Sage 300 Web Screens SDK

Web Screen Customization

July 2017

The MIT License (MIT)

Copyright © 2017 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](#_Toc487213254)

[2. New to 2018 – split into two wizards 5](#_Toc487213255)

[3. Development environment prerequisites 6](#_Toc487213256)

[3.1 Install Sage 300 6](#_Toc487213257)

[3.2 Install Visual Studio Web Customization Package (optional) 6](#_Toc487213258)

[4. Getting started 7](#_Toc487213259)

[4.1 Generate files for a customization package 7](#_Toc487213260)

[4.2 Create the customization project (optional) 10](#_Toc487213261)

[4.3 Open the settings and JavaScript files and start to debug 13](#_Toc487213262)

[5. Development - add controls to the screen 14](#_Toc487213263)

[5.1 XML settings 15](#_Toc487213264)

[5.2 Initialize controls 17](#_Toc487213265)

[5.3 Customized Order Entry screen 18](#_Toc487213266)

[6. Development - add JavaScript 20](#_Toc487213267)

[6.1 Initialize controls 20](#_Toc487213268)

[6.2 Intercept existing events and add custom logic 21](#_Toc487213269)

[6.3 Add event handler and call back functions 24](#_Toc487213270)

[7. Development - add custom model, view model, and controller 26](#_Toc487213271)

[7.1 Add event handler and call back functions 26](#_Toc487213272)

[7.2 Add custom controller action methods 26](#_Toc487213273)

[8. Packaging the customization 28](#_Toc487213274)

1. Overview

This document is intended to serve as a guide for illustrating how to perform a Sage 300 web screen customization (customization).

The first, and most important steps in the customization process are to become familiar with the web screen to be customized.

Customization steps:

1. XML file for adding control(s)
2. JavaScript file for implementing the business logic and/or control behavior
3. Optional MVC Controller/Model/Service for extending functionality

This guide performs a mock customization of the Order Entry screen:

* Adding controls
* Binding new controls to existing fields and newly added fields
* Intercepting events of existing controls and newly added controls

1. New to 2018 – split into two wizards

Standalone customization wizard:

Visual Studio plugin customization wizard:

1. Development environment prerequisites

Before starting the customization, Sage 300 must be installed on the developer machine as well as either the Sage 300 UI Customization Wizard.

Familiarity with the web screen to be customized is a requirement. Additionally, knowledge and expertise in the following technologies are required:

* Microsoft ASP.NET MVC 5
* Microsoft C#
* JavaScript
* JQuery and JQuery Ajax (<https://jquery.com/>, <http://api.jquery.com/jquery.ajax/>)
* Kendo UI (<http://www.telerik.com/kendo-ui>)
* Knockout JavaScript Library (<http://knockoutjs.com/documentation/introduction.html>)
  1. Install Sage 300

Install Sage 300 on the developer machine. After installation, change the web application’s web.config file’s debug mode:

1. In {*Sage 300 installation folder*}\Online\Web, open the web.config file.
2. Locate the <System.web> entry and the <compilation debug="**false**" targetFramework="4.x.x" /> tag, where 4.x.x is version number, e.g. 4.5.1
3. Change the debug attribute to <compilation debug="**true**" targetFramework="4.x.x" />
   1. Install Visual Studio Web Customization Package (optional)

The Sage 300 UI Customization Wizard (Sage300CustomProject.vsix) will install this plugin to Visual Studio and is compatible with Visual Studio Versions 2013 and 2015.

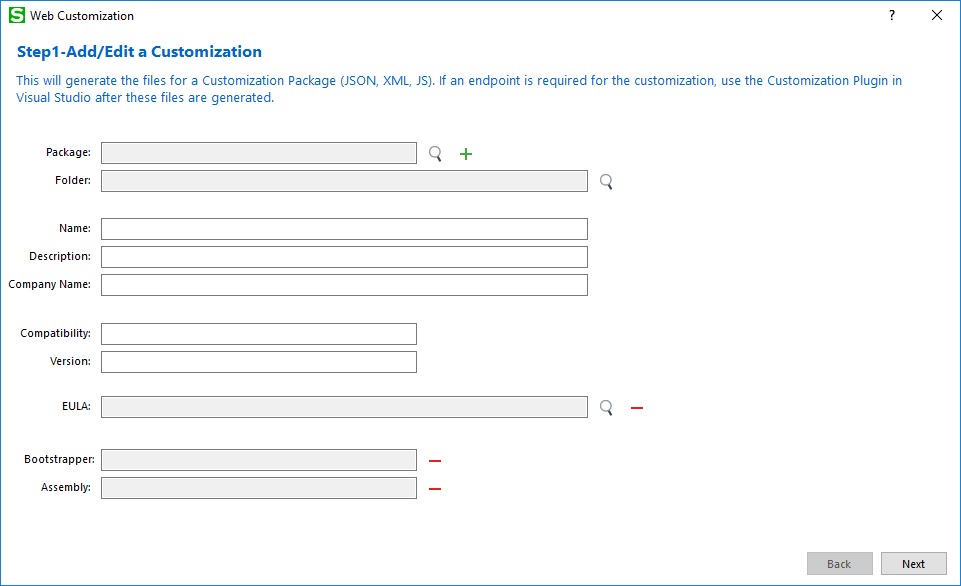
Selecting this file will invoke a dialog, prompting to be installed. Select **OK** to install the plugin.

Note: If the plugin is already installed, it must be uninstalled first and this is accomplished in Visual Studio by selecting Tools/Extensions and Updates, locating the plugin and selecting the Uninstall option.

