Sage 300 Web Screens SDK

Creating Payment Codes Screen - Tutorial

June 2021

The MIT License (MIT)

Copyright © 2021 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](#_Toc73458349)

[2. A/R Payment Codes 5](#_Toc73458350)

[2.1 Desktop Screen 5](#_Toc73458351)

[2.2 Web Screen 5](#_Toc73458352)

[3. Solution Wizard 6](#_Toc73458353)

[3.1 Solution Information 7](#_Toc73458354)

[3.2 Kendo UI Information 8](#_Toc73458355)

[3.3 Resource Files 8](#_Toc73458356)

[3.4 Generate Solution 10](#_Toc73458357)

[3.5 Set Login.aspx as the Start Page 10](#_Toc73458358)

[4. Code Generation Wizard 11](#_Toc73458359)

[4.1 Select Code Type and Credentials 12](#_Toc73458360)

[4.2 Add/Edit Flat Entities 13](#_Toc73458361)

[4.3 Select Options 14](#_Toc73458362)

[4.4 Generate Code 15](#_Toc73458363)

[4.5 Wizard Completion 16](#_Toc73458364)

[5. Complete the Screen 17](#_Toc73458365)

[5.1 Payment Codes Business Entity Interface 17](#_Toc73458366)

[5.2 Payment Codes Service Interface 18](#_Toc73458367)

[5.3 Payment Codes Entity Service 19](#_Toc73458368)

[5.4 Payment Codes Repository 21](#_Toc73458369)

[5.5 Payment Codes Model 25](#_Toc73458370)

[5.6 Payment Codes View Model 27](#_Toc73458371)

[5.7 Payment Codes Internal Controller 28](#_Toc73458372)

[5.8 Payment Codes Controller 31](#_Toc73458373)

[5.9 Payment Codes Partial Razor View 35](#_Toc73458374)

[5.10 Payment Codes Repository JavaScript 37](#_Toc73458375)

[5.11 Payment Codes Knockout Binding JavaScript 39](#_Toc73458376)

[5.12 Payment Codes Behaviour JavaScript 40](#_Toc73458377)

[6. Review the Screen 46](#_Toc73458378)

1. Overview

This document is intended to serve as a tutorial for creating the AR Payment Codes web screen with the Sage 300 Web SDK Wizard.

The wizard creates the solution, the required framework files and generates as much code as possible. However, the generated code specific to the AR Payment Codes screen is only for the key field to the screen plus framework components, such as: Create New Button, Save Button, Delete Button, Options Menu, Finder, etc. Thus, there is additional manual work required to complete the screen to include the remaining required controls.

Therefore, this tutorial will:

* Use the Solution Wizard to create the Visual Studio solution for the Valued Partner Company utilizing the TU module
* Use the Code Generation Wizard to create the code files and screen for the AR Payment Codes Business View (AR0012)
* Use Visual Studio to complete the code required to have a fully functioning AR Payment Codes screen

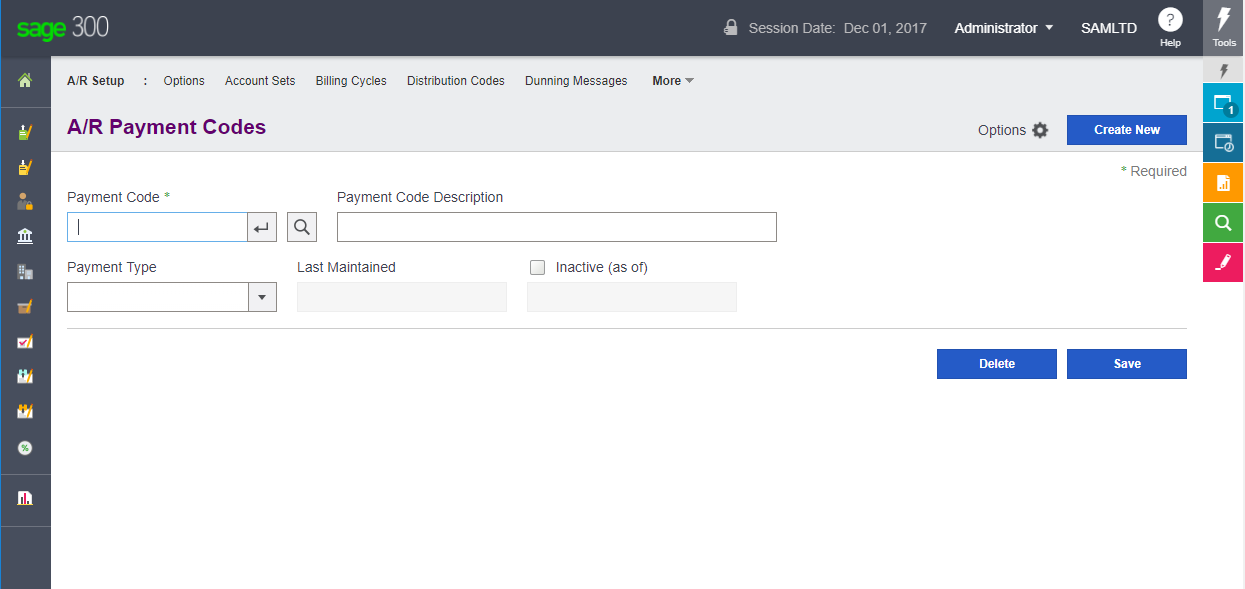
This document’s content is derived from the following documents and these documents should be referred to for further and complete documentation:

* docs\wizards\Sage300SDK\_SolutionWizard.docx
* docs\wizards\Sage300SDK\_CodeGenerationWizard.docx
* docs\development\Sage300SDK\_CreatingSetupWebScreen.docx

1. A/R Payment Codes
   1. Desktop Screen



* 1. Web Screen



1. Solution Wizard

* Launch Visual Studio

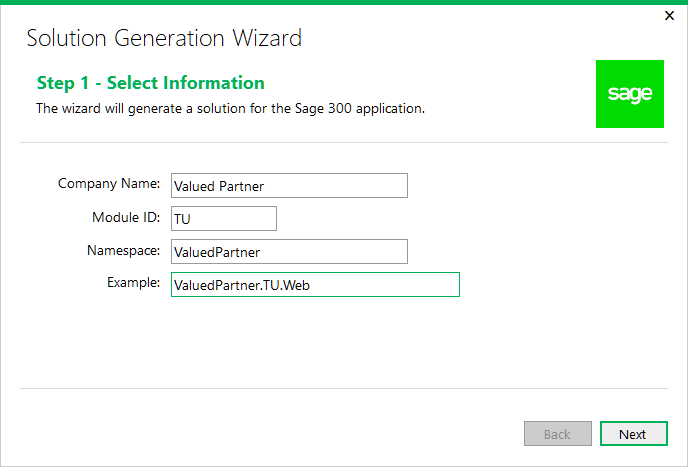
Note: It is recommended to run Visual Studio with administrator privileges

* Select **New/Project** to display the following dialog:



* Enter **ValuedPartner** for the solution name and change the location if needed.
* Press **OK** to create the solution and display the Sage 300 Solution Wizard
  1. Solution Information

After supplying the information required to create a new project (solution), the following dialog box appears:



* Enter Valued Partner for the Company Name field

Used for copyright information and to provide a default value for the Namespace field.

* Enter TU for the Module ID field

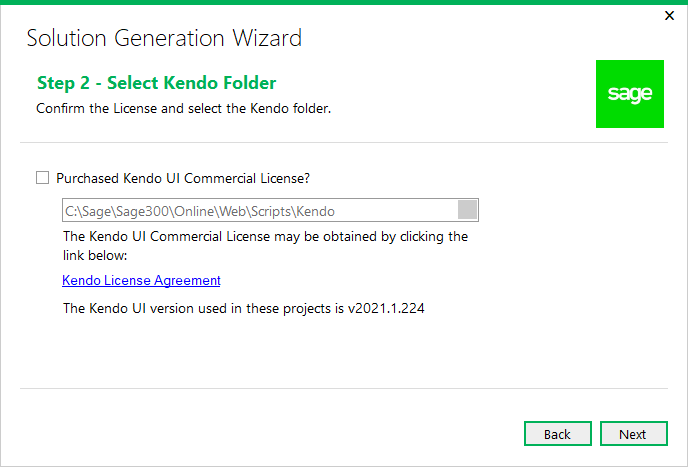
Used as a Module designator that will be used to segregate the projects, and will also become part of the namespace.

* ValuedPartner is defaulted for the Namespace field

Used as the base Namespace value. The namespace will become a composite of this field and the Module ID. This field is defaulted from the Company Name field but may be overridden.

Click Next to proceed.

* 1. Kendo UI Information



Select the Purchased Kendo Commercial License checkbox.

