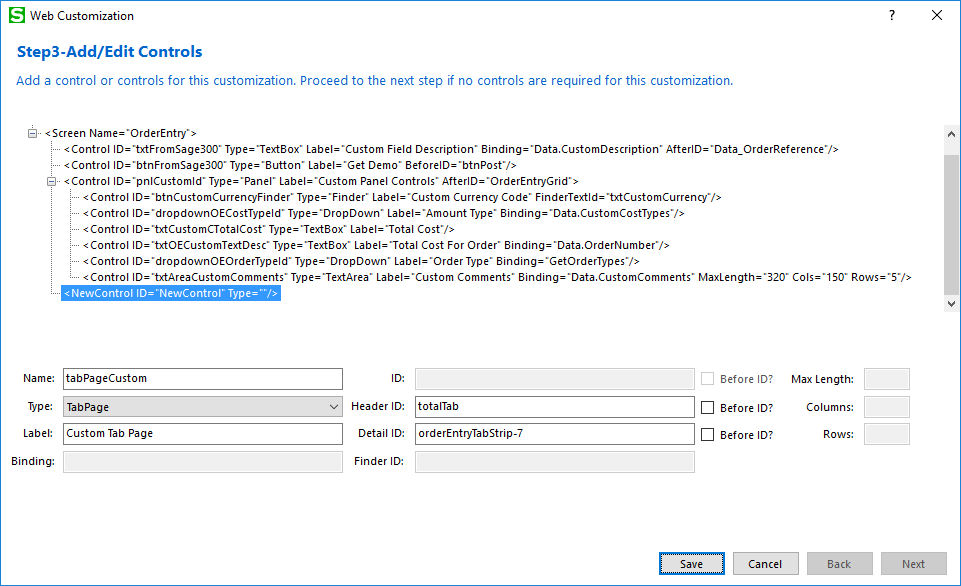
The previous step just added a control to the newly created panel. The remaining controls will be added to the panel in the same way by Right clicking the Control for **pnlCustomId** and select **Add Control.** The controls are listed here and Click **Save** for each control to be added to the panel.

* Name="dropdownOECostTypeId" Type="DropDown" Label="Amount Type" Binding ="Data.CustomCostTypes"
* Name="txtCustomCTotalCost" Type ="TextBox" Label="Total Cost"
* Name="txtOECustomTextDesc" Type ="TextBox" Label="Total Cost For Order" Binding="Data.OrderNumber"
* Name="dropdownOEOrderTypeId" Type="DropDown" Label="Order Type" Binding ="GetOrderTypes"
* Name="txtAreaCustomComments" Type="TextArea" Label="Custom Comments" MaxLength="320" Cols="150" Rows="5" Binding ="Data.CustomComments"

New Control 10: The **tab** containing several controls



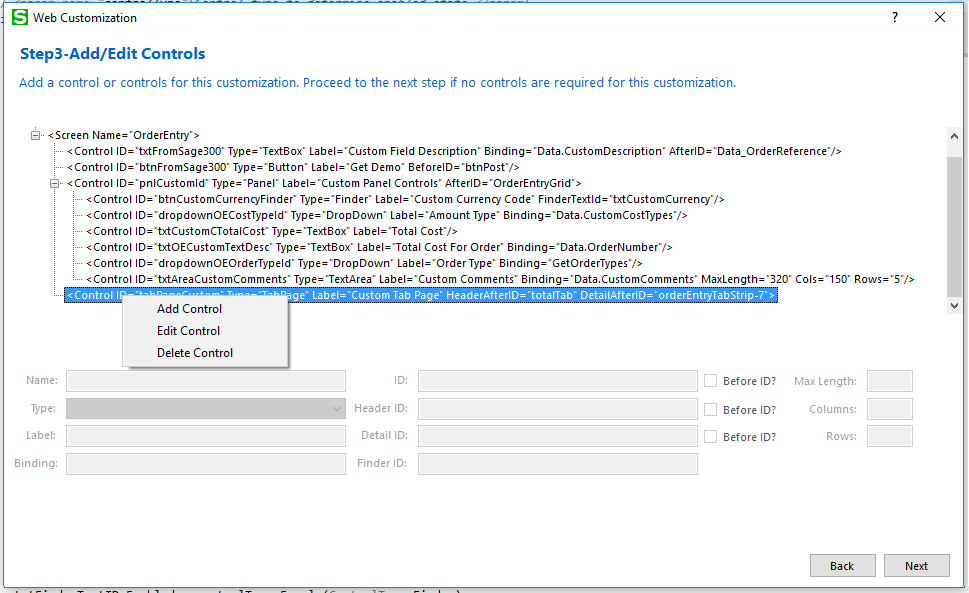
1. Right click the Screen for **OrderEntry** and select **Add Control**
2. Enter **tabPageCustom** for the Name field
3. Select **TabPage** for the Type field
4. Enter **Custom Tab Page** for the Label field
5. Enter **totalTab** for the Header ID field
6. Enter **orderEntryTabStrip-7** for the Detail Id field



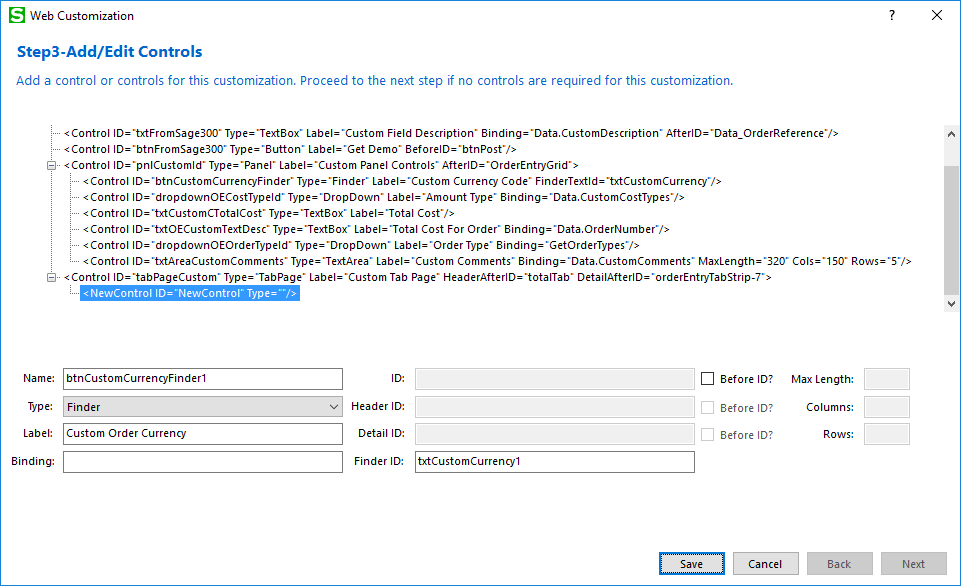
1. Click **Save** to save the control

New Control 11: Add controls to the newly added **tab page.**

**Note:** Controls will be added in the order they appear on the screen and thus will not need to specify the location relevant to a Sage control



