Sage 300 Web Screens SDK

Creating a Setup Web Screen

May 2020

The MIT License (MIT)

Copyright © 2020 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](#_Toc40954261)

[2. Sage 300 Desktop Screens 6](#_Toc40954262)

[2.1 Process Overview 7](#_Toc40954263)

[2.2 Links and References 7](#_Toc40954264)

[3. Coding Standards 9](#_Toc40954265)

[4. Getting Started 10](#_Toc40954266)

[4.1 A/R Payment Codes Screen 10](#_Toc40954267)

[4.1.1 A/R Payment Codes – Sage 300 Desktop 10](#_Toc40954268)

[4.1.2 A/R Payment Codes - Sage 300 Web Screen 11](#_Toc40954269)

[5. Code Generation Wizard 12](#_Toc40954270)

[5.1 Reference 13](#_Toc40954271)

[5.2 Wizard Inputs 14](#_Toc40954272)

[5.3 Wizard Completion 17](#_Toc40954273)

[5.4 Wizard Outputs 18](#_Toc40954274)

[5.4.1 Evaluate Generated Code 21](#_Toc40954275)

[6. Portal Menu Help 25](#_Toc40954276)

[6.1 Adding to the Portal Menu Help 25](#_Toc40954277)

[7. Complete the Screen 26](#_Toc40954278)

[7.1 Payment Codes Business Entity Interface 26](#_Toc40954279)

[7.2 Payment Codes Service Interface 26](#_Toc40954280)

[7.3 Payment Codes Service 26](#_Toc40954281)

[7.4 Payment Codes Repository 26](#_Toc40954282)

[7.5 Payment Codes Model 26](#_Toc40954283)

[7.6 Payment Codes View Model 27](#_Toc40954284)

[7.7 Payment Codes Internal Controller 27](#_Toc40954285)

[7.8 Payment Codes Controller 27](#_Toc40954286)

[7.9 Payment Codes Partial Razor View 27](#_Toc40954287)

[7.10 Payment Codes Repository JavaScript 27](#_Toc40954288)

[7.11 Payment Codes Knockout Extension JavaScript 27](#_Toc40954289)

[7.12 Payment Codes Behaviour JavaScript 28](#_Toc40954290)

[8. Review the Screen 29](#_Toc40954291)

[9. Appendix: Complete Code Files 30](#_Toc40954292)

[9.1 Payment Codes Business Entity Interface 30](#_Toc40954293)

[9.1.1 IPaymentCodesEntity.cs 30](#_Toc40954294)

[9.2 Payment Codes Service Interface 31](#_Toc40954295)

[9.2.1 IPaymentCodesService.cs 31](#_Toc40954296)

[9.3 Payment Codes Service 32](#_Toc40954297)

[9.3.1 PaymentCodesEntityService.cs 32](#_Toc40954298)

[9.4 Payment Codes Repository 34](#_Toc40954299)

[9.4.1 PaymentCodesRepository.cs 34](#_Toc40954300)

[9.5 Payment Codes Model 38](#_Toc40954301)

[9.5.1 PaymentCodes.cs 38](#_Toc40954302)

[9.6 Payment Codes View Model 40](#_Toc40954303)

[9.6.1 PaymentCodesViewModel.cs 40](#_Toc40954304)

[9.7 Payment Codes Internal Controller 41](#_Toc40954305)

[9.7.1 PaymentCodesControllerInternal.cs 41](#_Toc40954306)

[9.8 Payment Codes Controller 44](#_Toc40954307)

[9.8.1 PaymentCodesController.cs 44](#_Toc40954308)

[9.9 Payment Codes Partial Razor View 48](#_Toc40954309)

[9.9.1 \_PaymentCode.cshtml 48](#_Toc40954310)

[9.10 Payment Codes Repository JavaScript 50](#_Toc40954311)

[9.10.1 ...PaymentCodesRepository.js 50](#_Toc40954312)

[9.11 Payment Codes Knockout Binding JavaScript 52](#_Toc40954313)

[9.11.1 …PaymentCodesKoExtn.js 52](#_Toc40954314)

[9.12 Payment Codes Behaviour JavaScript 53](#_Toc40954315)

[9.12.1 …PaymentCodesBahviour.js 53](#_Toc40954316)

1. Overview

This document is intended to serve as a guide for illustrating how to create a basic setup screen for Sage 300.