* Purchased Kendo Commercial License

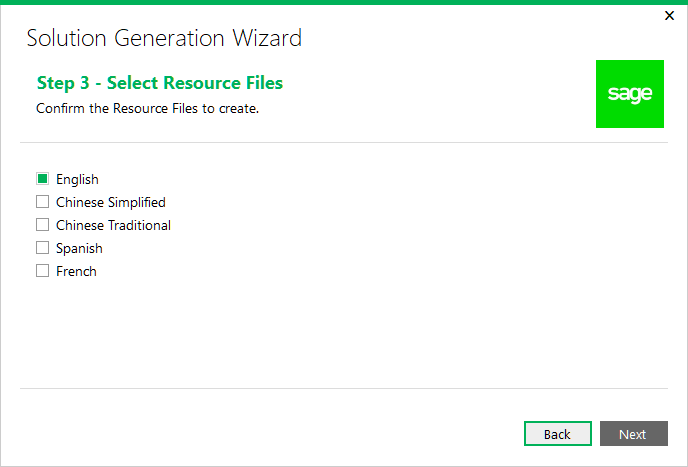
This checkbox must be selected or the Solution Wizard will not proceed. This is the acknowledgment that the user/consumer has purchased a Kendo License since Sage does not distribute the Kendo files for development purposes.

* Kendo Folder

Defaults to the location of the Sage 300 Web installation in order to consume that version of the Kendo file. An alternate folder may be entered, but the Kendo folder must have the version specified as being compatible with Sage 300c Web Screens

Click Next to proceed.

* 1. Resource Files



The following information is required by the wizard:

* English

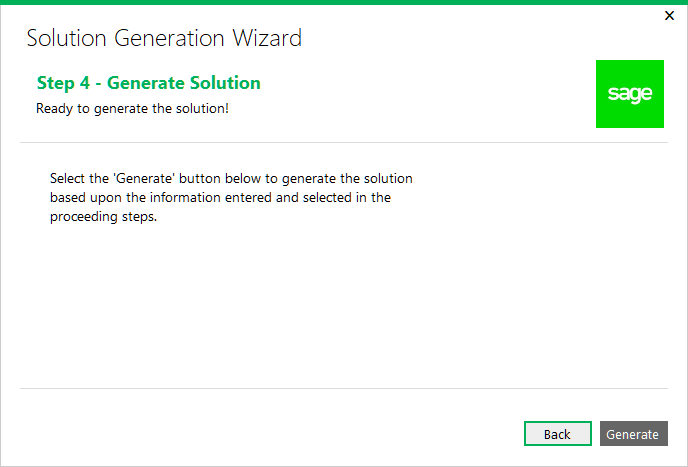
This checkbox is selected by default and cannot be unselected in order to ensure that at least the English Resource Files will be generated in the Resources Project.

* Chinese Simplified, Chinese Traditional, Spanish and French

Select these checkboxes to generate the appropriate language files in the Resources Project. For these optional languages, the keys will be generated with blank values. These blank values must be translated prior to deployment.

Click Next to proceed.

* 1. Generate Solution



Click Generate to generate the solution.

* 1. Set Login.aspx as the Start Page

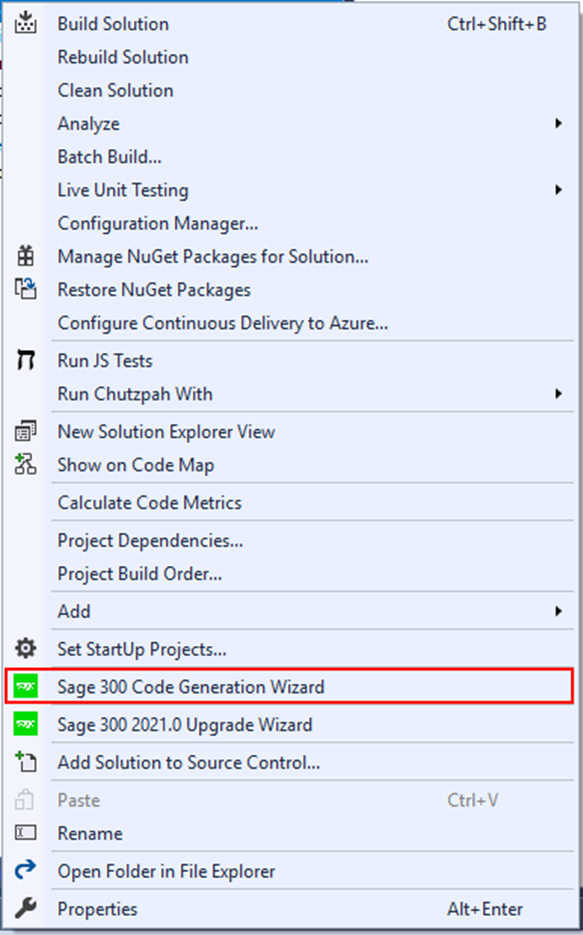
In Sage 300 version 2022, a login page was added to the SDK Samples and generated partner solutions, which replaced hardcoded login credentials in the Global.asax.cs file.

In the Solution Explorer, Right-Click the **Login.aspx** file in the Web project and select **Set As Start Page**.

Failure to set this ASPX file for the solution will result in a failure when running the solution in Debug mode.

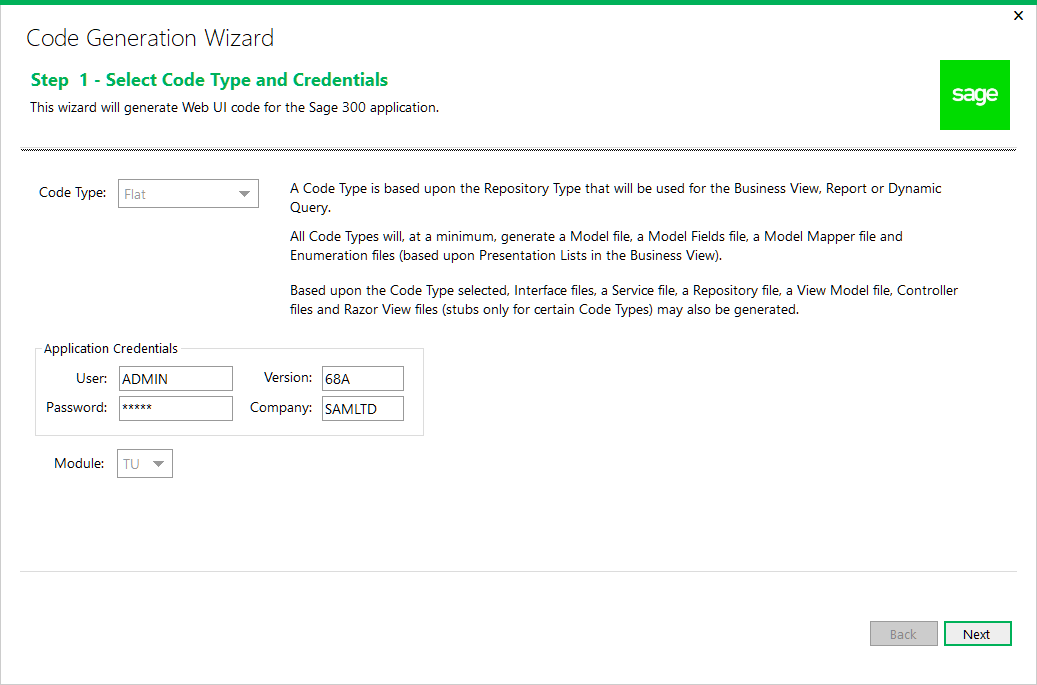
1. Code Generation Wizard

Within Visual Studio, right-click on the **ValuedPartner** solution from the Solution Explorer to display the following context menu:



Select the **Sage 300 Code Generation Wizard** option.

* 1. Select Code Type and Credentials

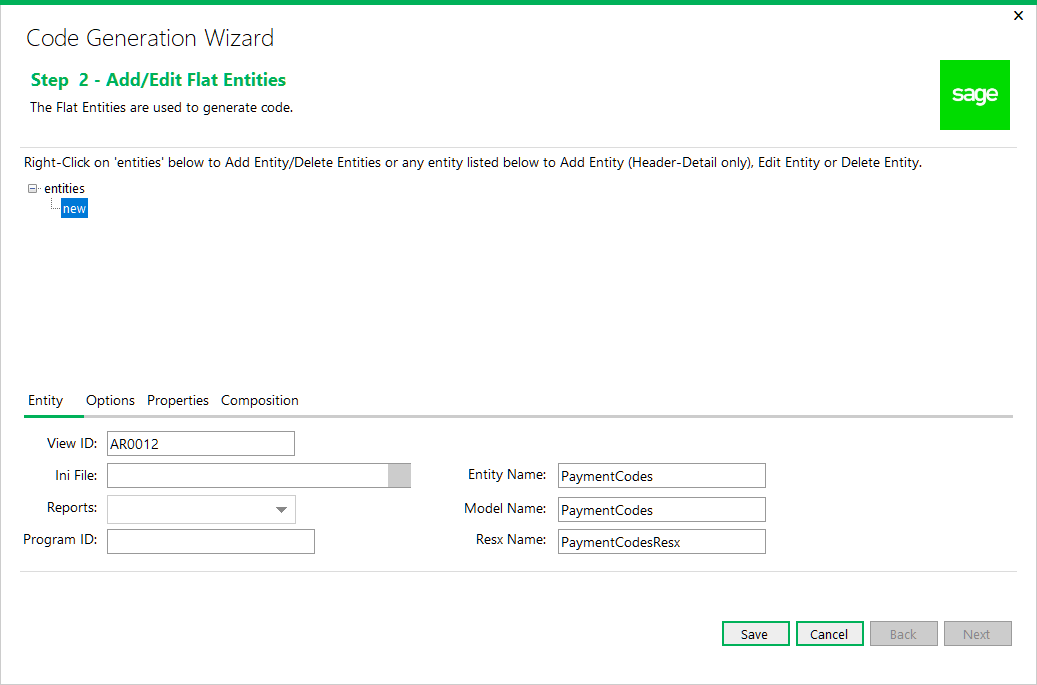


Select Flat code type because this is a simple setup screen (stateless).