1. Right click the Control for **tabPageCustom** and select **Add Control**
2. Enter **btnCustomCurrencyFinder1** for the Name field
3. Select **Finder** for the Type field
4. Enter **Custom Order Currency** for the Label field
5. Enter **txtCustomCurrency1** for the Finder ID field

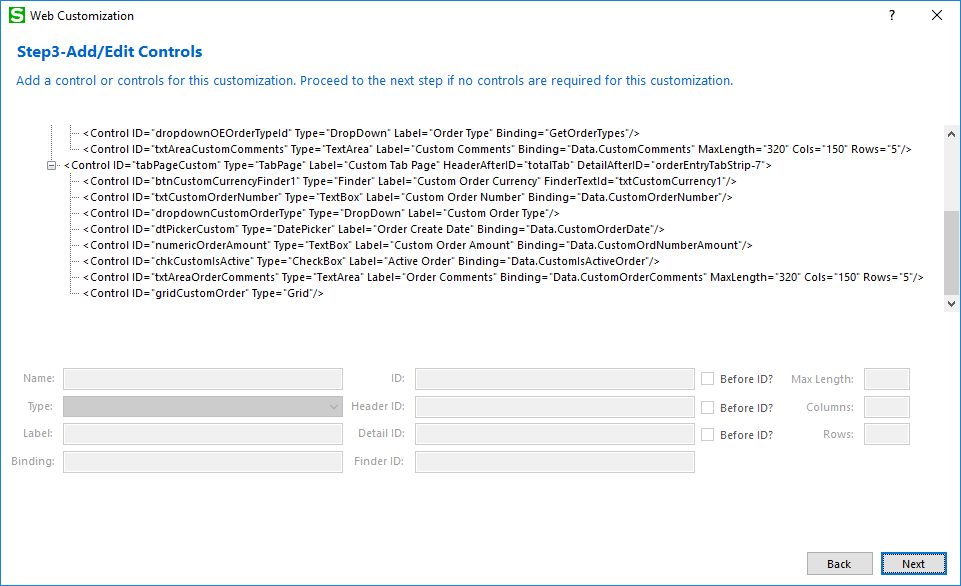


1. Click **Save** to save the control

New Controls 12-18: 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 **tabPageCustom** and select **Add Control.** The controls are listed here and Click **Save** for each control to be added to the panel.

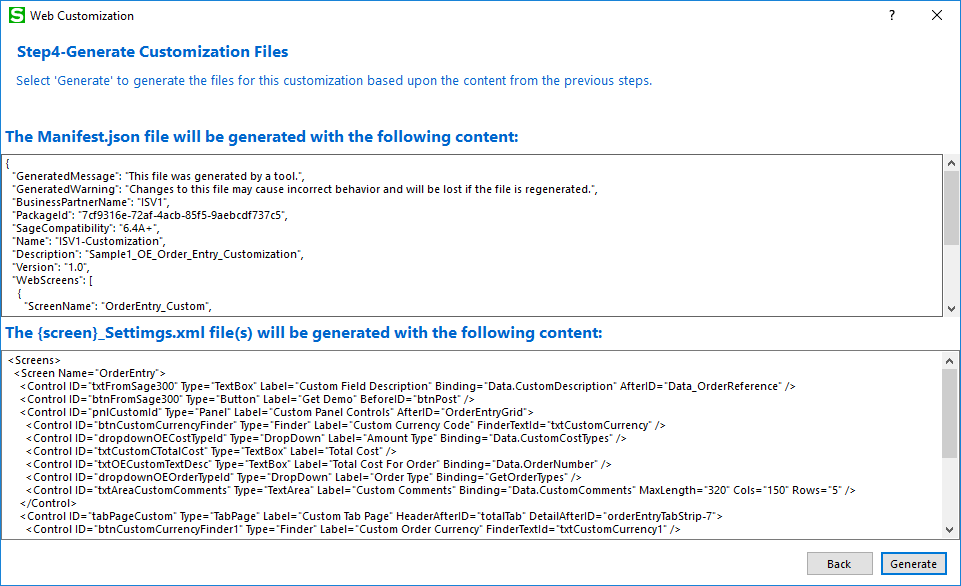
* Name="txtCustomOrderNumber" Type ="TextBox" Label="Custom Order Number" Binding="Data.CustomOrderNumber"
* Name="dropdownCustomOrderType" Type="DropDown" Label="Custom Order Type”
* Name="dtPickerCustom" Type="DatePicker" Label="Order Create Date" Binding ="Data.CustomOrderDate"
* Name="numericOrderAmount" Type ="TextBox" Label="Custom Order Amount" Binding="Data.CustomOrdNumberAmount"
* Name="chkCustomIsActive" Type="CheckBox" Label="Active Order" Binding ="Data.CustomIsActiveOrder"
* Name="txtAreaOrderComments" Type="TextArea" Label="Order Comments" MaxLength="320" Cols="150" Rows="5" Binding ="Data.CustomOrderComments"
* Name="gridCustomOrder" Type="Grid"



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

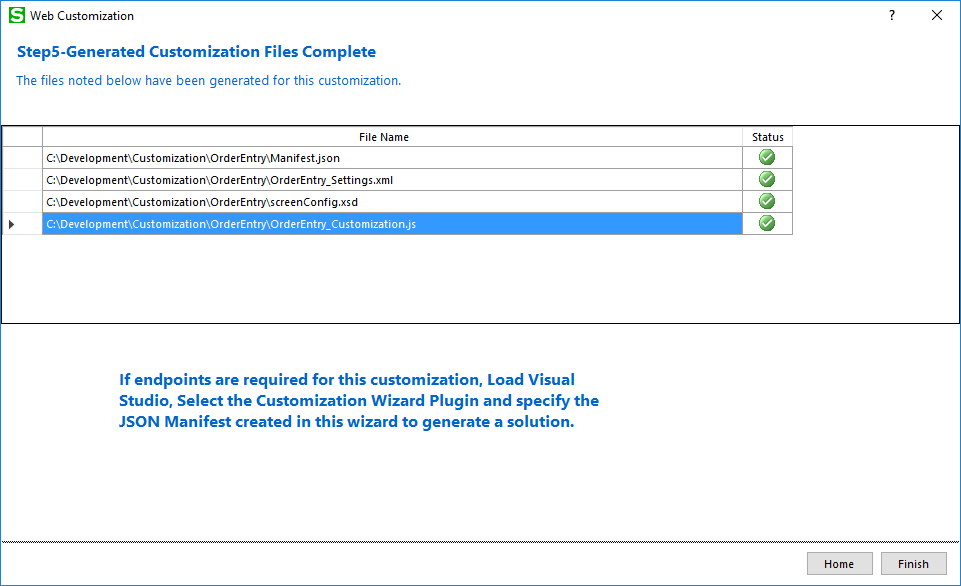
* + 1. Step 4 – Generate Customization Files

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.