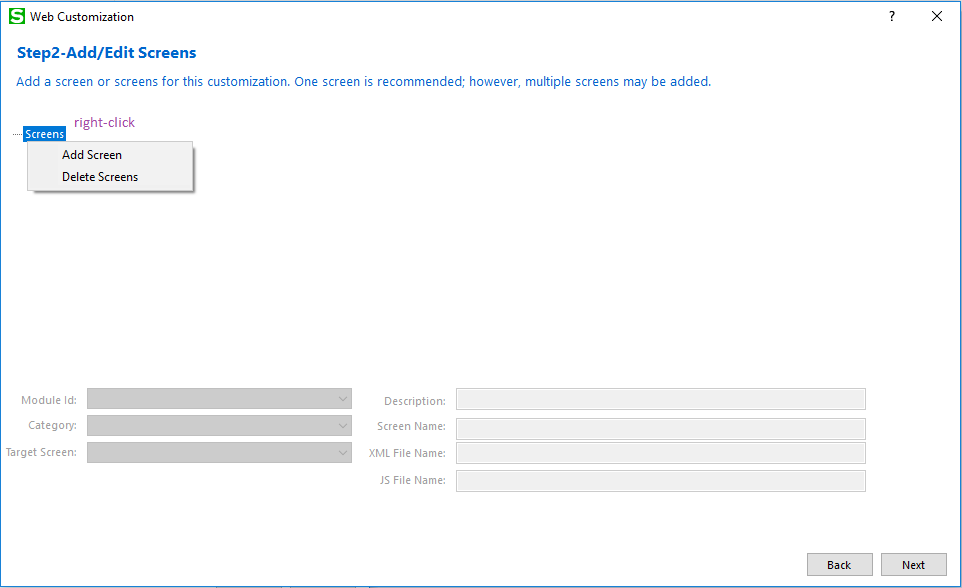
1. Getting started
   1. Generate files for a customization package

To create the JSON, XML and JS files: (refer to *Sample1 - OE Order Entry Customization*)

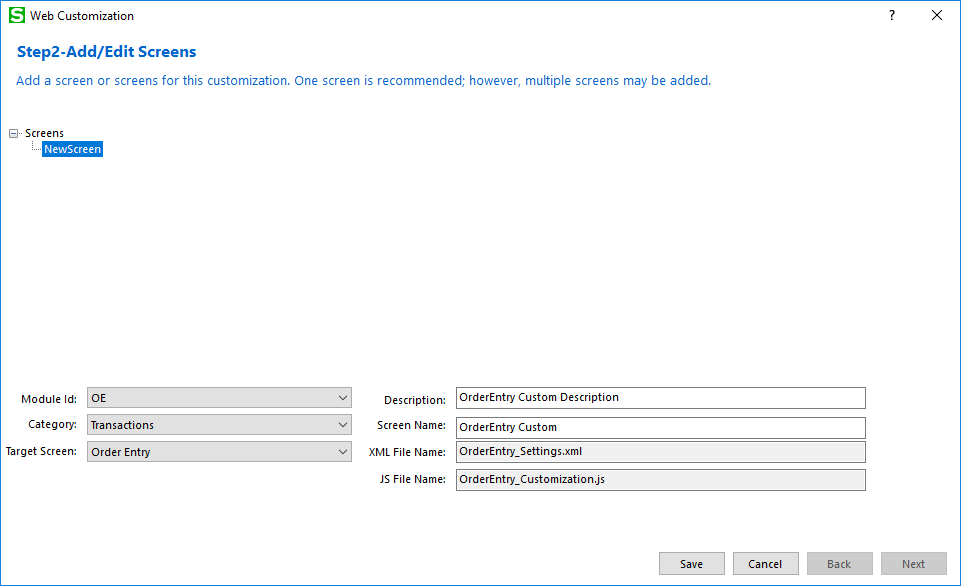
1. Open the standalone customization wizard:



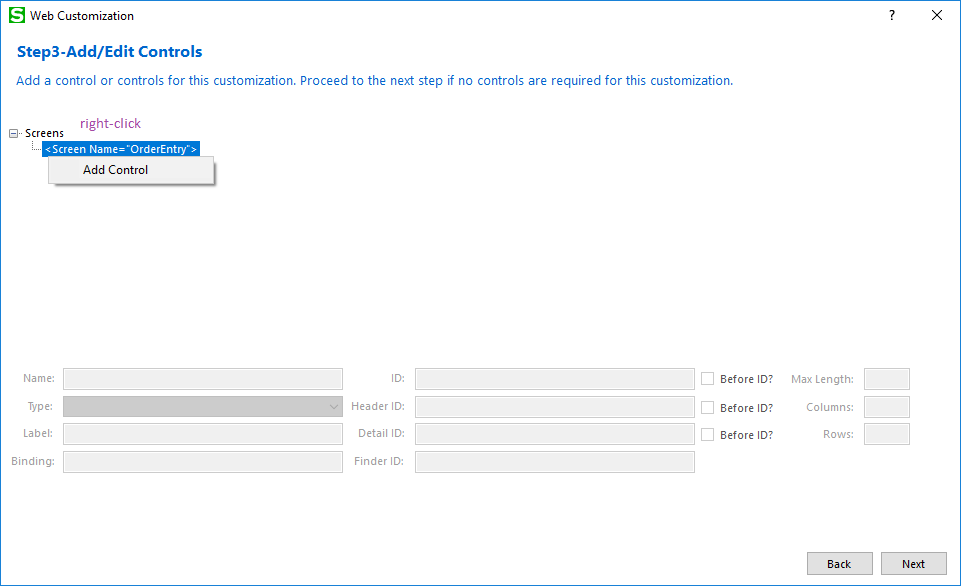
1. Populate the fields. Click on the magnifying glass next to Package to look up an existing customization or click the green plus button to generate a new GUID for a new customization.
2. Enter the name, description, and company name for the customization. Ensure the compatibility and version are as desired. A EULA can also be specified at this stage.
3. If a bootstrapper and assembly are required, the developer should proceed to the Visual Studio plugin wizard upon completion of this wizard (for advanced developers only). Modification of these two fields is disabled in this wizard.
4. Click **Next** to proceed.
5. Right click the Screens tree node to add a screen.