Ensure the **Application Credentials** and **Module** are correct and change them if necessary.

Click **Next** to proceed.

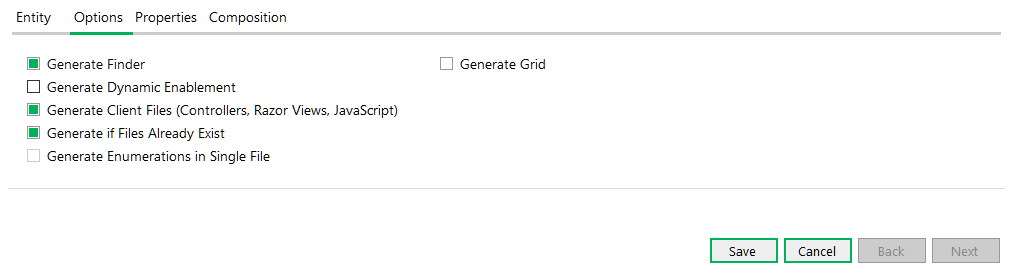
* 1. Add/Edit Flat Entities



* Right-click on the ‘**entities**’ tree node and select **Add Entity** to add a new entity.
* Enter **AR0012** for the View ID field.
* After entering the Business View, the Entity Name will default to **PaymentCode** for the Entity being created, which is generated from the Business View’s description. Change this value to **PaymentCodes**.
* Change the Model Name to **PaymentCodes** as well.
* The Resx Name is defaulted from the Entity Name and can be overridden if needed but for the purposes for this tutorial, keep it as is.
* Click **Save**.

Click on the Options tab.

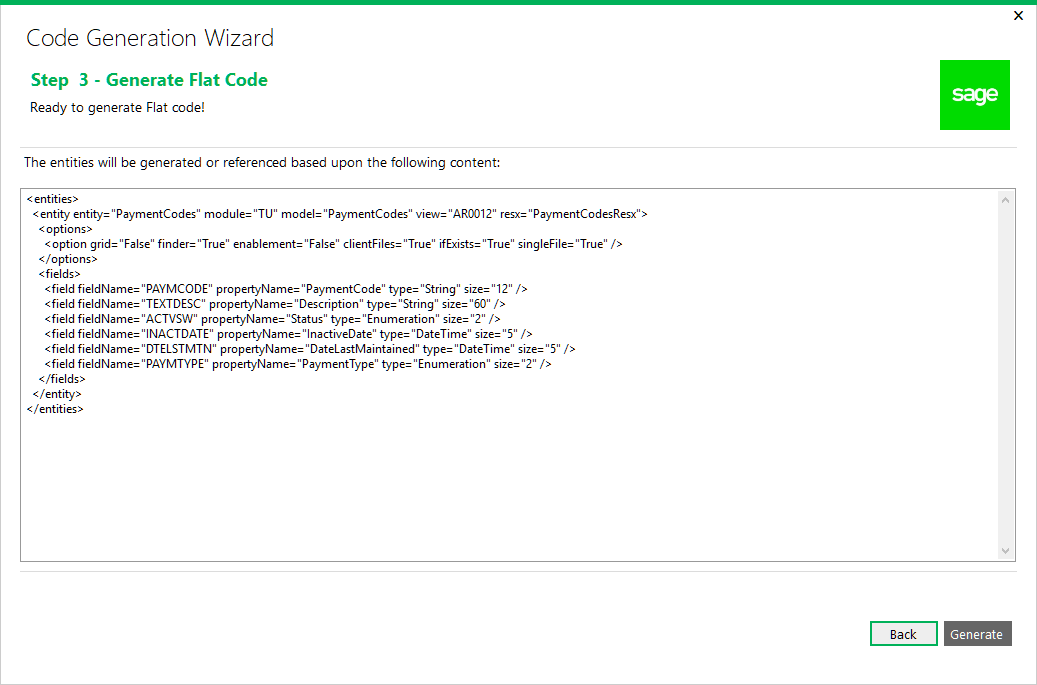
* 1. Select Options



* Select the Generate Finder check box to generate the Finder’s internal controller for the view selected.
* The Generate Dynamic Enablement check box is optional and will generate the DynamicAttributes property in the Model Fields class. This provides a reverse mapping of the Server Fields (Business View) to the Model Properties (Business Entity) for those fields in the Business View that have the CheckEditable attribute.
  + Do not select the Generate Dynamic Enablement checkbox
* The **Generate Client Files** check box, if selected, will generate the client files for a Business View (controllers, Razor Views, and JavaScript files).
* Select the Generate if Files Already Exist check box to allow you to skip files that are generated in case they already exist. An example of this would be if you ran the wizard twice on the same Business View.

Click **Next** to proceed.

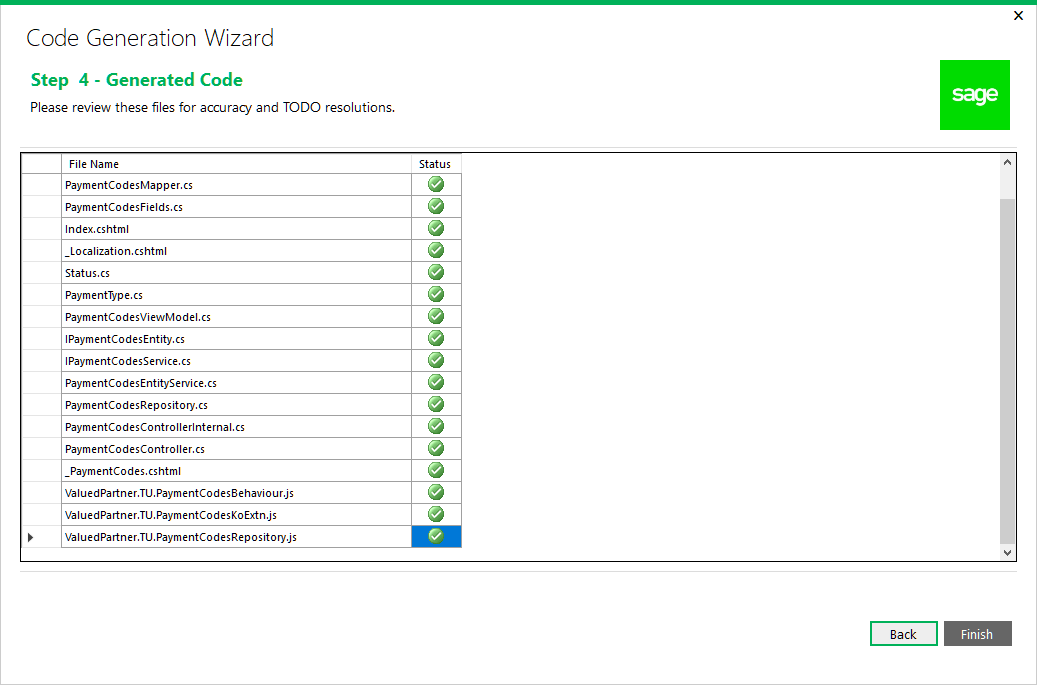
* 1. Generate Code



Verify that the XML shown in the preview is correct.

Click **Next** to proceed.

* 1. Wizard Completion



When the wizard has completed generating the code files based upon the wizard’s inputs, the list of files is displayed. These files, based upon the Generate if Files Already Exist option, have been added to the appropriate projects within the solution.

Note: Nuget Packages may need to be downloaded if your Visual Studio Options are not set to download during build.

Select the **Build\Build Solution** option to compile the solution in order to run the screen.

1. Complete the Screen

The generation activities up to this point have dealt with creating the components required for the A/R Payment Codes screen and implementing base behaviors and minimal functionality.

The screen is able to be compiled and is functional.

This section will deal with specific changes to complete the A/R Payment Codes screen.