Files have been generated!



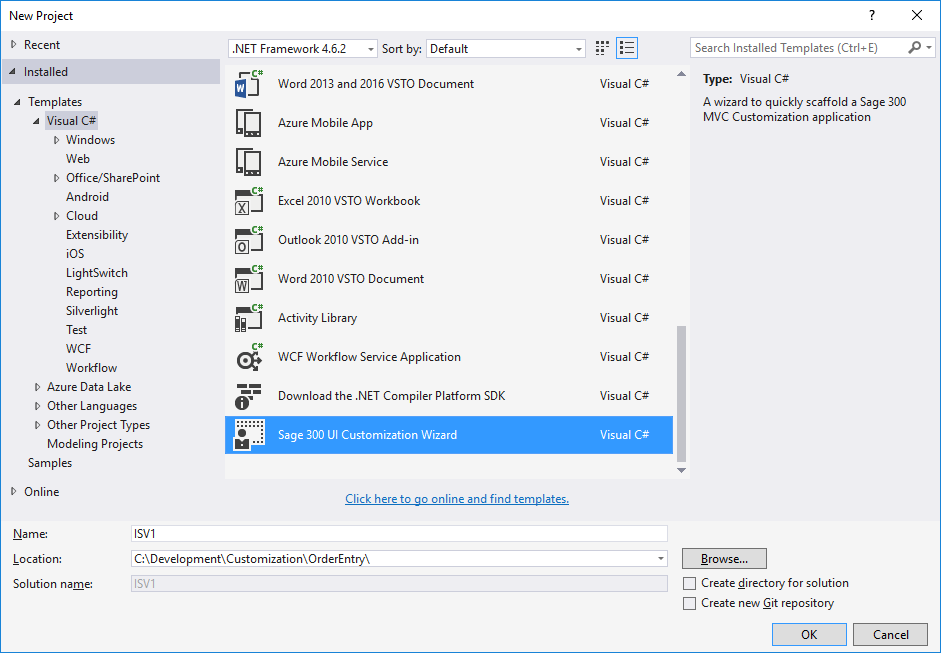
* Manifest
  + **Manifest.json**
* Control Settings file
  + **OrderEntry\_Settings.xml**
* JavaScript file
  + **OrderEntry\_Customization.js**
* XML Schema file
  + **ScreenConfig.xsd**

Since the customization requires an endpoint, we will proceed to the Customization Plug-in in Visual Studio.

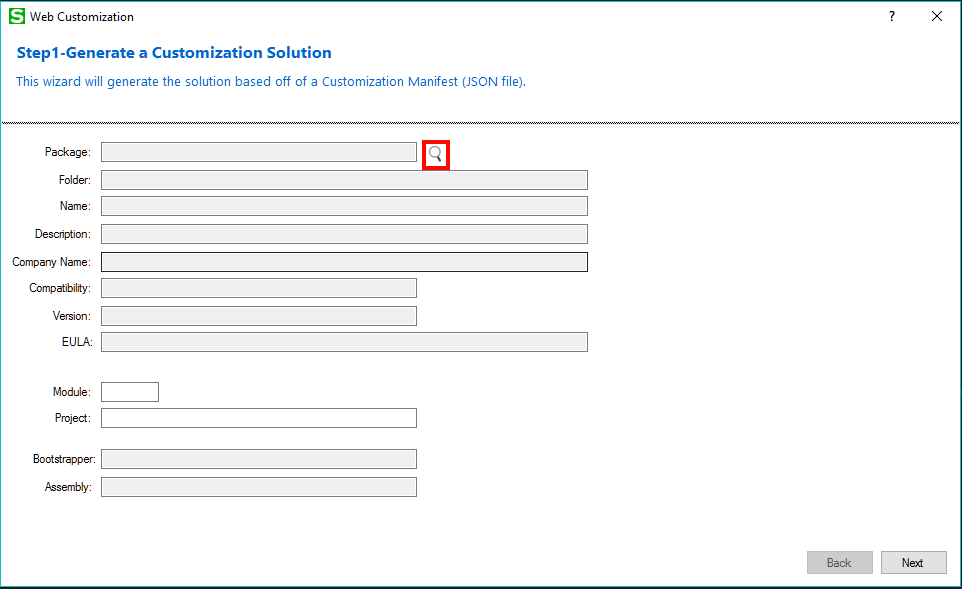
* 1. Plug-in Customization Wizard

Create the solution for the customization.

* + 1. Step 1 – Create a New Project



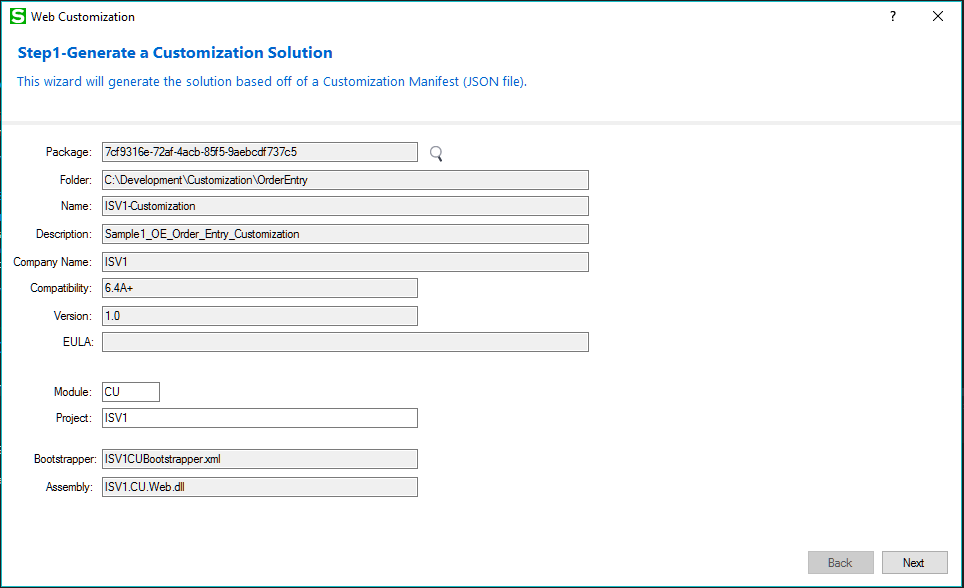
1. Open Visual Studio to create a new project,
2. In the new project dialog box, select Sage 300 UI Customization Wizard.
3. Enter **ISV1** for the name and select the location where the manifest (customization) was created
4. Click OK.
   * 1. Step 2 – Generate a Customization Solution