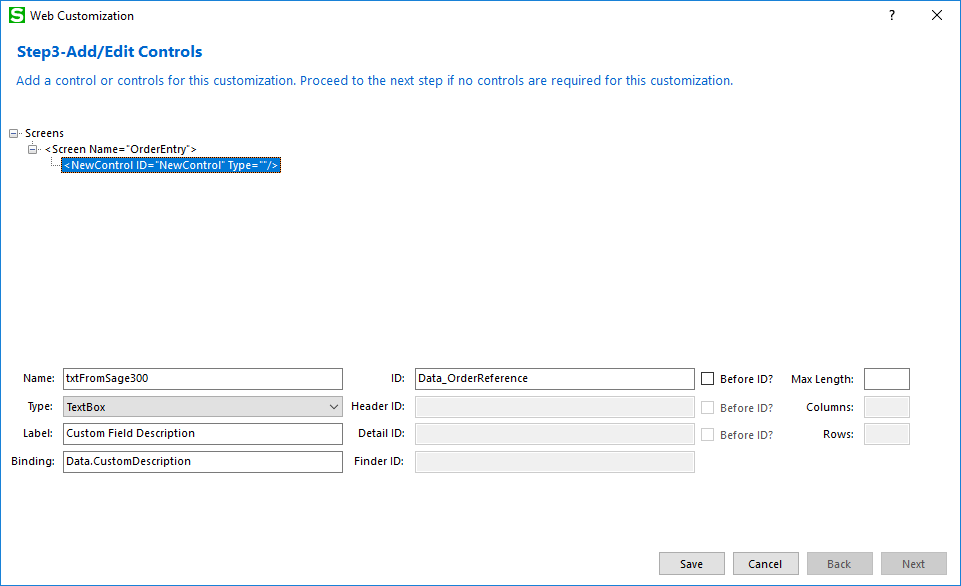
1. Select the module id, category and target screen as desired. The rest of the fields will autofill themselves. When finished, click **Save**. Repeat these steps to add another screen. Press **Next** to proceed.



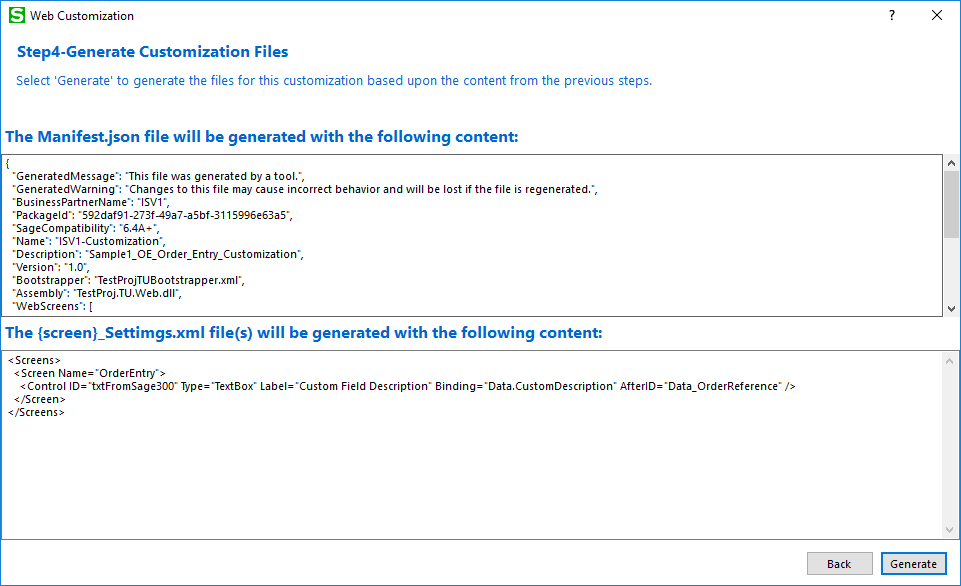
1. Right click the created screen to add a control.



1. Fill in the required fields. When finished, click **Save**. Repeat these steps to add another control. Press **Next** to proceed.



1. The wizard will preview the contents of the generated JSON and XML files. If there are any corrections to be made, press **Back** to return to the previous step. Click **Generate** to finalize the customization files.



Template files are generated as follows in the Folder specified in Step 1:

* Settings file: *{screen name}\_*Settings.xml, such as OrderEntry\_Settings.xml
* JavaScript file: *{screen name}\_*Customization.js, such as OrderEntry\_Customization.js
* Validating xsd file: ScreenConfig.xsd

**Note:** Do not change the names of the generated files. These determine which screen the customization will be applied to.