* 1. Payment Codes Business Entity Interface

Access the **IPaymentCodesEntity.cs** class and add the **UpdateInactiveStatus** and **IsModuleActive** routines to the class (the code to be added is highlighted in yellow).

|  |
| --- |
| // Copyright (c) 2020 Valued Partner All rights reserved.  #region Namespace  using ValuedPartner.TU.Models;  using Sage.CA.SBS.ERP.Sage300.Common.Interfaces.Repository;  #endregion  namespace ValuedPartner.TU.Interfaces.BusinessRepository  {  /// <summary>  /// Interface IPaymentCodesEntity  /// </summary>  /// <typeparam name="T">Where T is type of <see cref="PaymentCodes"/></typeparam>  public interface IPaymentCodesEntity<T> : IBusinessRepository<T>, ISecurity  where T : PaymentCodes, new()  {  /// <summary>  /// Update status  /// </summary>  /// <param name="model">Model for PaymentCodes</param>  /// <returns>Model for PaymentCodes</returns>  T UpdateInactiveStatus(T model);  /// <summary>  /// Check whether module is active or not.  /// </summary>  /// <param name="moduleId">module Id</param>  /// <returns>True if module is active else false</returns>  bool IsModuleActive(string moduleId);  }  } |

* 1. Payment Codes Service Interface

Access the **IPaymentCodesService.cs** class and add the **UpdateInactiveStatus** and **IsModuleActive** routines to the class (the code to be added is highlighted in yellow).

|  |
| --- |
| // Copyright (c) 2020 Valued Partner All rights reserved.  #region Namespace  using ValuedPartner.TU.Models;  using Sage.CA.SBS.ERP.Sage300.Common.Interfaces.Service;  #endregion  namespace ValuedPartner.TU.Interfaces.Services  {  /// <summary>  /// Interface IPaymentCodesService  /// </summary>  /// <typeparam name="T">Where T is type of <see cref="PaymentCodes"/></typeparam>  public interface IPaymentCodesService<T> : IEntityService<T>, ISecurityService  where T : PaymentCodes, new()  {  /// <summary>  /// Update status  /// </summary>  /// <param name="model">Model for PaymentCodes</param>  /// <returns>Model for PaymentCodes</returns>  T UpdateInactiveStatus(T model);  /// <summary>  /// Check whether module is active or not.  /// </summary>  /// <param name="moduleId">module Id</param>  /// <returns>True if module is active else false</returns>  bool IsModuleActive(string moduleId);  }  } |

* 1. Payment Codes Entity Service

Access the **PaymentCodesEntityService.cs** class and add the UpdateInactiveStatus and IsModuleActive routines to the class (the code to be added is highlighted in yellow).

|  |
| --- |
| // Copyright (c) 2020 Valued Partner All rights reserved.  #region Namespace  using System;  using Sage.CA.SBS.ERP.Sage300.Common.Models;  using Sage.CA.SBS.ERP.Sage300.Common.Services.Base;  using ValuedPartner.TU.Interfaces.BusinessRepository;  using ValuedPartner.TU.Interfaces.Services;  using ValuedPartner.TU.Models;  #endregion  namespace ValuedPartner.TU.Services  {  /// <summary>  /// Class for PaymentCodesEntityService  /// </summary>  /// <typeparam name="T">Where T is type of <see cref="PaymentCodes"/></typeparam>  public class PaymentCodesEntityService<T> : FlatService<T, IPaymentCodesEntity<T>>, IPaymentCodesService<T>  where T : PaymentCodes, new()  {  #region Constructor  /// <summary>  /// Constructor for PaymentCodes  /// </summary>  /// <param name="context">Request Context</param>  public PaymentCodesEntityService(Context context)  : base(context)  {  }  #endregion  #region Public  /// <summary>  /// Update status  /// </summary>  /// <param name="model">Model for PaymentCodes</param>  /// <returns>Model for PaymentCodes</returns>  public virtual T UpdateInactiveStatus(T model)  {  if (model == null)  {  throw new ArgumentNullException("model");  }  using (var repository = Resolve<IPaymentCodesEntity<T>>())  {  return repository.UpdateInactiveStatus(model);  }  }  /// <summary>  /// Check whether module is active or not.  /// </summary>  /// <param name="moduleId">module Id</param>  /// <returns>True if module is active else false</returns>  public bool IsModuleActive(string moduleId)  {  using (var repository = Resolve<IPaymentCodesEntity<T>>())  {  return repository.IsModuleActive(moduleId);  }  }  #endregion  }  } |

* 1. Payment Codes Repository

It is important to mention at this point a concept called Benign Programming or Agnostic Programming. The more generic the class, the less amount of refactoring is to be required if a particular class is copied and pasted into another class.

What does this mean? If the screen has a single business entity, the wizard will name it \_**businessEntity** instead of something specific to the entity. Parameters will be generic unless multiple entities prevents this (such as **id** vs. **paymentCode**) (the code to be added is highlighted in yellow).

Access the **PaymentCodesRepository.cs** class and add the **UpdateInactiveStatus** and **IsModuleActive** routines to the class.

|  |
| --- |
| // Copyright (c) 2020 Valued Partner All rights reserved.  #region Namespace  using System;  using System.Linq.Expressions;  using Sage.CA.SBS.ERP.Sage300.Common.BusinessRepository;  using Sage.CA.SBS.ERP.Sage300.Common.BusinessRepository.Base;  using Sage.CA.SBS.ERP.Sage300.Common.Interfaces.Entity;  using Sage.CA.SBS.ERP.Sage300.Common.Models;  using Sage.CA.SBS.ERP.Sage300.Common.Utilities;  using Sage.CA.SBS.ERP.Sage300.Common.Models.Enums;  using Sage.CA.SBS.ERP.Sage300.Common.Models.Enums.ExportImport;  using Sage.CA.SBS.ERP.Sage300.Common.Models.ExportImport;  using ValuedPartner.TU.BusinessRepository.Mappers;  using ValuedPartner.TU.Interfaces.BusinessRepository;  using ValuedPartner.TU.Models;  #endregion  namespace ValuedPartner.TU.BusinessRepository  {  /// <summary>  /// Class PaymentCodes Repository  /// </summary>  /// <typeparam name="T">Where T is type of <see cref="PaymentCodes"/></typeparam>  public class PaymentCodesRepository<T> : FlatRepository<T>, IPaymentCodesEntity<T>  where T : PaymentCodes, new()  {  #region Variables  /// <summary>  /// Mapper  /// </summary>  private ModelMapper<T> \_mapper;  /// <summary>  /// Business Entity  /// </summary>  private IBusinessEntity \_businessEntity;  #endregion  #region Constructor  /// <summary>  /// Constructor for PaymentCodes  /// </summary>  /// <param name="context">Context</param>  public PaymentCodesRepository(Context context)  : base(context, new PaymentCodesMapper<T>(context), ActiveFilter)  {  SetFilter(context);  }  /// <summary>  /// Constructor for PaymentCodes  /// </summary>  /// <param name="context">Context</param>  /// <param name="session">Business Entity Session</param>  public PaymentCodesRepository(Context context, IBusinessEntitySession session)  : base(context, new PaymentCodesMapper<T>(context), ActiveFilter, session)  {  SetFilter(context);  }  #endregion  #region Protected/public methods  /// <summary>  /// Additional Access Check for Export and Import  /// </summary>  /// <returns>User Access</returns>  public override UserAccess GetAccessRights()  {  var userAccess = base.GetAccessRights();  if (SecurityCheck(Security.TUImport))  {  userAccess.SecurityType |= SecurityType.Import;  }  if (SecurityCheck(Security.TUExport))  {  userAccess.SecurityType |= SecurityType.Export;  }  return userAccess;  }  /// <summary>  /// Create entities for repository  /// </summary>  /// <returns>Business Entity</returns>  protected override IBusinessEntity CreateBusinessEntities()  {  CreateBusinessEntitiesInternal();  return \_businessEntity;  }  /// <summary>  /// Get Update Expression  /// </summary>  /// <param name="model">Model for PaymentCodes</param>  /// <returns>Expression</returns>  protected override Expression<Func<T, bool>> GetUpdateExpression(T model)  {  return entity =>  (entity.PaymentCode.StartsWith(model.PaymentCode));  }  /// <summary>  /// Update status  /// </summary>  /// <param name="model">Model for PaymentCodes</param>  /// <returns>Model for PaymentCodes</returns>  public virtual T UpdateInactiveStatus(T model)  {  CheckRights(GetAccessRights(), SecurityType.Modify);  var filter = GetUpdateExpression(model);  if (Search(\_businessEntity, filter))  {  \_mapper.Map(model, \_businessEntity);  \_businessEntity.SetValue(PaymentCodes.Index.Status, model.Status, true);  }  return \_mapper.Map(\_businessEntity);  }  /// <summary>  /// Check whether module is active or not.  /// </summary>  /// <param name="moduleId">module Id</param>  /// <returns>True if module is active else false</returns>  public bool IsModuleActive(string moduleId)  {  return IsApplicationActive(moduleId);  }  #endregion  #region Import/Export methods  /// <summary>  /// Get export or import business entity property  /// </summary>  /// <param name="option">export/import option, default to null</param>  /// <param name="isExport">true if for export, default to false</param>  /// <returns>Business Entity Property</returns>  public override BusinessEntityProperty GetExportImportBusinessEntityProperty(string option = null, bool isExport = false)  {  return new BusinessEntityProperty(PaymentCodes.EntityName, ViewKeyType.UserSpecified);  }  #endregion  #region Private methods  /// <summary>  /// ActiveFilter Condition  /// </summary>  /// <value>The active filter</value>  private static Expression<Func<T, bool>> ActiveFilter  {  get { return null; }  }  /// <summary>  /// Creates the business entities  /// </summary>  private void CreateBusinessEntitiesInternal()  {  \_businessEntity = OpenEntity(PaymentCodes.EntityName);  }  /// <summary>  /// Set Filter  /// </summary>  /// <param name="context">Context</param>  private void SetFilter(Context context)  {  ValidRecordFilter = null;    // TODO: Evaluate the ValidRecordFilter expression and uncomment if desired. This is a validator expression  // used by the framework to prevent 'invalid' records from being returned. However, the  // underlying Business View and database already prevent this type of behavior and  // will cause 'valid' records to be deemed 'invalid' and will not return them to the View  // Model (i.e. 0 value keys).  // TODO: Delete TODO statements when complete  // ValidRecordFilter = (model =>  // !string.IsNullOrEmpty(model.PaymentCode));  \_mapper = new PaymentCodesMapper<T>(context);  }  #endregion  }  } |

* 1. Payment Codes Model

Access the **PaymentCodes.cs** class

* Evaluate the attributes that have been added to the generated properties.
* Evaluate the property names that have been generated by the wizard.
* These generated names are not set in stone.
* These names have been generated from the description of the field/property in the business view. If it does not make sense, is not properly cased or whatever, it is the developer’s responsibility to change it.
* Additional attributes may be added later in the development process.