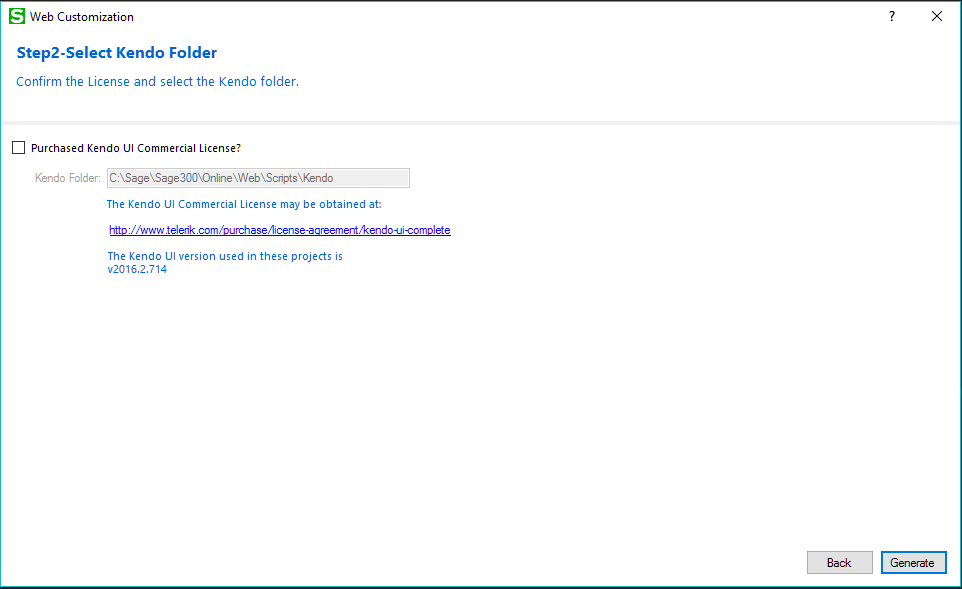
1. Click on the magnifying glass next to Package to select the manifest JSON file created for this customization.
2. The rest of the fields will populate according to the contents of the file:

**Note**: Modify the Project field to be ISV1 as the sample for Order Entry is based on a non-standard name.

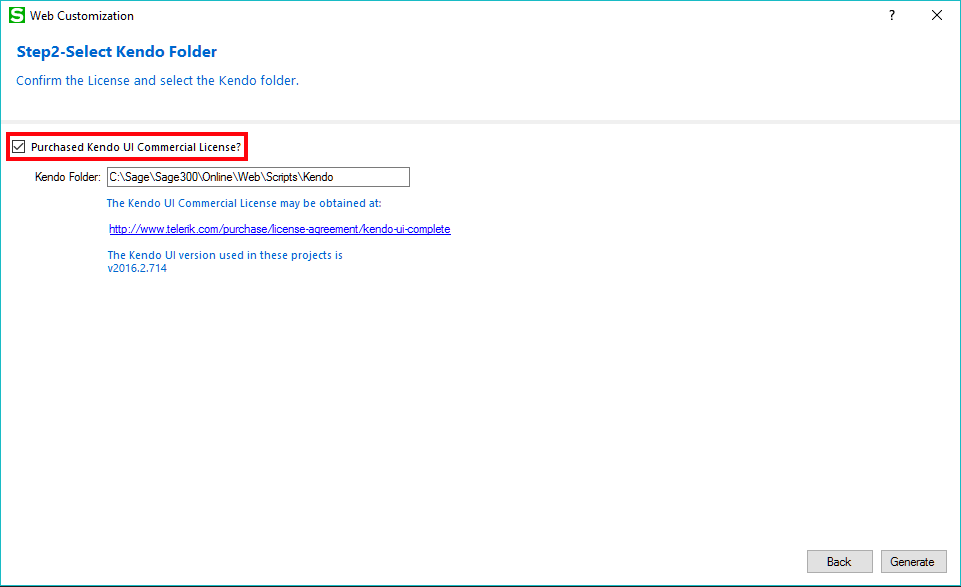
**Note**: The Assembly generated will have the module name included as this is now the standard for web projects whereas the customization sample is based upon a previous naming scheme.



1. Click **Next** to go to the next screen.



1. It is expected that our partners have purchased their own license to use the Kendo UI components. Click on the checkbox to confirm that you have a license to use the Kendo UI components.



1. Press **Generate** to create the solution.
2. Select Nuget Package Manager and restore Nuget Packages
3. Compile the project and ensure it does not have any compilation errors.
4. 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 **OrderEntry\_Customization.js** in the editor of your choice to perform the modifications in the following sections.

* 1. Kendo Initialization

Some Kendo UI controls require initialization in JavaScript.

* + 1. initDropDownList Function

Add the following code snippet in the initDropDownList function:

|  |
| --- |
| // Amount type dropdown list  $("#dropdownOECostTypeId").kendoDropDownList({  });  // Order type dropdown list  $("#dropdownOEOrderTypeId").kendoDropDownList({  });  // Custom drop down list in custom tab page  $("#dropdownCustomOrderType").kendoDropDownList({  }); |

* + 1. initDatePicker Function

Add the following code snippet in the initDatePicker function:

|  |
| --- |
| $("#dtPickerCustom").kendoDatePicker({  }); |

* 1. Initialization

Some controls require initialization.

For example, a Panelcontrol should be expanded, a Tab Page should be disabled when the screen first opens, etc.