Links and references to other documents and wiki articles are presented instead of duplicating their content.

At present, there are several types or categories of screens that are developed for Sage 300:

* Setup screens (typically stateless)
* Options screens
* Header-Detail screens (typically stateful)
* Reports
* Process screens
* Inquiry screens
* Dynamic Query screens

The initial versions of Sage 300 (R1, R2 and R3) are a Minimum Viable Product (MVP) and therefore, not all functionality found in the Sage 300 Desktop is brought forth into the Sage 300 Web UI’s at this time.

1. Sage 300 Desktop Screens

The first, and most important, step in the process of creating a Sage 300 web screen is to become familiar with the Sage 300 Desktop screen that will serve as the starting point.

Development of any screen in Sage 300 will use the desktop screen and code as a reference for creating the new screen. We will call this process Comparison Development. What better way to create a screen than with a functioning one as an example?

The Visual Basic 6 code of the legacy screen contains the majority of the information required to make the new screen successful. It contains the business logic required to drive the screen and interact with the business layer (business views).

**Note:** Not all code, logic and functionality will be brought into the new screen. For example, the new screen may not be as “chatty” with the business views; the web paradigm will aim to reduce this chattiness in order to reduce round trips to the server.

The technology stack is different for Sage 300, yet the interaction with the business layer and the functional requirements of the screen will remain largely the same. Interaction with the Sage 300 .NET API layer is via a new Business Entity layer (a wrapper around the API).

Numerous tools are leveraged throughout the creation process:

* Visual Studio 6 to research, evaluate, and debug the legacy screen.
* RVSpy and DBSpy to record the interaction of the screen as it interacts with the API and the database.
* Macros to record flow, interactions, and events.
* UI Info and View Doc to understand the business views associated with the screen.
  1. Process Overview

* 1. Links and References

| Resource | Location |
| --- | --- |
| Sage 300 .NET API | <http://smist08.wordpress.com/2013/10/12/an-introduction-to-the-sage-300-erp-net-api/> |
| Programming Views in .NET | <http://smist08.wordpress.com/2013/10/20/starting-to-program-the-sage-300-erp-views-in-net/> |
| Opening Sage 300 .NET API Session | <http://smist08.wordpress.com/2012/12/15/opening-sage-300-erp-sessions/> |
| Visual Basic 6 IDE | <http://www.vb6.us/tutorials/getting-know-vb6-ide> |
| RVSpy and DBSpy | <http://smist08.wordpress.com/2011/03/12/rvspy-and-dbspy/> |
| Sage 300 Business Logic | <http://smist08.wordpress.com/2010/09/11/accpacs-business-logic/> |
| Sage 300 Databases | <http://smist08.wordpress.com/2010/07/10/accpac-and-its-databases/> |
| Sonata Induction Kit | Sage300ERP-VBAnalysis-BestPractices v0 1.docx |
| Stephen Smith Blog | <https://smist08.wordpress.com> |
| John Thomas Blog | <https://jthomas903.wordpress.com> |
|  |  |

1. Coding Standards

Before getting start with screen development, it is important to understand all of the coding standards that have been implemented in Sage 300.

Refer to the following documents:

* *All-In-One Code Framework Coding Standards*

This is a Microsoft document that illustrates standards which Sage 300 has adopted

* *Sage 300 SDK Coding Standards (Web Screens)*

This is a Sage document that supplements and, at times, overrides the Microsoft document.

1. Getting Started

With a firm foundation of the screen, it is now time to start creating a screen in the new paradigm (an MVC application using C#)!