Add the **RegularExpression** annotation to the **PaymentCode** property (the code to be added is highlighted in yellow).

|  |
| --- |
| // Copyright (c) 2020 Valued Partner All rights reserved.  #region Namespace  using System;  using System.ComponentModel.DataAnnotations;  using Sage.CA.SBS.ERP.Sage300.Common.Models;  using Sage.CA.SBS.ERP.Sage300.Common.Models.Attributes;  using Sage.CA.SBS.ERP.Sage300.Common.Resources;  using ValuedPartner.TU.Models.Enums; // For common enumerations  using ValuedPartner.TU.Resources; // For common resources  using ValuedPartner.TU.Resources.Forms;  #endregion  namespace ValuedPartner.TU.Models  {  /// <summary>  /// Partial class for PaymentCodes  /// </summary>  public partial class PaymentCodes : ModelBase  {  /// <summary>  /// Gets or sets PaymentCode  /// </summary>  [Key]  [Required(ErrorMessageResourceName = "Required", ErrorMessageResourceType = typeof(AnnotationsResx))]  [StringLength(12, ErrorMessageResourceName = "MaxLength",ErrorMessageResourceType = typeof(AnnotationsResx))]  [Display(Name = "PaymentCode", ResourceType = typeof (PaymentCodesResx))]  [RegularExpression(@"^[a-zA-Z0-9]+$", ErrorMessageResourceName = "AlphaNumeric", ErrorMessageResourceType = typeof(AnnotationsResx))]  public string PaymentCode { get; set; }  /// <summary>  /// Gets or sets Description  /// </summary>  [StringLength(60, ErrorMessageResourceName = "MaxLength",ErrorMessageResourceType = typeof(AnnotationsResx))]  [Display(Name = "Description", ResourceType = typeof (PaymentCodesResx))]  public string Description { get; set; }  /// <summary>  /// Gets or sets Status  /// </summary>  [Display(Name = "Status", ResourceType = typeof (PaymentCodesResx))]  public ValuedPartner.TU.Models.Enums.Status Status { get; set; }  /// <summary>  /// Gets or sets InactiveDate  /// </summary>  [ValidateDateFormat(ErrorMessageResourceName="DateFormat", ErrorMessageResourceType = typeof(AnnotationsResx))]  [Display(Name = "InactiveDate", ResourceType = typeof (PaymentCodesResx))]  public DateTime InactiveDate { get; set; }  /// <summary>  /// Gets or sets DateLastMaintained  /// </summary>  [ValidateDateFormat(ErrorMessageResourceName="DateFormat", ErrorMessageResourceType = typeof(AnnotationsResx))]  [Display(Name = "DateLastMaintained", ResourceType = typeof (PaymentCodesResx))]  public DateTime DateLastMaintained { get; set; }  /// <summary>  /// Gets or sets PaymentType  /// </summary>  [Display(Name = "PaymentType", ResourceType = typeof (PaymentCodesResx))]  public ValuedPartner.TU.Models.Enums.PaymentType PaymentType { get; set; }  #region UI Strings  /// <summary>  /// Gets Status string value  /// </summary>  public string StatusString  {  get { return EnumUtility.GetStringValue(Status); }  }  /// <summary>  /// Gets PaymentType string value  /// </summary>  public string PaymentTypeString  {  get { return EnumUtility.GetStringValue(PaymentType); }  }  #endregion  }  } |

* 1. Payment Codes View Model

Access the **PaymentCodesViewModel.cs** class and add the **FormattedInactiveDate** property and modify the **PaymentTypes** property as this will now be set by the controller, since it will have a filter applied (the code to be added is highlighted in yellow).

|  |
| --- |
| // Copyright (c) 2020 Valued Partner All rights reserved.  #region Namespace  using System;  using System.Collections;  using System.Collections.Generic;  using Sage.CA.SBS.ERP.Sage300.Common.Web;  using Sage.CA.SBS.ERP.Sage300.Common.Models;  using Sage.CA.SBS.ERP.Sage300.Common.Utilities;  using ValuedPartner.TU.Models;  using ValuedPartner.TU.Models.Enums;  #endregion  namespace ValuedPartner.TU.Web.Areas.TU.Models  {  /// <summary>  /// Class for PaymentCodesViewModel  /// </summary>  /// <typeparam name="T">Where T is type of <see cref="PaymentCodes"/></typeparam>  public class PaymentCodesViewModel<T> : ViewModelBase<T>  where T : PaymentCodes, new()  {  /// <summary>  /// Status list  /// </summary>  public IEnumerable<CustomSelectList> Status  {  get { return EnumUtility.GetItemsList<ValuedPartner.TU.Models.Enums.Status>(); }  }  /// <summary>  /// PaymentType list  /// </summary>  public IEnumerable<CustomSelectList> PaymentTypes  {  get { return EnumUtility.GetItemsList<ValuedPartner.TU.Models.Enums.PaymentType>(); }  set { }  }  /// <summary>  /// Gets FormattedInactiveDate  /// </summary>  public string FormattedInactiveDate  {  get  {  return DateUtil.GetShortDate((Data != null &&  Data.Status == ValuedPartner.TU.Models.Enums.Status.Inactive ?  Data.InactiveDate :  DateUtil.GetNowDate()), string.Empty);  }  }  }  } |

* 1. Payment Codes Internal Controller

Internal methods are only used by the public controller and the complexities are placed here as opposed to the public controller.

Access the **PaymentCodesControllerInternal.cs** class and add the private constants **PaymentAndProcessingModuleId** and **PaymentProcessing** (the code to be added is highlighted in yellow).

Add the **UpdateInactiveStatus** routine to the class (the code to be added is highlighted in yellow).

Modify the **GetViewInfo** routine and add the **FilteredPaymentTypes** routine to the class (the code to be added is highlighted in yellow).

|  |
| --- |
| // Copyright (c) 2020 Valued Partner All rights reserved.  #region Namespace  using System;  using System.Linq;  using System.Linq.Expressions;  using System.Collections.Generic;  using Sage.CA.SBS.ERP.Sage300.Common.Models;  using Sage.CA.SBS.ERP.Sage300.Common.Resources;  using Sage.CA.SBS.ERP.Sage300.Common.Web;  using Sage.CA.SBS.ERP.Sage300.Common.Web.Controllers.ExportImport;  using ValuedPartner.TU.Interfaces.Services;  using ValuedPartner.TU.Models;  using ValuedPartner.TU.Models.Enums;  using ValuedPartner.TU.Resources.Forms;  using ValuedPartner.TU.Web.Areas.TU.Models;  #endregion  namespace ValuedPartner.TU.Web.Areas.TU.Controllers  {  /// <summary>  /// PaymentCodes Internal Controller  /// </summary>  /// <typeparam name="T">Where T is type of <see cref="PaymentCodes"/></typeparam>  public class PaymentCodesControllerInternal<T> : BaseExportImportControllerInternal<T, IPaymentCodesService<T>>  where T : PaymentCodes, new()  {  #region Private variables  private const string PaymentAndProcessingModuleId = "YP";  private const int PaymentProcessing = 5;  #endregion  #region Constructor  /// <summary>  /// New instance of <see cref="PaymentCodesControllerInternal{T}"/>  /// </summary>  /// <param name="context">Context</param>  public PaymentCodesControllerInternal(Context context)  : base(context)  {  }  #endregion  #region Internal methods  /// <summary>  /// Create a PaymentCodes  /// </summary>  /// <returns>JSON object for PaymentCodes</returns>  internal PaymentCodesViewModel<T> Create()  {  var viewModel = GetViewModel(new T(), null);  viewModel.UserAccess = GetAccessRights();  return viewModel;  }  /// <summary>  /// Get a PaymentCodes  /// </summary>  /// <param name="id">Id for PaymentCodes</param>  /// <returns>JSON object for PaymentCodes</returns>  internal PaymentCodesViewModel<T> GetById(string id)  {  var data = Service.GetById(id);  var userMessage = new UserMessage(data);  return GetViewModel(data, userMessage);  }  /// <summary>  /// Add a PaymentCodes  /// </summary>  /// <param name="model">Model for PaymentCodes</param>  /// <returns>JSON object for PaymentCodes</returns>  internal PaymentCodesViewModel<T> Add(T model)  {  var data = Service.Add(model);  var userMessage = new UserMessage(data,  string.Format(CommonResx.AddSuccessMessage, PaymentCodesResx.PaymentCode, data.PaymentCode));  return GetViewModel(data, userMessage);  }  /// <summary>  /// Update a PaymentCodes  /// </summary>  /// <param name="model">Model for PaymentCodes</param>  /// <returns>JSON object for PaymentCodes</returns>  internal PaymentCodesViewModel<T> Save(T model)  {  var data = Service.Save(model);  var userMessage = new UserMessage(data, CommonResx.SaveSuccessMessage);  return GetViewModel(data, userMessage);  }  /// <summary>  /// Delete a PaymentCodes  /// </summary>  /// <param name="id">Id for PaymentCodes</param>  /// <returns>JSON object for PaymentCodes</returns>  internal PaymentCodesViewModel<T> Delete(string id)  {  Expression<Func<T, bool>> filter = param => param.PaymentCode == id;  var data = Service.Delete(filter);  var userMessage = new UserMessage(data,  string.Format(CommonResx.DeleteSuccessMessage, PaymentCodesResx.PaymentCode, data.PaymentCode));  return GetViewModel(data, userMessage);  }  /// <summary>  /// Update status  /// </summary>  /// <param name="model">Model for PaymentCodes</param>  /// <returns>Model for PaymentCodes</returns>  internal PaymentCodesViewModel<T> UpdateInactiveStatus(T model)  {  var data = Service.UpdateInactiveStatus(model);  var userMessage = new UserMessage(data, CommonResx.SaveSuccessMessage);  return GetViewModel(data, userMessage);  }  #endregion  #region Private methods  /// <summary>  /// Generic routine to return a view model for PaymentCodes  /// </summary>  /// <param name="model">Model for PaymentCodes</param>  /// <param name="userMessage">User Message for PaymentCodes</param>  /// <returns>View Model for PaymentCodes</returns>  private PaymentCodesViewModel<T> GetViewModel(T model, UserMessage userMessage)  {  return new PaymentCodesViewModel<T>  {  Data = model,  UserMessage = userMessage,  PaymentTypes = FilteredPaymentTypes()  };  }  /// <summary>  /// Filter Payment Types  /// </summary>  /// <returns>Filtered List</returns>  private IEnumerable<CustomSelectList> FilteredPaymentTypes()  {  // Locals  var isModuleActive = Service.IsModuleActive(PaymentAndProcessingModuleId);  var paymentTypes = EnumUtility.GetItemsList<PaymentType>();  // Filter list if module is not active  if (paymentTypes != null && !isModuleActive)  {  paymentTypes = from selectList in paymentTypes  where Convert.ToInt16(selectList.Value) != PaymentProcessing  select selectList;  }  return paymentTypes;  }  #endregion  }  } |