* + 1. Init Function

Add the following snippet to the init function for a Panel control to be collapsible:

|  |
| --- |
| sg.utls.collapsibleScreen.setup("expandedEntry", "simpleEntry", ["pnlCustomIdHeader"], []);  $("#pnlCustomIdHeader").trigger("click"); |

* + 1. initOtherControls Function

Add the following snippet to the initOtherControls function to disable the Tab Page control:

|  |
| --- |
| var tabStrip = $("#orderEntryTabStrip").data("kendoTabStrip");  tabStrip.enable(tabStrip.tabGroup.children().eq(7), false); |

* + 1. initFinders Function

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

|  |
| --- |
| sg.finderHelper.setFinder("btnCustomCurrencyFinder",  sg.finder.TaxCurrencyFinder, ISV1OrderEntryCustomizationUICallback.currencyCode, null,  "Custom Currency Finder", sg.finderHelper.createDefaultFunction("txtCustomCurrency",  "CurrencyCodeId", sg.finderOperator.StartsWith), null, true);  sg.finderHelper.setFinder("btnCustomCurrencyFinder1", sg.finder.TaxCurrencyFinder,  ISV1OrderEntryCustomizationUICallback.currencyCode1, null, "Custom Currency Finder",  sg.finderHelper.createDefaultFunction("txtCustomCurrency1", "CurrencyCodeId",  sg.finderOperator.StartsWith), null, true); |

* + 1. initNumericTextBox Function

Add the following snippet to the initNumericTextBox control for numeric numbers:

|  |
| --- |
| $("#numericOrderAmount").kendoNumericTextBox({  spinners : false,  decimals: 3  }); |

* + 1. Kendo UI Grid, Dropdown and other controls

Refer to the sample code for details.

* + 1. initDropdownList Function

Update initDropDownList Amount type dropdown list to add binding data source and change event handler:

|  |
| --- |
| $("#dropdownOECostTypeId").kendoDropDownList({  dataSource: {  data: ["Extended Price", "Discounted Extended Amount", "Extended Order Cost"]  },  change: function (e) {  var idx = this.selectedIndex;  $("#txtCustomCTotalCost").val(ISV1OrderEntryCustomizationUI.total[idx]);  }  }); |

* 1. Intercept existing events and add custom logic

There are two ways to intercept an existing event.

1. Get the existing event and handler, unbind the existing handler, and bind it to the custom handler

As an example, to intercept the existing Prepayment button’s click event, show a confirmation message, and then cancel the prepayment click action, add the following code in the initIntercept function:

|  |
| --- |
| // Prepayment button event intercept  if ($('#btnPrepayment')[0] == undefined) return;  var prepaymentHandler = null;  var prepaymentHandlers = $('#btnPrepayment').data('events').click;  if (prepaymentHandlers && prepaymentHandlers.length > 0) {  prepaymentHandler = prepaymentHandlers[0].handler;  }  // unbind the original  $('#btnPrepayment').unbind('click');  // bind the modified one  $('#btnPrepayment').click(function () {  sg.utls.showKendoConfirmationDialog(  function () {  var data = orderEntryUI.orderEntryModel.Data;  var amountDue = data.AmountDueLessCurrPrepayment();  data.AmountDueLessCurrPrepayment(amountDue + 1000);  prepaymentHandler();  },  function () {  return false;  },  "This will intercept the Sage 300c Order Entry show prepayment action and add extra cost to the order prepayment amount due. Are you sure you want to do that?");  }); |

1. Intercept the screen’s Ajax call and add customization logic before or after the Ajax call

To intercept an existing Ajax call, you will need to get part of the Ajax call URL using a web tool such as Fiddler. For the Order Entry GetDetails action, part of the URL is /OE/OrderEntry/GetDetails, so in the JavaScript code, use this part of the URL to specify which Ajax call to intercept.

As an example, we will add code to calculate some order detail field amounts and will show them in the added Total Cost textbox.

To define the variables, refer to the sample code.

* + 1. ajaxSuccess Function

Add the following code to the $(document).ajaxSuccess(function (event, xhr, settings) function:

|  |
| --- |
| if (settings.url.indexOf("/OE/OrderEntry/GetDetails") > -1) {  var data = jQuery.parseJSON(settings.data);  ISV1OrderEntryCustomizationUI.orderNumber = data.model.OrderNumber;  if (data.model.OrderNumber && data.model.OrderNumber !== "\*\*\* NEW \*\*\*") {  $("#btnFromCustom").trigger('click');  var grid = $("#OrderDetailGrid").data("kendoGrid");  var gridData = grid.dataSource.view();  for (var i = 0; i < gridData.length; i++) {  ISV1OrderEntryCustomizationUI.total[0] += gridData[i].ExtendedPrice;  ISV1OrderEntryCustomizationUI.total[1] += gridData[i].DiscountedExtendedAmount;  ISV1OrderEntryCustomizationUI.total[2] += gridData[i].ExtendedOrderCost;  }  $("#txtCustomCTotalCost").val(ISV1OrderEntryCustomizationUI.total[0]);  var comment = "Total Cost for " + data.model.OrderNumber + " is " + ISV1OrderEntryCustomizationUI.total[0];  $("#txtAreaCustomComments").val(comment);  $("#txtCustomCurrency").val("CAD");  //Enable custom tab page  var tabStrip = $("#orderEntryTabStrip").kendoTabStrip().data("kendoTabStrip");  tabStrip.enable(tabStrip.tabGroup.children().eq(7), true);  }  } |

* + 1. ajaxComplete Function

When the GetDetails routine is successful, add the following code to send the Ajax call to load custom details information from the custom controller:

|  |
| --- |
| $(document).ajaxComplete(function (event, xhr, settings) {  if (settings.url.indexOf("/OE/OrderEntry/GetDetails") > -1) {  var data = jQuery.parseJSON(settings.data);  if (data.model.OrderNumber && data.model.OrderNumber !== "\*\*\* NEW \*\*\*") {  var url = sg.utls.url.buildUrl("CU", "ISV1Customization", "GetOrderDetails");  $.ajax({  type: 'get',  dataType: 'json',  cache: false,  //url: url,  //Use local web server url, when deploy to Sage 300c, use above url  url: 'http://localhost/ISV1.web/OnPremise/CU/ISV1Customization/GetOrderDetails',  data: { id: data.model.OrderNumber },  success: function (data) {  ISV1OrderEntryCustomizationUICallback.loadCustomOrder(data);  },  error: function () {  }  });  }  }  }); |

The Ajax call URL should be the web server URL. When deployed to the web application, it should use the sg.utls.url.buildUrl function to build the Ajax call URL, such as sg.utls.url.buildUrl("CU", "ISV1Customization", "GetOrderDetails"). The parameters are the area (module), controller name, and action method. For details, refer to the sample code.

* 1. Add event handler and call back functions

Some functions or methods require a call back function.