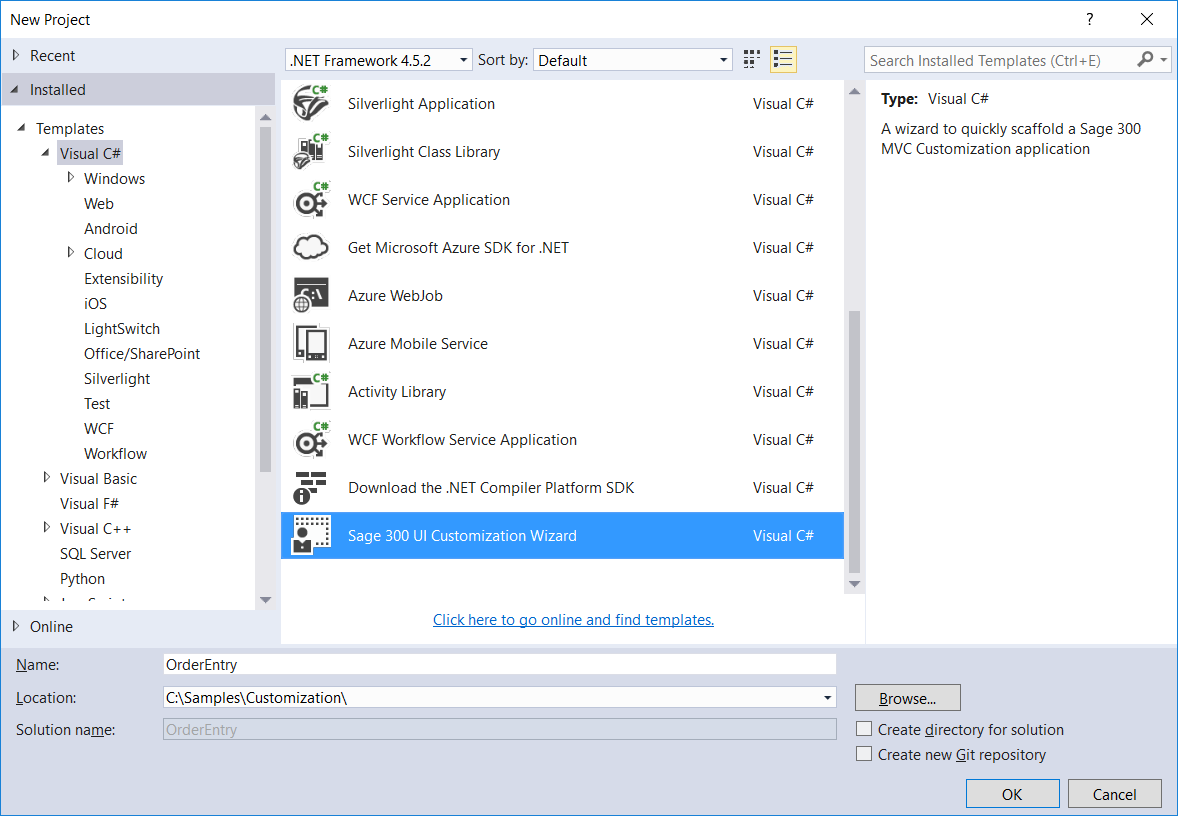
For detailed descriptions of all JSON and XML fields, refer to the XML File Specification document and the JSON File Specification document.

At this point, if a solution or endpoint customization is required, then you should proceed to the Visual Studio plugin wizard.

* 1. Create the customization project (optional)

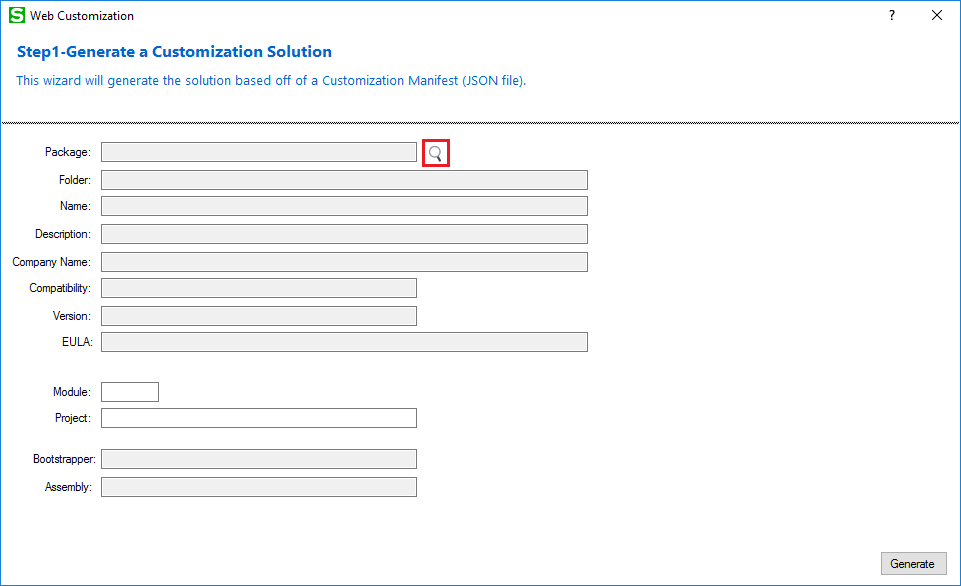
To create the project: (refer to *Sample1 - OE Order Entry Customization*)

1. Open Visual Studio, create a new project, and in the new project dialog box, select Sage 300 UI Customization Wizard.

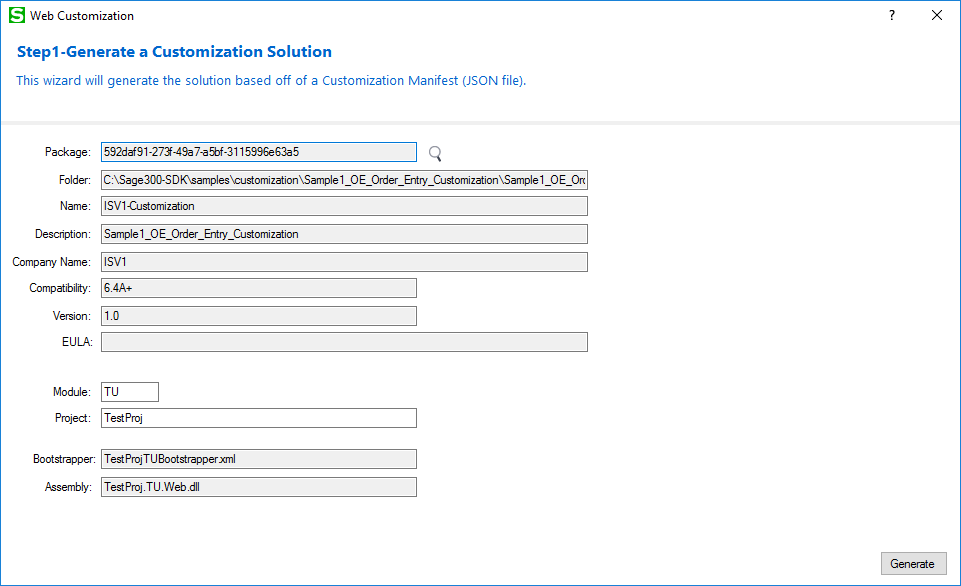


1. Enter the name and location, and then click OK.

The customization wizard page appears:



1. Click on the magnifying glass next to Package to select the manifest JSON file (see the Manifest JSON File Specifications document for additional details).
2. The rest of the fields will populate according to the contents of the file:

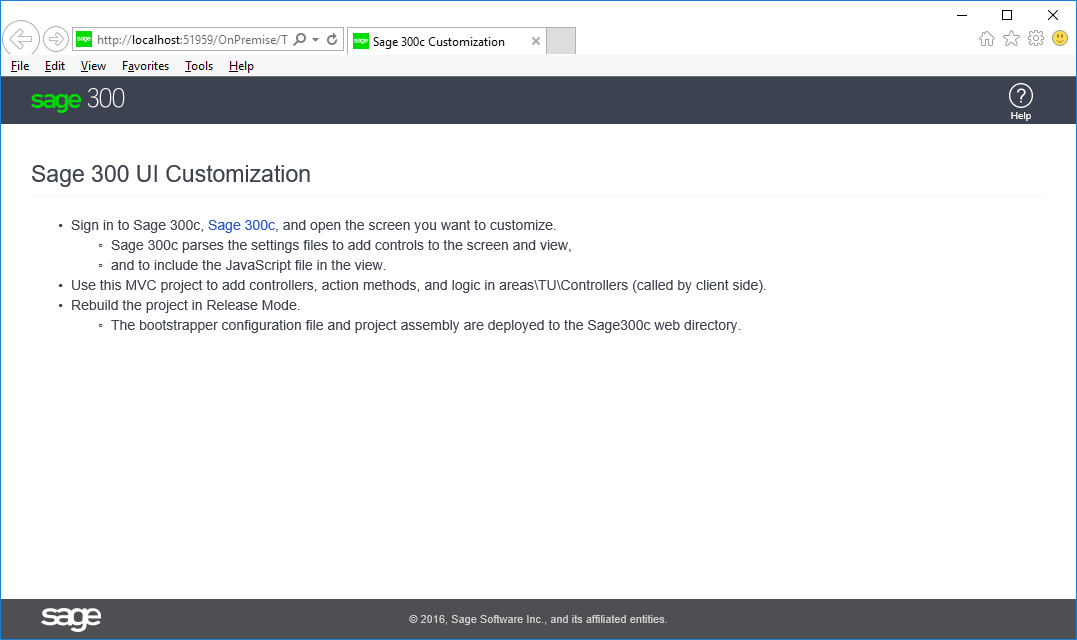


1. Click **Generate** to create the solution.
2. Select Nuget Package Manager and restore Nuget Packages
3. Compile the project and make sure it does not have any compilation errors.
   1. Open the settings and JavaScript files and start to debug

The customization wizard will generate template settings and JavaScript files under *{Sage300 installed shared data folder}\Customization folder*

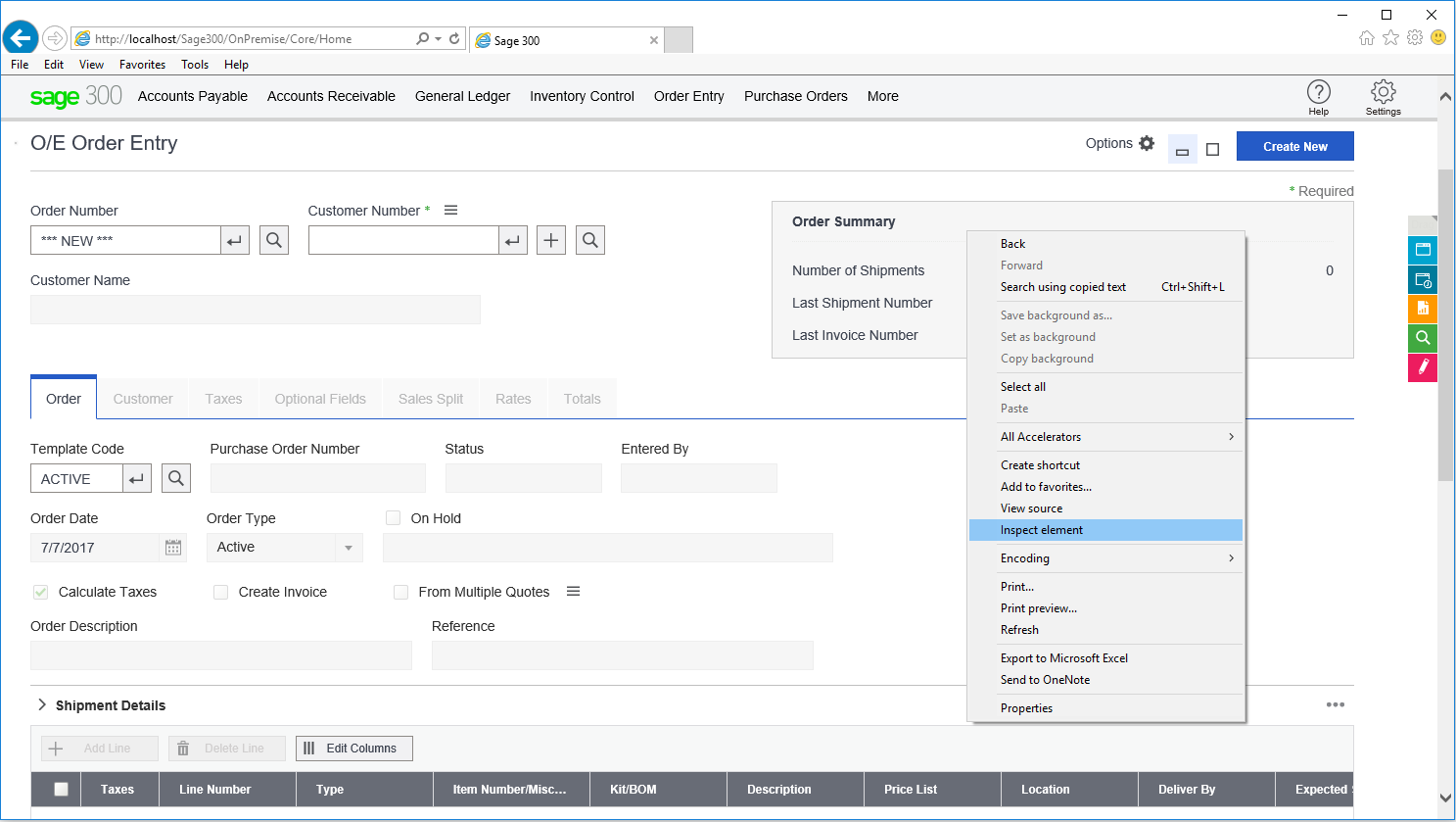
To start debugging:

1. Open the above generated screen settings 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.

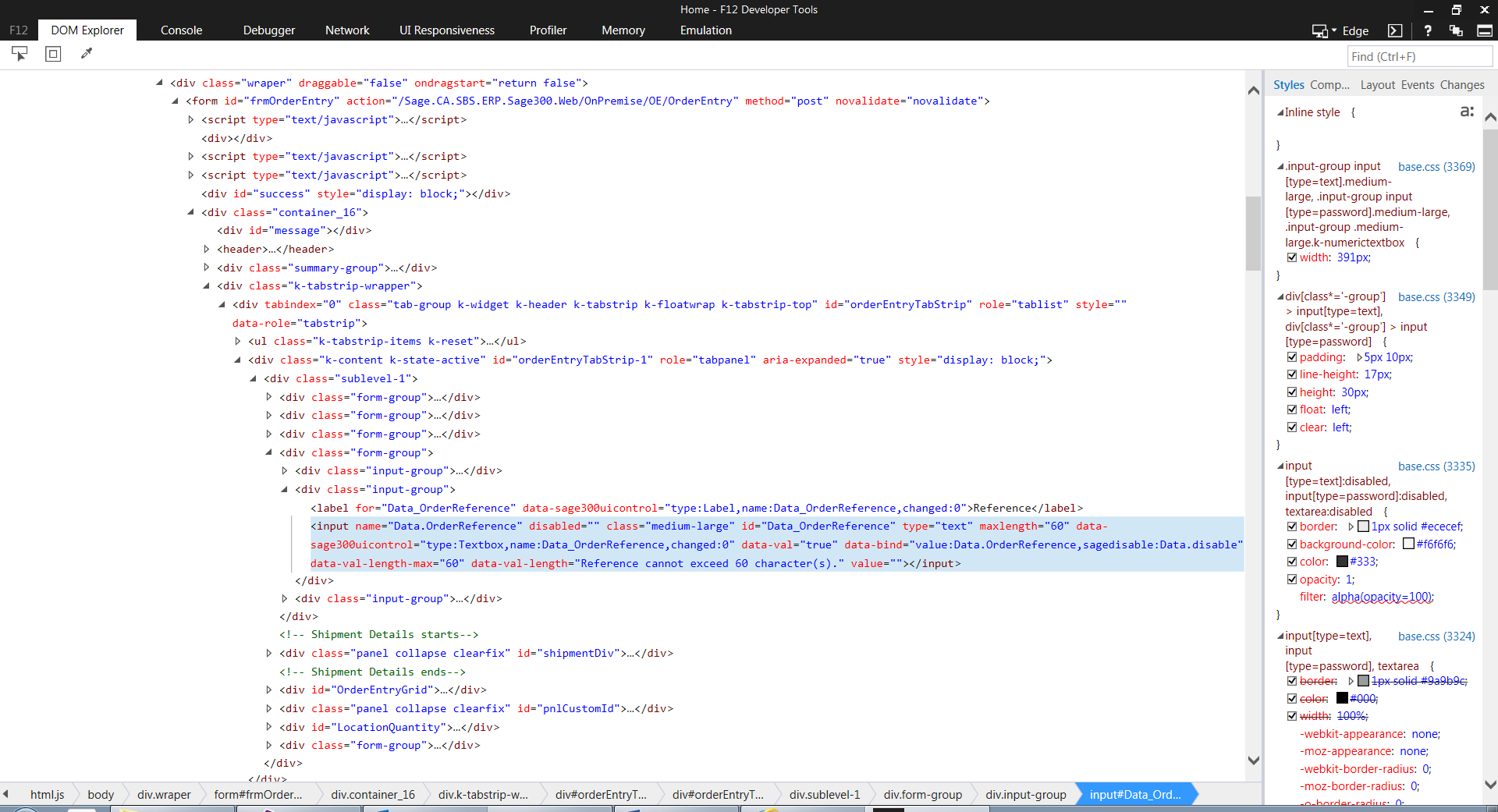


1. Sign in to the Sage 300c web application and navigate to the O/E Order Entry screen. As this screen rendering html document includes the OrderEntry\_Customization.js, when the O/E Order Entry document screen is ready, it will hit the breakpoint.
2. Development - add controls to the screen

With the O/E Order Entry screen open, use your browser’s Inspect element feature to find an existing element ID, which will be used to insert the control. For example, if you want to find the “Reference” textbox element Id, right click the textbox, and it will show:



Select the Inspect element context menu; it will show F12 window with DOM Explorer page:



The “Reference” textbox element id is “Data\_OrderReference”.

In the Order Entry sample, we will add the following controls:

* A Textbox control after the Data\_OrderReference control, binding to the Data.CustomDescription field
* A Button control before the btnPost control
* A Panel control containing several controls after the OrderEntryGrid control
* A Tab control containing several controls
  1. XML settings

In OrderEntry\_Settings.xml, add the following nodes under the root node:

<Control ID="txtFromSage300" Type ="TextBox" Label="Get from call sage 300c controller" AfterID="Data\_OrderReference" Binding="Data.CustomDescription"/>

<Control ID="btnFromSage300" Type="Button" Label="GetFromSage300c" 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" Binding=""/>

<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" MaxLength="320" Cols="150" Rows="5" Binding ="Data.CustomComments" />

</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" Binding ="" />

<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" MaxLength="320" Cols="150" Rows="5" Binding ="Data.CustomOrderComments" />

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

</Control>

**Note:** The controls in a container do not need the AfterID or BeforeID attribute since these controls are added in the container control by the layout specified in the XML file. See the screenConfig.xsd.