* 1. Payment Codes Controller

Public methods are invoked by the MVC framework and from JavaScript. The complexities of the public methods are hidden in the internal controller’s methods.

Access the **PaymentCodesController.cs** class and add the **UpdateInactiveStatus** routine to the class (the code to be added is highlighted in yellow).

|  |
| --- |
| // Copyright (c) 2020 Valued Partner All rights reserved.  #region Namespace  using Microsoft.Practices.Unity;  using System.Web.Mvc;  using Sage.CA.SBS.ERP.Sage300.Common.Exceptions;  using Sage.CA.SBS.ERP.Sage300.Common.Models;  using Sage.CA.SBS.ERP.Sage300.Common.Models.Enums;  using Sage.CA.SBS.ERP.Sage300.Common.Resources;  using Sage.CA.SBS.ERP.Sage300.Common.Web;  using ValuedPartner.TU.Models;  using ValuedPartner.TU.Models.Enums;  using ValuedPartner.TU.Resources.Forms;  using ValuedPartner.TU.Web.Areas.TU.Models;  #endregion  namespace ValuedPartner.TU.Web.Areas.TU.Controllers  {  /// <summary>  /// PaymentCodes Public Controller  /// </summary>  /// <typeparam name="T">Where T is type of <see cref="PaymentCodes"/></typeparam>  public class PaymentCodesController<T> : MultitenantControllerBase<PaymentCodesViewModel<T>>  where T : PaymentCodes, new()  {  #region Public variables  /// <summary>  /// Gets or sets the internal controller  /// </summary>  public PaymentCodesControllerInternal<T> ControllerInternal { get; set; }  #endregion  #region Constructor  /// <summary>  /// Constructor for PaymentCodes  /// </summary>  /// <param name="container">Unity Container</param>  public PaymentCodesController(IUnityContainer container)  : base(container,"TUPaymentCodes")  {  }  #endregion  #region Initialize MultitenantControllerBase  /// <summary>  /// Override Initialize method  /// </summary>  /// <param name="requestContext">Request Context</param>  protected override void Initialize(System.Web.Routing.RequestContext requestContext)  {  base.Initialize(requestContext);  ControllerInternal = new PaymentCodesControllerInternal<T>(Context);  }  #endregion  #region Public methods  /// <summary>  /// Load screen  /// </summary>  /// <param name="id">Id for PaymentCodes</param>  /// <returns>JSON object for PaymentCodes</returns>  public virtual ActionResult Index(string id)  {  PaymentCodesViewModel<T> viewModel;  try  {  viewModel = !string.IsNullOrEmpty(id) ? ControllerInternal.GetById(id) : ControllerInternal.Create();  }  catch (BusinessException businessException)  {  return  JsonNet(BuildErrorModelBase(CommonResx.GetFailedMessage, businessException,  PaymentCodesResx.PaymentCode));  }  return View(viewModel);  }  /// <summary>  /// Get PaymentCodes  /// </summary>  /// <param name="id">Id for PaymentCodes</param>  /// <returns>JSON object for PaymentCodes</returns>  [HttpPost]  public virtual JsonNetResult Get(string id)  {  try  {  if (!string.IsNullOrEmpty(id))  {  return JsonNet(ControllerInternal.GetById(id));  }  }  catch (BusinessException businessException)  {  return  JsonNet(BuildErrorModelBase(CommonResx.GetFailedMessage, businessException,  PaymentCodesResx.PaymentCode));  }  return JsonNet(new PaymentCodesViewModel<T>());  }  /// <summary>  /// Add PaymentCodes  /// </summary>  /// <param name="model">Model for PaymentCodes</param>  /// <returns>JSON object for PaymentCodes</returns>  [HttpPost]  public virtual JsonNetResult Add(T model)  {  try  {  ViewModelBase<ModelBase> viewModel;  return ValidateModelState(ModelState, out viewModel)  ? JsonNet(ControllerInternal.Add(model))  : JsonNet(viewModel);  }  catch (BusinessException businessException)  {  return  JsonNet(BuildErrorModelBase(CommonResx.AddFailedMessage, businessException,  PaymentCodesResx.PaymentCode));  }  }  /// <summary>  /// Create PaymentCodes  /// </summary>  /// <returns>JSON object for PaymentCodes</returns>  [HttpPost]  public virtual JsonNetResult Create()  {  return JsonNet(ControllerInternal.Create());  }  /// <summary>  /// Update PaymentCodes  /// </summary>  /// <param name="model">Model for PaymentCodes</param>  /// <returns>JSON object for PaymentCodes</returns>  [HttpPost]  public virtual JsonNetResult Save(T model)  {  try  {  ViewModelBase<ModelBase> viewModel;  return ValidateModelState(ModelState, out viewModel)  ? JsonNet(ControllerInternal.Save(model))  : JsonNet(viewModel);  }  catch (BusinessException businessException)  {  return JsonNet(BuildErrorModelBase(CommonResx.SaveFailedMessage, businessException));  }  }  /// <summary>  /// Delete PaymentCodes  /// </summary>  /// <param name="id">Id for PaymentCodes</param>  /// <returns>JSON object for PaymentCodes</returns>  [HttpPost]  public virtual JsonNetResult Delete(string id)  {  try  {  return JsonNet(ControllerInternal.Delete(id));  }  catch (BusinessException businessException)  {  return  JsonNet(BuildErrorModelBase(CommonResx.DeleteFailedMessage, businessException,  PaymentCodesResx.PaymentCode));  }  }  /// <summary>  /// Update status  /// </summary>  /// <param name="model">Model for PaymentCodes</param>  /// <returns>Model for PaymentCodes</returns>  [HttpPost]  public virtual JsonNetResult UpdateInactiveStatus(T model)  {  try  {  ViewModelBase<ModelBase> viewModel;  return ValidateModelState(ModelState, out viewModel)  ? JsonNet(ControllerInternal.UpdateInactiveStatus(model))  : JsonNet(viewModel);  }  catch (BusinessException businessException)  {  return JsonNet(BuildErrorModelBase(CommonResx.SaveFailedMessage, businessException));  }  }  #endregion  }  } |

* 1. Payment Codes Partial Razor View

Access the \_**PaymentCodes.cshtml** file and add the remaining controls to the screen (the code to be added is highlighted in green).