For example, the finder control setup method and Ajax calls method. In the Order Entry customization, an Ajax call will be made to the Common Services Tax Authorities controller to get a currency description.

* + 1. initButtons Function

Add the following code to the initButtons function:

|  |
| --- |
| $("#btnFromSage300").bind('click', function () {  var url = sg.utls.url.buildUrl("CS", "TaxAuthority", "GetCurrencyDescription");  sg.utls.ajaxPost(url, { currencyCode: "CAD" }, ISV1OrderEntryCustomizationUICallback.getCustomInfo);  }); |

* + 1. getCustomInfo Function

Add the following call back function getCustomInfo in ISV1OrderEntryCustomizationUICallback:

|  |
| --- |
| getCustomInfo: function (result) {  result = result.CurDescription + "-This info is get from sage300c controller";  $('#txtFromSage300').val(result);  //sg.utls.showKendoConfirmationDialog(function () { }, null, "Custom Text box info is get from sage 300 server ajax call");  sg.controls.Focus($("#txtCustomTextDesc"));  }, |

* + 1. Confirmation Dialog

When the btnFromSage300 is clicked, a confirmation message box will be displayed and the txtFromSage300 textbox will be populated.

* + 1. loadCustomerOrder Function

Add the call back function loadCustomerOrder in ISV1OrderEntryCustomizationUICallback:

|  |
| --- |
| // Load custom order details and apply knock out bindings  loadCustomOrder: function (data) {  data.Data.CustomOrderDate = new Date(parseInt(data.Data.CustomOrderDate.substr(6))).toLocaleDateString();  var customViewModel = ko.mapping.fromJS(data);  // apply knock out binding to custom page content  ko.applyBindings(customViewModel, document.getElementById("orderEntryTabStrip-8"));  // apply to dropdown list using kendo data source  $("#dropdownCustomOrderType").kendoDropDownList({  dataSource: data.Data.CustomOrderType,  change: function (e) {  }  });  $("#numericOrderAmount").data("kendoNumericTextBox").value(data.Data.CustomOrdNumberAmount);  $("#txtCustomCurrency1").val(data.Data.CustomOrderCurrency);  } |

When the order details are loaded, the custom order details fields will be populated with custom order details.

* + 1. Other Functions

To add other call back functions to ISV1OrderEntryCustomizationUICallback, refer to the sample code.

1. Add custom model, view model, and controller

You can add a custom model, view model, and controller under the Areas\CU\Models and Areas\CU\Controllers folders within the solution generated by the plug-in customization wizard.

The custom view model fields can be bound to custom added controls. See sample order entry custom tab page field bindings.

* 1. Add event handler and call back functions

Add the custom view model class (as the OrderViewModel.cs file) to the Models folder. The file content is as follows:

|  |
| --- |
| public class OrderViewModel  {  public List<string> CustomOrderType  {  get  {  return new List<string> {"Order Type 1", "Order Type 2", "Order Type 3", "Order Type 4"};  }  }  public string CustomOrderNumber { get; set; }  public string CustomOrderComments { get; set; }  public DateTime CustomOrderDate { get; set; }  public double CustomOrdNumberAmount { get; set; }  public bool CustomIsActiveOrder { get; set; }  public string CustomOrderCurrency { get; set; }  } |

* 1. Add custom controller action methods

Add the following action methods in the ISV1CustomizationController class:

|  |
| --- |
| [HttpGet]  public JsonResult GetOrderDetails(string id)  {  // Call Sage 300c controller/service/repository or custom service to get data  var data = new OrderViewModel();  data.CustomOrderNumber = "CustomOrder\_00001";  data.CustomOrderComments = "This is custom order comments.";  data.CustomOrderDate = new DateTime(2016, 10, 28);  data.CustomIsActiveOrder = true;  data.CustomOrdNumberAmount = 20100.82;  data.CustomOrderCurrency = "CAD";    JsonResult jsonResult = new JsonResult {JsonRequestBehavior = JsonRequestBehavior.AllowGet};  jsonResult.Data = Json(data);  return jsonResult;  } |

When built in release mode, it will deploy the ISV1CUBootstrapper.xml and ISV1.CU.web.dll to the Sage 300c web application root and bin folder respectively. The Sage 300 web application will load this assembly and it will be used by the application.

1. Packaging the Customization

When the customization is complete (coded, tested, etc.), the developer should package the customization files to a customization package (zip file). This customization package will be consumed in the Customization Admin screen. This screen will automatically manage the import of the customization files, assignment to the applicable companies, etc.

A customization package can include a customization for a single screen or a customization for multiple screens.

Note: It is recommended that you create the package for a single screen in order to simplify the administration and management of the customization.

The customization package must include for this customization:

* The **Manifest.json** file
* The **OrderEntry\_Settings.xml** file
* The **screenConfig.xsd** file
* The **OrderEntry\_Customization.js** file
* The **ISV1CUBootstrapper.xml** file
* The **ISV1.CU.Web.dll** file
  1. Manifest.json Contents