**Tip:** Always have access to the legacy screen for Comparison Development so you aren’t reinventing the wheel!

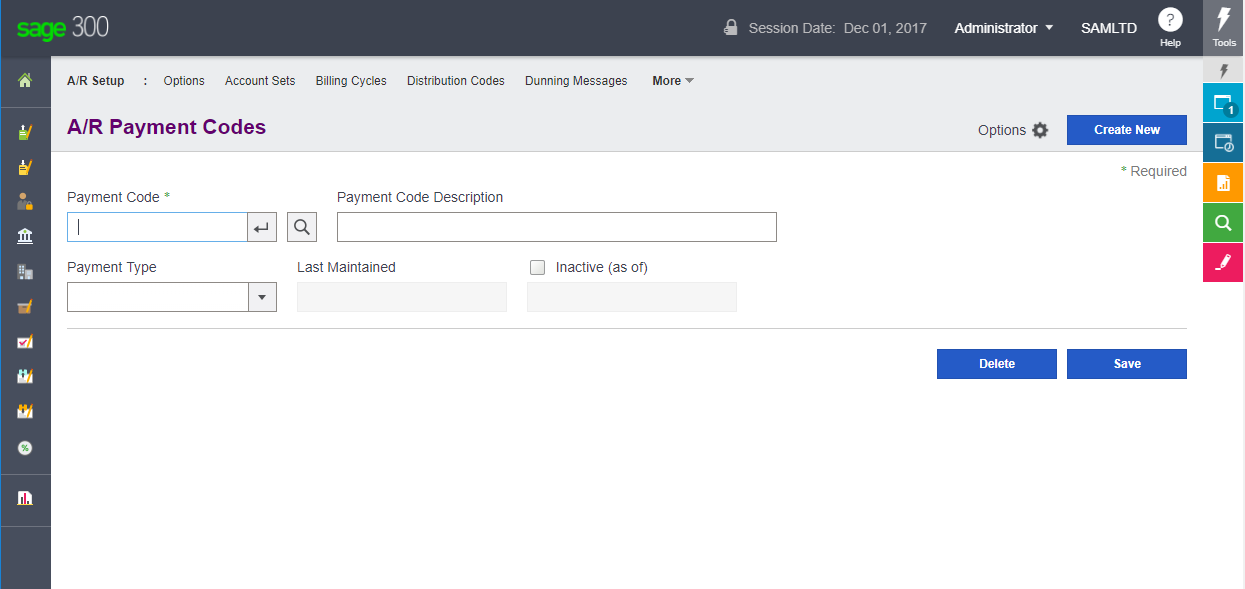
* 1. A/R Payment Codes Screen

The A/R Payment Codes screen (business view AR0012) is used as the sample screen in this document.

* + 1. A/R Payment Codes – Sage 300 Desktop



* + 1. A/R Payment Codes - Sage 300 Web Screen



1. Code Generation Wizard

The Code Generation Wizard is a C# plug-in developed in Visual Studio 2013, compatible with Visual Studio 2015, 2017 and invoked from Visual Studio. This is used to generate code files from a Sage 300 Business View or a Sage 300 Report, or from manually entered criteria for Dynamic Queries.

The types of files generated are:

* Model File

The new Business Entity which represents the Business View’s fields in the MVC pattern.

* Model Mapper File

Maps the Business View fields to the Business Entity properties.

* Model Fields File

Business Entity classes for referencing Business View fields by Index and/or by Name.

* Enumeration Files

Business View Presentation Lists created as enumerations for the Model.

* Razor View Files (for all code types other than Dynamic Query)
* Main Razor View.
* Partial Razor View to support localization.
* Partial Razor View to support the content of the screen (only Flat code types at this time).
* Interface Files

Business Entity interfaces for the Repository and Service layers.

* Service File

Business Entity Service class for the Repository.

* Repository File

Business Entity Repository for the Business Entity.

* View Model File

View Model for the Razor View.

* Controller File (Internal)

Internal controller (internal methods only) for the public controller.

* Controller File (Public)

Public controller (public methods).

* Finder Controller (optional)

Finder controller (internal controller) for the Business Entity.