|  |
| --- |
| @\* Copyright (c) 2020 Valued Partner All rights reserved. \*@  @model ValuedPartner.TU.Web.Areas.TU.Models.PaymentCodesViewModel<ValuedPartner.TU.Models.PaymentCodes>  @using PaymentCodesResx = ValuedPartner.TU.Resources.Forms.PaymentCodesResx  @using Sage.CA.SBS.ERP.Sage300.Common.Web.AreaConstants  @using Sage.CA.SBS.ERP.Sage300.Common.Resources  @using Sage.CA.SBS.ERP.Sage300.Common.Web.HtmlHelperExtension  @using Sage.CA.SBS.ERP.Sage300.Common.Models.Enums  @using AnnotationsResx = Sage.CA.SBS.ERP.Sage300.Common.Resources.AnnotationsResx  <script type="text/javascript">  @Html.ConvertToJsVariableUsingNewtonSoft("PaymentCodesViewModel", Model)  </script>  @Html.Partial("~/Areas/TU/Views/PaymentCodes/Partials/\_Localization.cshtml")  <div id="success" class="message"></div>  <div class="form-screen">  <div id="message"></div>  <header>  <section class="header-group-1">  <div class="header-wrapper">  <div class="header-headline">  @Html.SageHeader1Label("PaymentCodesHeader", PaymentCodesResx.Entity)  </div>  @Html.Partial(Core.OptionsMenu, Model.UserAccess, new ViewDataDictionary { { OptionsMenu.UseLessCss, true } })  <div class="header-options">  @if (Model.UserAccess.SecurityType.HasFlag(SecurityType.Modify))  {  @Html.KoSageButton("btnNew", null, new { @value = CommonResx.CreateNew, @id = "btnNew", @class = "btn btn-primary" })  }  </div>  </div>  <div class="flag-required">  <span class="req-option">@CommonResx.RequiredLegend</span>  </div>  </section>  </header>  <div class="form-group">  <div class="search-group">  @Html.SageLabelFor(model => model.Data.PaymentCode, new { @id = "lblPaymentCode", @class = "required" })  @Html.KoSageTextBoxFor(model => model.Data.PaymentCode, new { @sagevalue = "Data.PaymentCode", @valueUpdate = "'input'" }, new { @id = "txtPaymentCode", @class = "default", @formatTextbox = "alphaNumeric" })  @Html.KoSageButton("btnLoadPaymentCode", null, new { @id = "btnLoad", @class = "icon btn-go", @tabindex = "-1" })  @Html.KoSageButton("btnFinderPaymentCode", null, new { @class = "icon btn-search", @id = "btnFinderPaymentCode", @tabindex = "-1" })  @Html.ValidationMessageFor(model => model.Data.PaymentCode)  </div>  <div class="input-group">  @\*@Html.ValidationMessageFor(m => m.Data.PaymentCode)\*@  @Html.SageLabelFor(model => model.Data.Description)  @Html.KoSageTextBoxFor(model => model.Data.Description, new { @value = "Data.Description", @valueUpdate = "'input'" }, new { @id = "tbDescription", @class = "large" })  @Html.ValidationMessageFor(model => model.Data.Description, null, new { @class = "" })  </div>  </div>  <div class="form-group">  <div class="dropdown-group">  @Html.SageLabelFor(model => model.Data.PaymentType)  @Html.KoSageDropDownList("ddlPaymentType", new { @options = "PaymentTypes", @value = "Data.PaymentType", @optionsText = "'Text'", @optionsValue = "'Value'" }, new { @id = "ddlPaymentType", @class = "default" })  </div>  <div class="input-group">  @Html.SageLabelFor(model => model.Data.DateLastMaintained)  @Html.KoSageTextBox("txLastDateMaintained", new { @value = "Data.ComputedLastMaintainedDate" }, new { @disabled = "true", @class = "default" })  </div>  <div class="input-group with-checkbox">  <div class="child">  @Html.KoSageCheckBox("chkStatus", false, new { @sagechecked = "Data.Inactive" }, new { @id = "chkStatus" })  @Html.SageLabel(CommonResx.InactiveAsOfDate, null, new { @for = "chkStatus", @class = "" })  </div>  @Html.KoSageTextBox("txInactiveDate", new { @value = "Data.ComputedInactiveDate" }, new { @disabled = true, @class = "default" })  </div>  </div>  </div>  <section class="footer-group-1">  @if (Model.UserAccess.SecurityType.HasFlag(SecurityType.Modify))  {  @Html.KoSageButton("btnSave", new { }, new { @value = CommonResx.Save, @id = "btnSave", @class = "btn btn-primary" })  @Html.KoSageButton("btnDelete", new { }, new { @value = CommonResx.Delete, @id = "btnDelete", @class = "btn btn-primary" })  }  </section>  </div> |

* 1. Payment Codes Repository JavaScript

Access the **…PaymentCodesRepository.js** file and add the **updateInactiveStatus** method (the code to be added is highlighted in yellow).

|  |
| --- |
| /\* Copyright (c) 2020 Valued Partner All rights reserved. \*/  // Add the following commented line to enable TypeScript static type checking  // Remove this line if not needed  //@ts-check  "use strict";  // Ajax call to controller  var paymentCodesAjax = {  call: function (method, data, callbackMethod) {  var url = sg.utls.url.buildUrl("TU", "PaymentCodes", method);  sg.utls.ajaxPost(url, data, callbackMethod);  }  };  var paymentCodesRepository = {  /\*\*  \* Get  \*  \* @method get  \* @param id  \* @param callbackMethod  \*/  get: function(id, callbackMethod) {  var data = { 'id': id };  paymentCodesAjax.call("Get", data, callbackMethod);  },  /\*\*  \* Create  \*  \* @method create  \* @param callbackMethod  \*/  create: function(callbackMethod) {  var data = {};  paymentCodesAjax.call("Create", data, callbackMethod);  },  /\*\*  \* Delete  \*  \* @method delete  \* @param id  \* @param callbackMethod  \*/  delete: function(id, callbackMethod) {  var data = { 'id': id };  paymentCodesAjax.call("Delete", data, callbackMethod);  },  /\*\*  \* Add  \*  \* @method add  \* @param data  \* @param callbackMethod  \*/  add: function(data, callbackMethod) {  paymentCodesAjax.call("Add", data, callbackMethod);  },  /\*\*  \* Update  \*  \* @method update  \* @param data  \* @param callbackMethod  \*/  update: function(data, callbackMethod) {  paymentCodesAjax.call("Save", data, callbackMethod);  },  /\*\*  \* Post  \*  \* @method update  \* @param data  \* @param callbackMethod  \*/  post: function(callbackMethod) {  paymentCodesAjax.call("Post", null, callbackMethod);  },  /\*\*  \* updateInactiveStatus  \*  \* @method updateInactiveStatus  \* @param data  \* @param callbackMethod  \*/  updateInactiveStatus: function (data, callbackMethod) {  paymentCodesAjax.call("UpdateInactiveStatus", data, callbackMethod);  }  // Additional methods go here  }; |

* 1. Payment Codes Knockout Binding JavaScript

Access the …**PaymentCodesKoExtn.js** file and add the new computed properties for **Status**, **InactiveDate**, and **DateLastMaintained** (the code to be added is highlighted in yellow).

|  |
| --- |
| /\* Copyright (c) 2020 Valued Partner All rights reserved. \*/  // Add the following commented line to enable TypeScript static type checking  // Remove this line if not needed  //@ts-check  "use strict";  function paymentCodesObservableExtension(viewModel, uiMode) {  var model = viewModel.Data;  model.UIMode = ko.observable(uiMode);  // Computed Fields go here  model.Inactive = ko.computed({  read: function () {  return (model.Status() === paymentCodesUI.status.InActive);  },  write: function (value) {  if (value) {  model.Status(paymentCodesUI.status.InActive);  } else {  model.Status(paymentCodesUI.status.Active);  }  }  });  model.ComputedInactiveDate = ko.computed(function () {  if (model.Status() === paymentCodesUI.status.Active) {  return null;  } else {  return sg.utls.kndoUI.getFormattedDate(model.InactiveDate()) ?  sg.utls.kndoUI.getFormattedDate(model.InactiveDate()) :  viewModel.FormattedInactiveDate();  }  });  model.ComputedLastMaintainedDate = ko.computed(function () {  return sg.utls.kndoUI.getFormattedDate(model.DateLastMaintained());  });  }; |

* 1. Payment Codes Behaviour JavaScript

Access the **…PaymentCodesBehaviour.js** file and add the business logic for the new screen controls (the code to be added is highlighted in yellow).