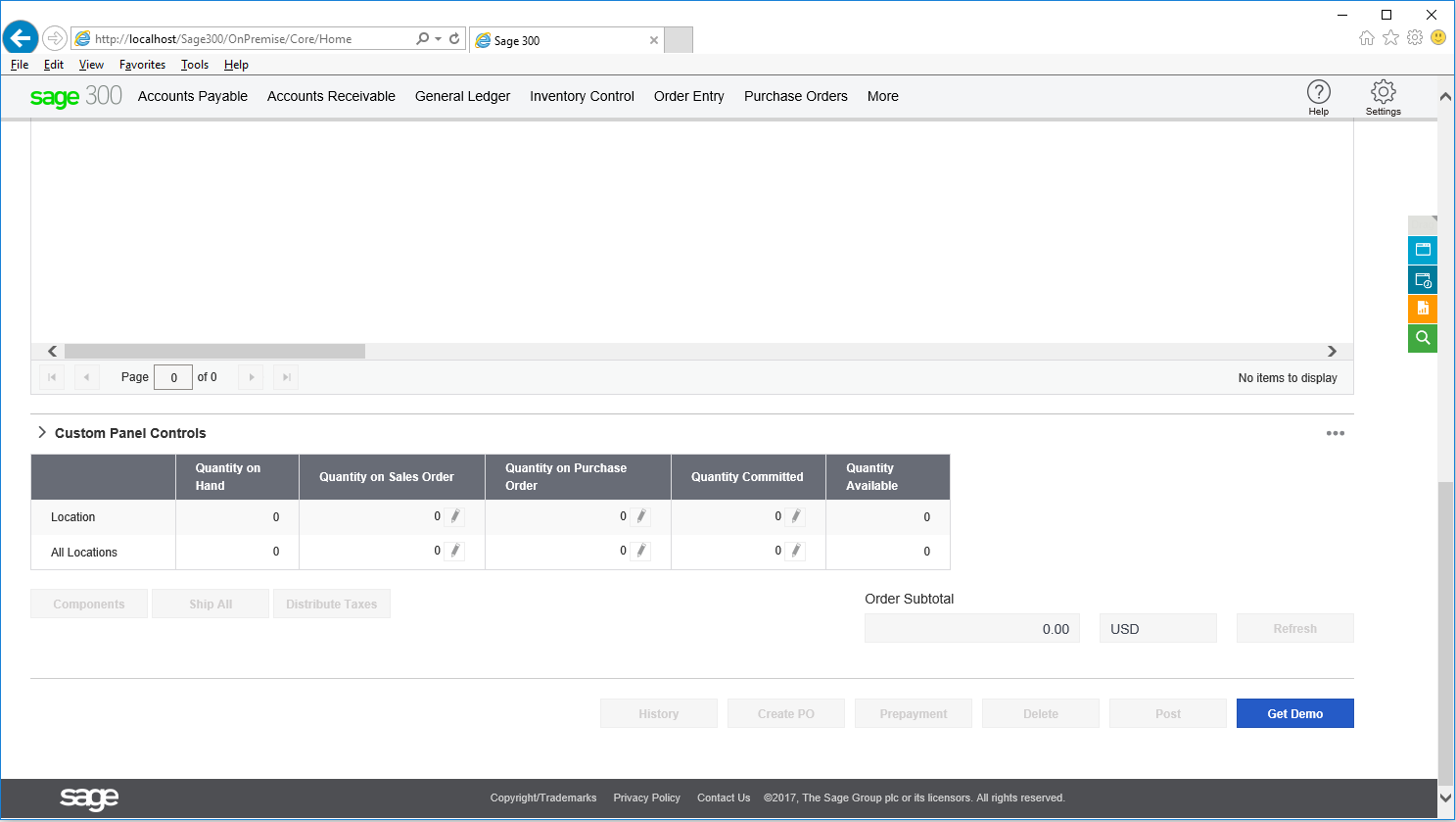
|  |
| --- |
| {  "GeneratedMessage": "This file was generated by a tool.",  "GeneratedWarning": "Changes to this file may cause incorrect behavior and will be lost if the file is regenerated.",  "BusinessPartnerName": "ISV1",  "PackageId": "7cf9316e-72af-4acb-85f5-9aebcdf737c5",  "SageCompatibility": "6.4A+",  "Name": "ISV1-Customization",  "Description": "Sample1\_OE\_Order\_Entry\_Customization",  "Bootstrapper": "ISV1CUBootstrapper.xml",  "Assembly": "ISV1.CU.Web.dll",  "Version": "1.0",  "WebScreens": [  {  "ScreenName": "OrderEntry\_Custom",  "ScreenDescription": "OrderEntry\_Custom Description",  "Category": "Transactions",  "TargetScreen": "OrderEntry",  "ControlsConfiguration": "OrderEntry\_Settings.xml",  "ControlsBehavior": "OrderEntry\_Customization.js",  "Module": "OE"  }  ]  } |