* JavaScript Files (for only Flat code types at this time)
* Behaviour Script to support Razor View behavior.
* Knockout Extension Script to support special Knockout Binding.
* Repository Script to support callbacks to the Public Controller.
* Resx Files

The five (5) supported languages within Sage 300 will have an associated Resx file.

* The English Resx file is the default and it will be marked as Public.
* The other resx files (Spanish, French Canadian, Chinese Simplified and Chinese Traditional) will be marked as No Code Generation.

The existing files that are modified are:

* Bootstrapper Files

Module and Web Bootstrappers to support the UnityFramework.

* Bundle Files

Bundle Registration Files to support JavaScript Bundles.

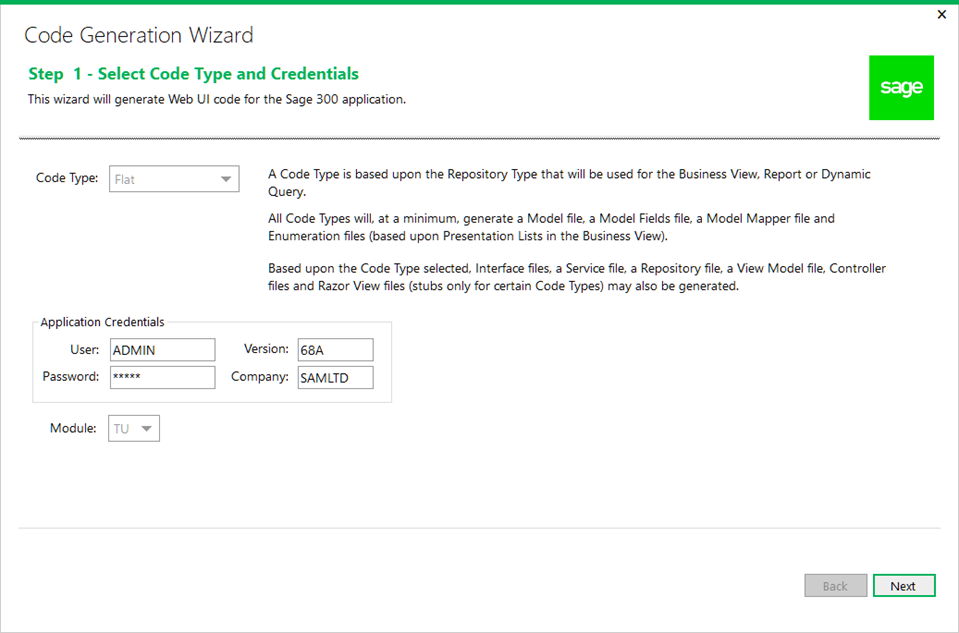
* Menu Files

Menu Details to support the Sage 300 Menu Navigation.

* 1. Reference

For more information, see *Sage 300 Web Screens SDK – Code Generation Wizard*.