|  |
| --- |
| /\* Copyright (c) 2020 Valued Partner All rights reserved. \*/  // Add the following commented line to enable TypeScript static type checking  // Remove this line if not needed  //@ts-check  "use strict";  var modelData;  var paymentCodesUI = paymentCodesUI || {};  paymentCodesUI = {  paymentCodesModel: {},  status: { InActive: 0, Active: 1 },  ignoreIsDirtyProperties: ["PaymentCode"],  computedProperties: ["UIMode", "Inactive", "ComputedInactiveDate", "ComputedLastMaintainedDate"],  hasKoBindingApplied: false,  isKendoControlNotInitialised: false,  paymentCode: null,  checkStatus: true,  /\*\*  \* Initialization  \*  \* @method init  \*/  init: function () {  paymentCodesUI.initButtons();  paymentCodesUI.initFinders();  paymentCodesUISuccess.initialLoad(PaymentCodesViewModel);  paymentCodesUISuccess.setkey();  },  /\*\*  \* Save  \*  \* @method savePaymentCodes  \*/  savePaymentCodes: function () {  if ($("#frmPaymentCodes").valid()) {  var data = sg.utls.ko.toJS(modelData, paymentCodesUI.computedProperties);  if (modelData.UIMode() === sg.utls.OperationMode.SAVE) {  paymentCodesRepository.update(data, paymentCodesUISuccess.update);  } else {  paymentCodesRepository.add(data, paymentCodesUISuccess.update);  }  }  },  /\*\*  \* Initialize the Buttons  \*  \* @method initButtons  \*/  initButtons: function () {  // Import/Export Buttons  sg.exportHelper.setExportEvent("btnOptionExport", "tupaymentcodes", false, $.noop);  sg.importHelper.setImportEvent("btnOptionImport", "tupaymentcodes", false, $.noop);  // Key field change event  $("#txtPaymentCode").on('blur', function (e) {  modelData.PaymentCode($("#txtPaymentCode").val());  if (sg.controls.GetString(modelData.PaymentCode()) !== "") {  paymentCodesUI.checkIsDirty(paymentCodesUI.get, paymentCodesUI.paymentCode);  }  });  // Create New Button  $("#btnNew").on('click', function () {  paymentCodesUI.checkIsDirty(paymentCodesUI.create, paymentCodesUI.paymentCode);  });  // Save Button  $("#btnSave").on('click', function () {  sg.utls.SyncExecute(paymentCodesUI.savePaymentCodes);  });  // Delete Button  $("#btnDelete").on('click', function () {  if ($("#frmPaymentCodes").valid()) {  var message = jQuery.validator.format(paymentCodesResources.DeleteConfirmMessage, paymentCodesResources.PaymentCodeTitle, modelData.PaymentCode());  sg.utls.showKendoConfirmationDialog(function () {  sg.utls.clearValidations("frmPaymentCodes");  paymentCodesRepository.delete(modelData.PaymentCode(), paymentCodesUISuccess.delete);  }, null, message, paymentCodesResources.DeleteTitle);  }  });  },  // Init Dropdowns here  initDropDownList: function () {  $("#ddlPaymentType").kendoDropDownList();  },  /\*\*  \* Initialize the Finders, if any  \*  \* @method initFinders  \*/  initFinders: function () {  sg.viewFinderHelper.setViewFinder("btnFinderPaymentCode", "txtPaymentCode",/\*TODO Modify the finder property\*/ {viewID:"AR0012", viewOrder:0, displayFieldNames:["PAYMCODE", "TEXTDESC", "ACTVSW", "INACTDATE"], returnFieldNames:["PAYMCODE"],filter:null, initKeyValues:[], parentValAsInitKey:true});  },  /\*\*  \* Get  \*  \* @method get  \*/  get: function () {  paymentCodesRepository.get(modelData.PaymentCode(), paymentCodesUISuccess.get);  },  /\*\*  \* Create  \*  \* @method create  \*/  create: function () {  sg.utls.clearValidations("frmPaymentCodes");  paymentCodesRepository.create(paymentCodesUISuccess.create);  },  /\*\*  \* Is Dirty check  \*  \* @method checkIsDirty  \* @param functionToCall  \* @param paymentCode  \*/  checkIsDirty: function (functionToCall, paymentCode) {  if (paymentCodesUI.paymentCodesModel.isModelDirty.isDirty() && paymentCode != null && paymentCode != "") {  sg.utls.showKendoConfirmationDialog(  function () { // Yes  sg.utls.clearValidations("frmPaymentCodes");  functionToCall.call();  },  function () { // No  if (sg.controls.GetString(paymentCode) != sg.controls.GetString(modelData.PaymentCode())) {  modelData.PaymentCode(paymentCode);  }  return;  },  jQuery.validator.format(globalResource.SaveConfirm, paymentCodesResources.PaymentCodeTitle, paymentCode));  } else {  functionToCall.call();  }  },  statusChange: function (value) {  if (value && sg.controls.GetString(modelData.PaymentCode() != "")) {  if ($("#frmPaymentCodes").valid()  && modelData.UIMode() === sg.utls.OperationMode.SAVE) {  if (paymentCodesUI.checkStatus) {  var data = sg.utls.ko.toJS(modelData, paymentCodesUI.computedProperties);  paymentCodesRepository.updateInactiveStatus(data, paymentCodesUISuccess.updateStatus);  }  paymentCodesUI.checkStatus = true;  }  }  }  };  // Callbacks  var paymentCodesUISuccess = {  /\*\*  \* Setkey  \*  \* @method setkey  \*/  setkey: function () {  paymentCodesUI.paymentCode = modelData.PaymentCode();  },  /\*\*  \* Get  \*  \* @method get  \* @param jsonResult  \*/  get: function (jsonResult) {  if (jsonResult.UserMessage && jsonResult.UserMessage.IsSuccess) {  if (jsonResult.Data != null) {  paymentCodesUI.checkStatus = (jsonResult.Data.Status === paymentCodesUI.status.Active);  paymentCodesUISuccess.displayResult(jsonResult, sg.utls.OperationMode.SAVE);  } else {  modelData.UIMode(sg.utls.OperationMode.NEW);  }  paymentCodesUISuccess.setkey();  }  },  /\*\*  \* Update  \*  \* @method update  \* @param jsonResult  \*/  update: function (jsonResult) {  if (jsonResult.UserMessage.IsSuccess) {  paymentCodesUISuccess.displayResult(jsonResult, sg.utls.OperationMode.SAVE);  paymentCodesUISuccess.setkey();  }  sg.utls.showMessage(jsonResult);  },  /\*\*  \* updateStatus  \*  \* @method updateStatus  \* @param jsonResult  \*/  updateStatus: function (jsonResult) {  if (!jsonResult.UserMessage.IsSuccess) {  modelData.Status(paymentCodesUI.status.Active);  sg.utls.showMessage(jsonResult);  }  },  /\*\*  \* Create  \*  \* @method create  \* @param jsonResult  \*/  create: function (jsonResult) {  paymentCodesUISuccess.displayResult(jsonResult, sg.utls.OperationMode.NEW);  paymentCodesUI.paymentCodesModel.isModelDirty.reset();  paymentCodesUISuccess.setkey();  sg.controls.Focus($("#txtPaymentCode"));  },  /\*\*  \* Delete  \*  \* @method delete  \* @param jsonResult  \*/  delete: function (jsonResult) {  if (jsonResult.UserMessage.IsSuccess) {  paymentCodesUISuccess.displayResult(jsonResult, sg.utls.OperationMode.NEW);  paymentCodesUI.paymentCodesModel.isModelDirty.reset();  paymentCodesUISuccess.setkey();  }  sg.utls.showMessage(jsonResult);  },  /\*\*  \* Display Result  \*  \* @method displayResult  \* @param jsonResult  \* @param uiMode  \*/  displayResult: function (jsonResult, uiMode) {  if (jsonResult != null) {  if (!paymentCodesUI.hasKoBindingApplied) {  paymentCodesUI.paymentCodesModel = ko.mapping.fromJS(jsonResult);  paymentCodesUI.hasKoBindingApplied = true;  modelData = paymentCodesUI.paymentCodesModel.Data;  paymentCodesObservableExtension(paymentCodesUI.paymentCodesModel, uiMode);  modelData.Inactive.subscribe(paymentCodesUI.statusChange);  paymentCodesUI.paymentCodesModel.isModelDirty = new ko.dirtyFlag(modelData, paymentCodesUI.ignoreIsDirtyProperties);  ko.applyBindings(paymentCodesUI.paymentCodesModel);  } else {  ko.mapping.fromJS(jsonResult, paymentCodesUI.paymentCodesModel);  modelData.UIMode(uiMode);  if (uiMode != sg.utls.OperationMode.NEW) {  paymentCodesUI.paymentCodesModel.isModelDirty.reset();  }  }  if (!paymentCodesUI.isKendoControlNotInitialised) {  paymentCodesUI.isKendoControlNotInitialised = true;  paymentCodesUI.initDropDownList();  $("#ddlPaymentType").data("kendoDropDownList").value(modelData.PaymentType(jsonResult.Data.PaymentType));  } else {  $("#ddlPaymentType").data("kendoDropDownList").value(modelData.PaymentType());  }  }  },  /\*\*  \* Initial Load  \*  \* @method initialLoad  \* @param result  \*/  initialLoad: function (result) {  if (result) {  paymentCodesUISuccess.displayResult(result, sg.utls.OperationMode.NEW);  } else {  sg.utls.showMessageInfo(sg.utls.msgType.ERROR, paymentCodesResources.ProcessFailedMessage);  }  sg.controls.Focus($("#txtPaymentCode"));  },  /\*\*  \* Is New  \*  \* @method isNew  \* @param model  \*/  isNew: function (model) {  if (model.PaymentCode() === null) {  return true;  }  return false;  }  };  // Initial Entry  $(function () {  paymentCodesUI.init();  $(window).on('beforeunload', function () {  if (globalResource.AllowPageUnloadEvent && paymentCodesUI.paymentCodesModel.isModelDirty.isDirty()) {  return jQuery('<div />').html(jQuery.validator.format(globalResource.SaveConfirm2, paymentCodesResources.PaymentCodeTitle)).text();  }  });  }); |

1. Review the Screen

At this point, the screen should be complete and functioning as expected.