* 1. OrderEntry\_Settings.xml Contents

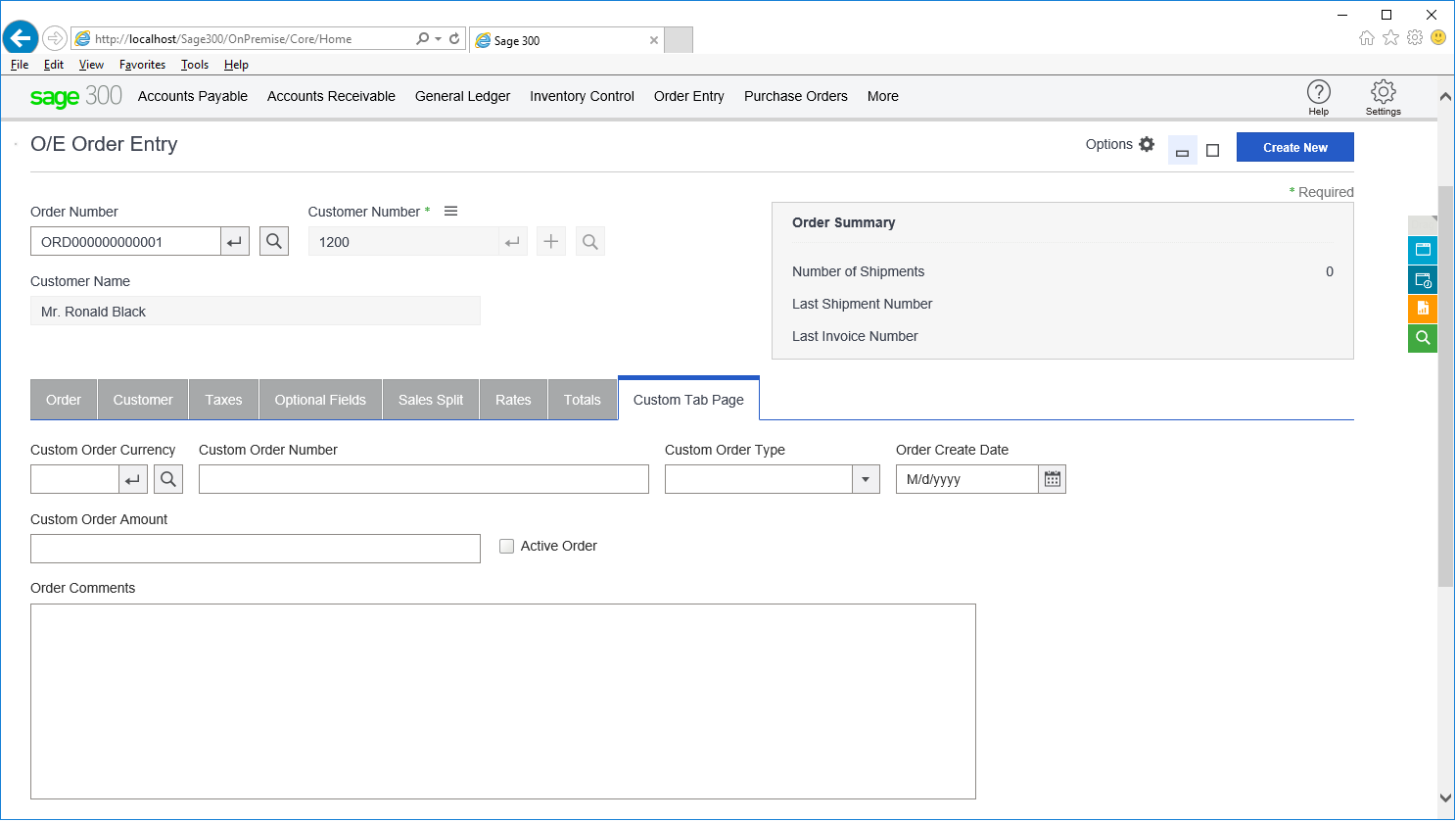
|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <!-- This file was generated by a tool. -->  <!-- Changes to this file may cause incorrect behavior and will be lost if the file is regenerated. -->  <Screen Name="OrderEntry" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="screenConfig.xsd">  <Control ID="txtFromSage300" Type="TextBox" Label="Custom Field Description" Binding="Data.CustomDescription" AfterID="Data\_OrderReference" />  <Control ID="btnFromSage300" Type="Button" Label="Get Demo" BeforeID="btnPost" />  <Control ID="pnlCustomId" Type="Panel" Label="Custom Panel Controls" AfterID="OrderEntryGrid">  <Control ID="btnCustomCurrencyFinder" Type="Finder" Label="Custom Currency Code" FinderTextId="txtCustomCurrency" />  <Control ID="dropdownOECostTypeId" Type="DropDown" Label="Amount Type" Binding="Data.CustomCostTypes" />  <Control ID="txtCustomCTotalCost" Type="TextBox" Label="Total Cost" />  <Control ID="txtOECustomTextDesc" Type="TextBox" Label="Total Cost For Order" Binding="Data.OrderNumber" />  <Control ID="dropdownOEOrderTypeId" Type="DropDown" Label="Order Type" Binding="GetOrderTypes" />  <Control ID="txtAreaCustomComments" Type="TextArea" Label="Custom Comments" Binding="Data.CustomComments" MaxLength="320" Cols="150" Rows="5" />  </Control>  <Control ID="tabPageCustom" Type="TabPage" Label="Custom Tab Page" HeaderAfterID="totalTab" DetailAfterID="orderEntryTabStrip-7">  <Control ID="btnCustomCurrencyFinder1" Type="Finder" Label="Custom Order Currency" FinderTextId="txtCustomCurrency1" />  <Control ID="txtCustomOrderNumber" Type="TextBox" Label="Custom Order Number" Binding="Data.CustomOrderNumber" />  <Control ID="dropdownCustomOrderType" Type="DropDown" Label="Custom Order Type" />  <Control ID="dtPickerCustom" Type="DatePicker" Label="Order Create Date" Binding="Data.CustomOrderDate" />  <Control ID="numericOrderAmount" Type="TextBox" Label="Custom Order Amount" Binding="Data.CustomOrdNumberAmount" />  <Control ID="chkCustomIsActive" Type="CheckBox" Label="Active Order" Binding="Data.CustomIsActiveOrder" />  <Control ID="txtAreaOrderComments" Type="TextArea" Label="Order Comments" Binding="Data.CustomOrderComments" MaxLength="320" Cols="150" Rows="5" />  <Control ID="gridCustomOrder" Type="Grid" />  </Control>  </Screen> |

1. Customized Screen
   1. Customized Order Entry screen

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



*Customization for Panel Controls*



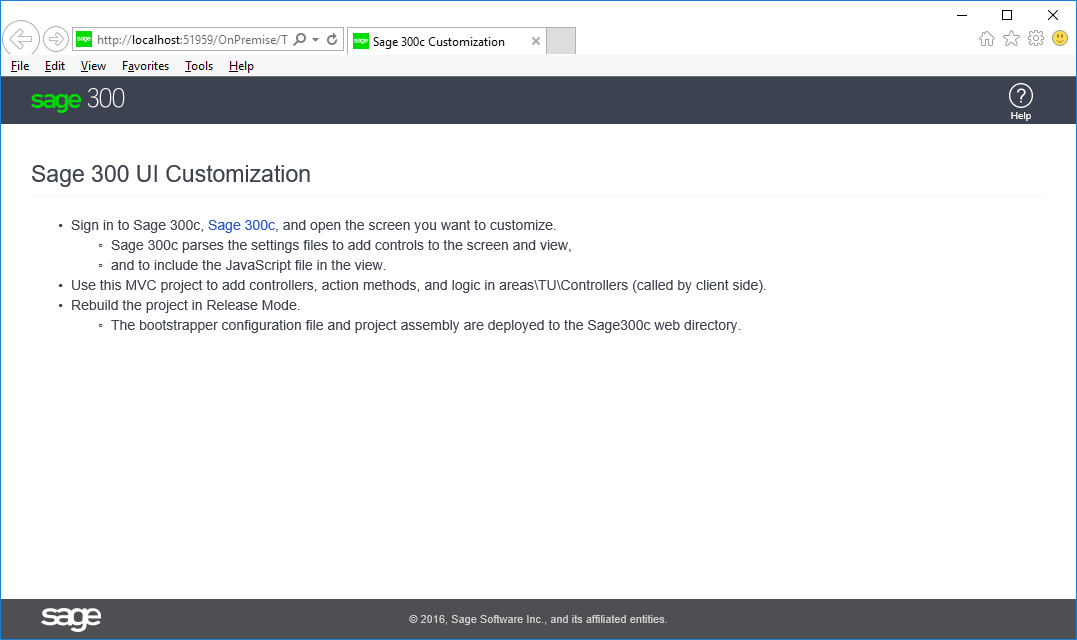
*Customization for Tab Page*

1. Debugging

Local debugging can be accomplished by copying the XML Settings file and JavaScript file to the shared data folder’s Customization folder of the local installation *{Sage300 installed shared data folder}\Customization folder*

To start debugging:

1. Open the above generated screen settings file OrderEntry\_Settings.xml and JavaScript file OrderEntry\_Customization.js in the code editor of VS 2013 / VS2015.
2. In the JS file, set a breakpoint at the initial entry point ISV1OrderEntryCustomizationUI.init()
3. Run the project with Internet Explorer.
4. When the default page appears, click the [Sage 300c application](http://localhost/Sage300) link to switch to the local Sage 300c web application.



1. Sign in to the Sage 300c web application and navigate to the O/E Order Entry screen. As this screen rendering html document includes the OrderEntry\_Customization.js, when the O/E Order Entry document screen is ready, it will hit the breakpoint