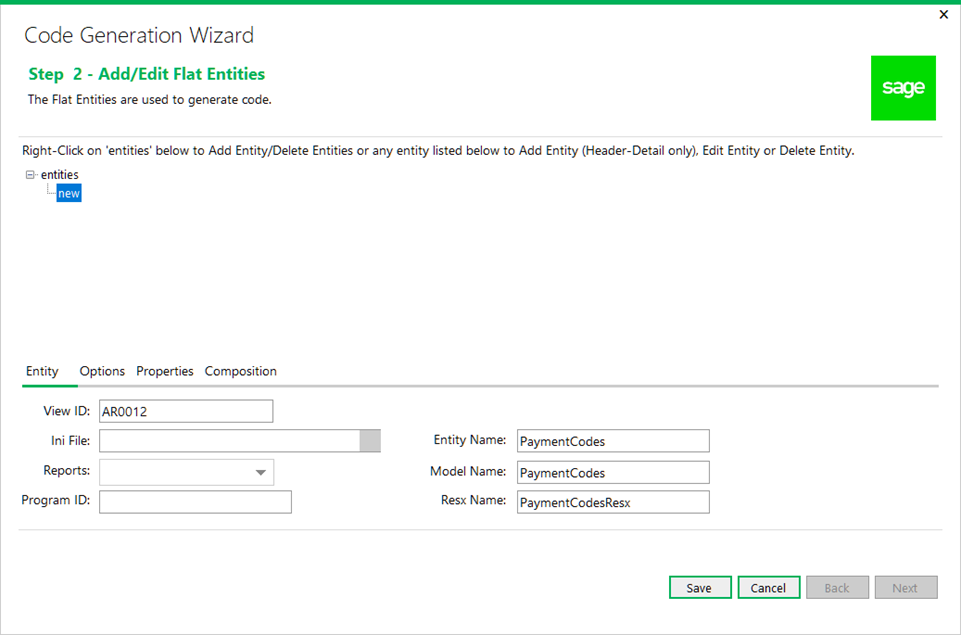
* 1. Wizard Inputs



For A/R Payment Codes, the wizard will require the following inputs:

Selection of the Flat code type because this is a simple setup screen (stateless).

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



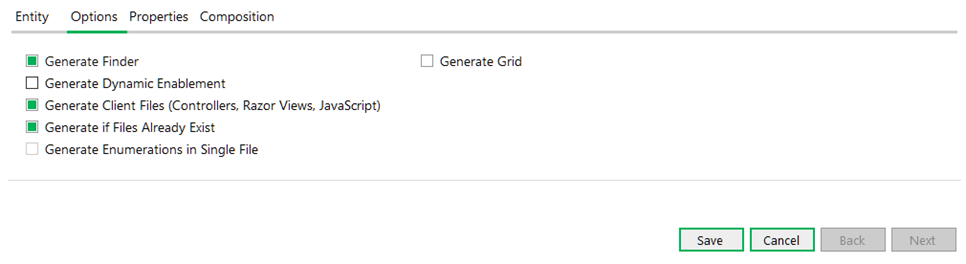
* The business view for AR Payment Code is AR0012.
* After entering the Business View, the Entity Name will default to PaymentCode for the Entity being created, which is generated from the Business View’s description.

The Entity Name column can be changed if a different name is required.

**Note:** In the case of the A/R Payment Codes screen, this was overridden to PaymentCodes since the Model will contain a field/property called PaymentCode. The Entity Name input allows this conflict to be resolved before code is generated. We have changed the Model Name to PaymentCodes as well.

* The Resx Name is defaulted from the Entity Name and can be overridden.

Other options can be selected by clicking on the Options tab:

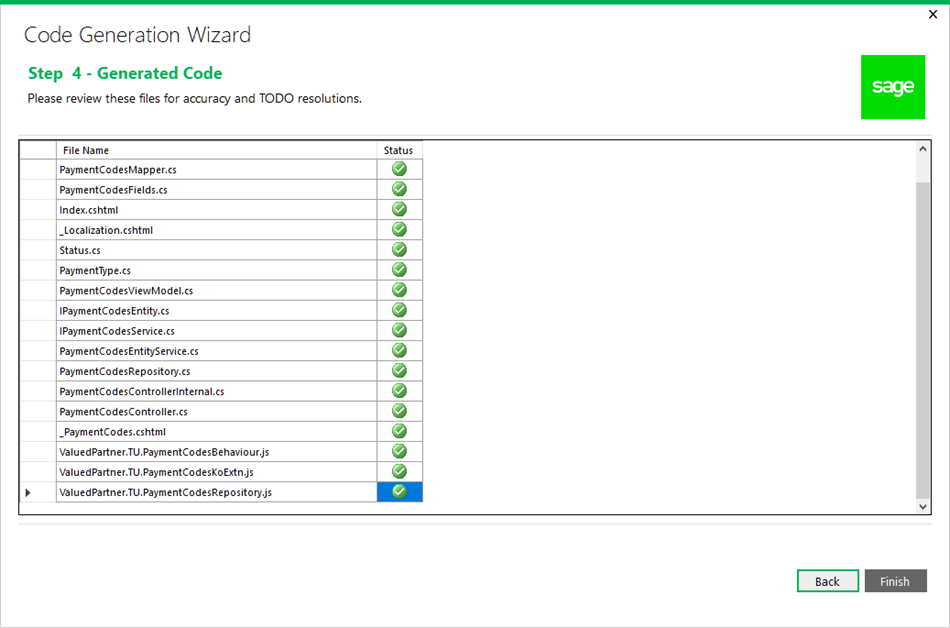


* The Generate Finder check box is optional and will generate the Finder’s internal controller for the view selected.
* The Generate Dynamic Enablement check box is optional and will generate the DynamicAttributes property in the Model Fields class. This provides a reverse mapping of the Server Fields (Business View) to the Model Properties (Business Entity) for those fields in the Business View that have the CheckEditable attribute.
* The **Generate Client Files** check box, if selected, will generate the client files for a Business View (controllers, Razor Views, and JavaScript files).
* The Generate if Files Already Exist check box is intended 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.
  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. The solution is able to be compiled and run to display the screen with a working Finder, and Save, Delete, and Create New buttons.

From this starting point, the real work begins in adding controls to the Razor Views, adding custom processing logic and generally wiring things up.

**Note:** This boiler-plating will still require evaluation to ensure correctness and potentially some TODO resolution (TODO statements may be included in the generated code where the wizard is unable to determine specific values required by the generated code).



* 1. Wizard Outputs

PaymentCodes.cs

* This is the model class.
* The name is singular by default unless overridden in the wizard.

PaymentCodesMapper.cs

* This is the mapping between the Sage 300 Business View (business view) and the Sage 300 Model (model), which is the Business Entity used in the MVC pattern.
* The base name is singular by default unless overridden in the wizard.

PaymentCodesFields.cs

* This is the class that holds the field names and index values for model and the business view which allow access to the Sage 300 .NET API either via the field/property name and/or its ordinal value.
* The names are generated from the description of the field/property names. Logic has been implemented to generate the best possible name for the model’s property.

**Note:** Each property name must be evaluated and modified where applicable. Do not assume that since the property name is generated that it must be accepted without question.

* The base name is singular by default unless overridden in the wizard.

Status.cs

This is an enumeration class created by the existence of a non-intrinsic data type in the business view. This is usually associated with a presentation list.

PaymentType.cs

This is an enumeration class created by the existence of a non-intrinsic data type in the business view. This is usually associated with a presentation list.

Index.cshtml

This razor view is the starting point for the screen being developed and will display the partial view(s) required to display the screen being developed.

\_Localization.cshtml

This razor view is used to localize the screen being developed. It only contains the minimum number of strings required by the generated code and is used as a starting point and example for other localized strings based upon the screen’s requirements.

\_PaymentCodes.cshtml

* This razor view is the partial view containing the minimal number of generated controls for the screen (key field, Create New button, Save button, Delete button, Import/Export options, and the Finder button)
* The base name is singular by default unless overridden in the wizard

IPaymentCodesEntity.cs

* This is the interface class for the Business Repository.
* Any public methods added to the Business Repository will be defined here.
* The base name is singular by default unless overridden in the wizard.

IPaymentCodesService.cs

* This is the interface class for the Business Service.
* Any public methods added to the Business Service will be defined here.
* The base name is singular by default unless overridden in the wizard.

PaymentCodesEntityService.cs

* This is the class for the Business Service.
* The Business Service is a layer between the Controller and the Repository.
* Any public methods added to the Business Service Interface will be implemented here.

This class is typically a pass through layer to the Business repository and does not usually contain any business logic.

* The base name is singular by default unless overridden in the wizard.

PaymentCodesViewModel.cs

* This is the class for the model being displayed in the razor view.
* If any enumeration classes for non-intrinsic data types have been created, public methods will be created in this class to access the enumeration list.

**Note:** If filtering of this list is required by the business requirements, these methods will require modification.

* The base name is singular by default unless overridden in the wizard

PaymentCodesControllerInternal.cs

* This is the class for the internal controller methods.