* Attributes
  + ID
    - A unique ID for the control (*Required*).
  + Type
    - The control type (*Required*, see below for list of control types).
  + Label
    - The text to be used as a label for the control.
  + MaxLength
    - The maximum number of characters allowed.
    - For TextBox, if you do not specify the value, the default value is 64.
    - For TextArea, if you do not specify the value, the default value is 300.
  + AfterID
    - The control will be inserted after the control with this ID.
  + BeforeID
    - The control will be inserted before the control with this ID.
  + Binding
    - Binds the controls to the Sage 300c view model data or custom view model data.
* Control Types:
  + Label
  + TextBox
  + TextArea
  + Button
  + CheckBox
  + RadioButton
  + DropDown (a Kendo UI DropDownList control)
  + DatePicker (a KendoUI DatePicker control)
  + Grid (a Kendo UI Grid control)
  + Finder
  + Panel (a container control)
  + TabPage (a container control)

(Refer to Sage300SDK\_WebScreenCustomization\_ControlConfigurationXMLFileSpecifications.docx for the {screen\_name}\_Settings.xml file specifications)

* 1. Initialize controls

Some controls use Kendo UI controls, which require initialization in the JavaScript code.

In the Order Entry sample, we will initialize three drop-down lists and one date picker control.

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({

});

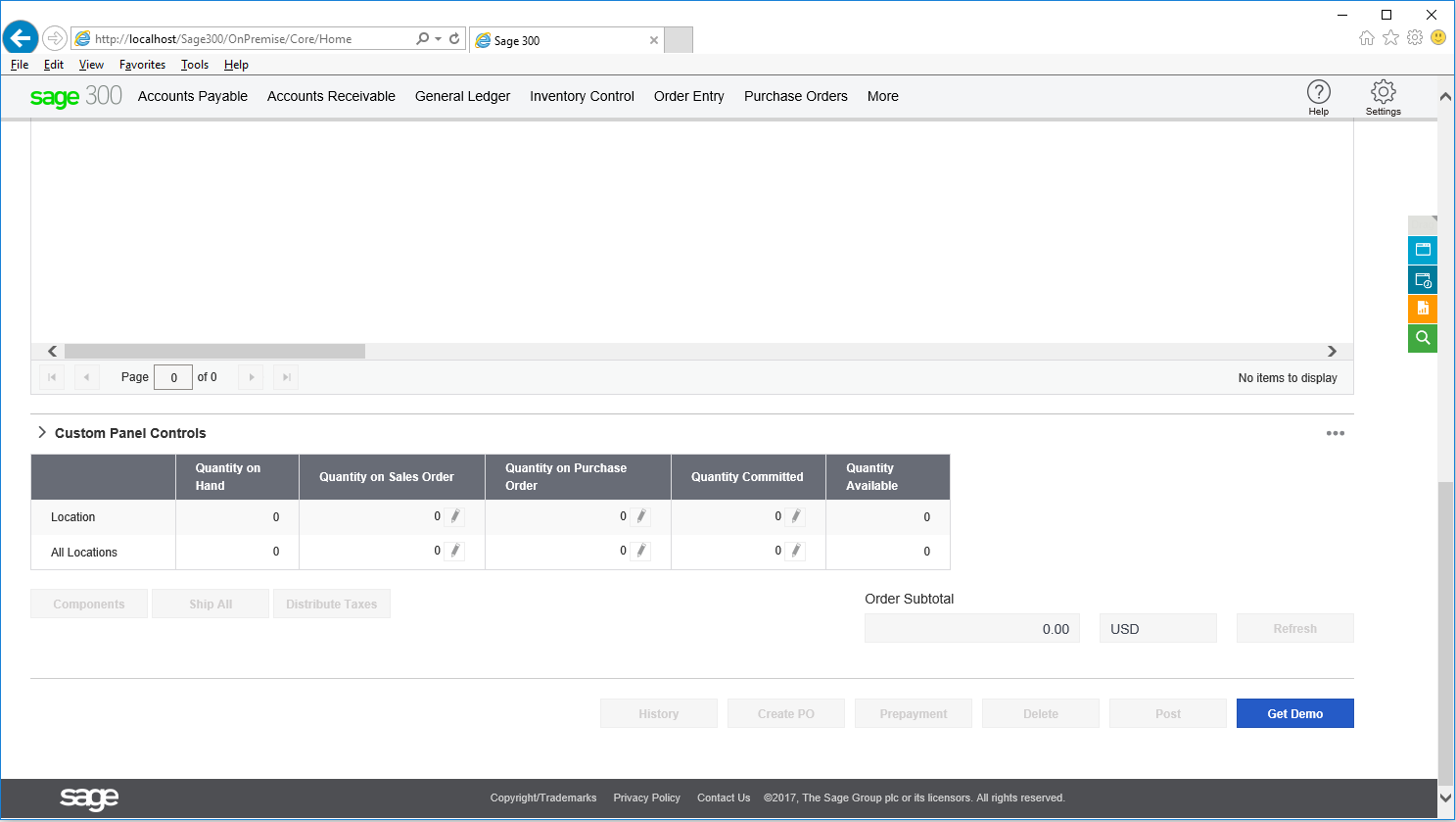
Add the following code snippet in the initDatePicker function:

$("#dtPickerCustom").kendoDatePicker({

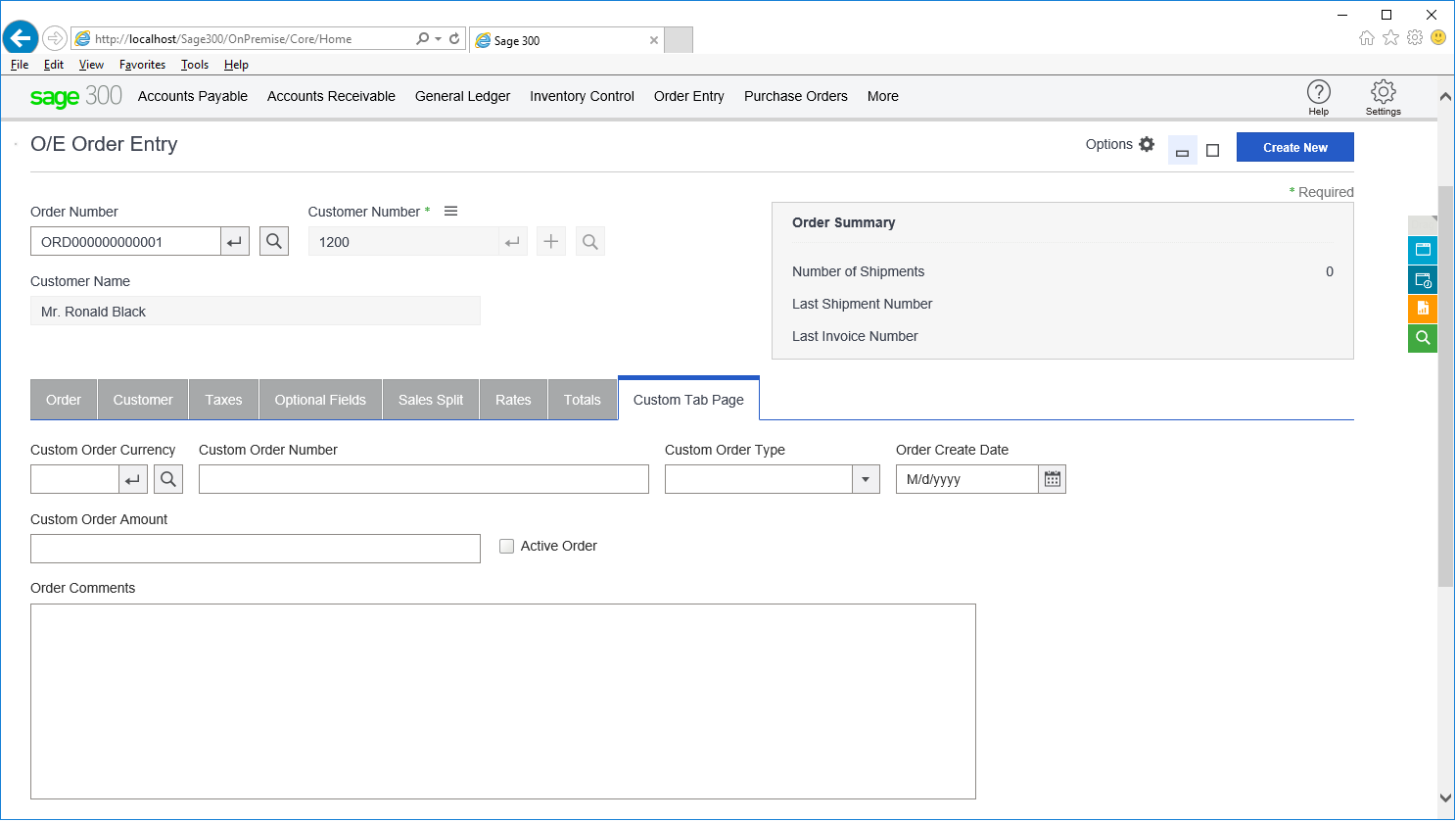
});

* 1. Customized Order Entry screen

When the Order Entry 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. Development - add JavaScript
   1. Initialize controls

Some controls will require initialization. For example, A Panel control should be expandable, a Tab Page should be disabled when the screen first opens, etc.

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

$("#numericOrderAmount").kendoNumericTextBox({

spinners : false,

decimals: 3

});

1. Add code to initialize the Kendo UI Grid, Dropdown and other controls as needed. Refer to the sample code for details.
2. 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. 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);

}

}

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 sample, an Ajax call will be made to the Common Services Tax Authorities controller to get a currency description.

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);

});

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"));

},

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

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.

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

1. Development - add custom model, view model, and controller

Use the Sage 300 UI Customization Wizard to create a template project in order to add a custom model, view model, and controller under the Areas\CU\Models and Areas\CU\Controllers folders. 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 Model 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 {companyName}CUBootstrapper.xml and {companyName}.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 pack the customization files to a package (zip file), as it will be used by 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 package will include the Manifest.json file which describes the customization package.

(Refer to Sage300SDK\_WebScreenCustomization\_ManifestJSONFileSpecifications.docx for the Manifest.json file specifications)

Example:

{

"BusinessPartnerName": "ISV1",

"PackageId": "Unique\_GUID",

"SageCompatibility": "6.4A+",

"Name": "ISV-Customization",

"Description": "ISV1's customization for the web screens",

"Version": "1.0",

"EULA": "<license file>",

"Bootstrapper": "ISV1CUBootstrapper.xml",

"Assembly": "ISV1.Web.dll",

"WebScreens":

[

{

"ScreenName": "CutsomOECopyOrder",

"ScreenDescription": "Order Entry Transactions Copy Orders",

"TargetScreen": "CopyOrder",

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

"ControlsBehavior": "CopyOrder\_Customization.js"

},

{

"ScreenName": "OrderEntry\_Custom",

"ScreenDescription": "OrderEntry\_Custom Description",

"TargetScreen": "OrderEntry",

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

"ControlsBehavior": "OrderEntry\_Customization.js"

},

{

"ScreenName": "CSTaxAuthorities",

"ScreenDescription": "Common Services Tax Authorities",

"TargetScreen": "TaxAuthorities",

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

"ControlsBehavior": "TaxAuthorities\_Customization.js"

}

]

}

The JSON file is a simple text file, and care must be taken to ensure that the content is accurate, as the Customization Admin screen will not import customization packages with invalid manifests.

The example JSON file includes configuration and JavaScript files for three custom screens.

The PackageId value must be unique. Use a GUID value. (The GUID used in the samples are for demo purpose only)

If assemblies are to be deployed, include the files of the assembly and the bootstrapper.xml file in the package, and specify them in this manifest file.