**Note:** These internal classes were simply created in an effort to simplify the public implementation.

* Methods have been created that represent basic, standard CRUD methods. These methods will require modifications for logic specific to the business entity being developed.
* The base name is singular by default unless overridden in the wizard.

PaymentCodesController.cs

* This is the class for the public controller methods.
* Methods have been created that represent basic, standard CRUD methods. These methods will require modifications for logic specific to the business entity being developed.
* The base name is singular by default unless overridden in the wizard.

PaymentCodesRepository.cs

* This is the class for the Business Entity and will contain the bulk of the business logic.
* The Business Repository is the destination layer which communicates with the business layer on the server.
* Any public methods added to the Business Repository Interface will be implemented here.
* The base name is singular by default unless overridden in the wizard.

FindPaymentCodesControllerInternal.cs

* This is the class for the finder’s controller methods for the Business Entity.
* A TODO statement is included to instruct the developer to that all columns have been added to the GetDefaultColumns method. Therefore, the developer will be required to modify this list if all columns are not to be defaulted.
* The base name is singular by default unless overridden in the wizard.

PaymentCodesResx.resx

This is the default Resx file for English. It will contain keys and values from the Business View and the Presentation Lists of certain fields.

PaymentCodesResx.es.resx

This is the Resx file for Spanish. It will contain only keys and blank values, if Spanish is selected in the Solution Wizard’s Resource Files Step.

PaymentCodesResx.fr-CA.resx

This is the Resx file for French. It will contain only keys and blank values, if French is selected in the Solution Wizard’s Resource Files Step.

PaymentCodesResx.zh-Hans.resx

This is the Resx file for Chinese Simplified. It will contain only keys and blank values, if Chinese Simplified is selected in the Solution Wizard’s Resource Files Step.

PaymentCodesResx.zh-Hant.resx

This is the Resx file for Chinese Traditional. It will contain only keys and blank values, if Chinese Traditional is selected in the Solution Wizard’s Resource Files Step.

**Note:** These code files have been generated based upon the information available in the business view and the coding standards for Sage 300. They will require evaluation and potential changes based upon the specific needs and requirements of the screen. We will continue to modify the wizard to generate the most accurate code possible.

* + 1. Evaluate Generated Code

At this point, it is important to evaluate and get acquainted with the generated code files.

5.4.1.1 Model Class – PaymentCodes.cs

* 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.

5.4.1.2 Model Mapper Class – PaymentCodesMapper.cs

* Evaluate the base.Map(entity) call in the Map getter routine. This is used for concurrency checking and is crucial.
* Evaluate the mapping of data from the business view to the model.

5.4.1.3 Model Fields Class – PaymentCodesFields.cs

* Evaluate EntityName variable which contains the business view.
* Evaluate the Fields class which is used to reference the business view by name.
* Evaluate the Index class which is used to reference the business view by ordinal value.

5.4.1.4 Enumeration Class – Status.cs

Evaluate the enumerations generated from the presentation list of the business view’s property.

**Note:** The enumeration has already been decorated with annotations from the generated resx file for localization

5.4.1.5 Enumeration Class – PaymentType.cs

Evaluate the enumerations generated from the presentation list of the business view’s property

Note: The enumeration has already been decorated with annotations from the generated resx file for localization.

5.4.1.6 Interface Class – IPaymentCodesEntity.cs

There are no changes required until new public methods are implemented.

5.4.1.7 Interface Class – IPaymentCodesService.cs

There are no changes required until new public methods are implemented.

5.4.1.8 Repository Class – PaymentCodesRepository.cs

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).

5.4.1.9 Service Class – PaymentCodesEntityService.cs

There are no changes required until new public methods are implemented.

5.4.1.10 Razor View – Index.cshtml

* This is the main view for the screen. It is merely a shell and will host the other partial view or views that make up the screen.
* Partial views are used to encapsulate logic, to reduce clutter when looking at a screen and to provide reuse.
* The screens are segmented into partial views based upon format, grouping or other considerations.

5.4.1.11 Razor View – \_Localization.cshtml

* This is the view for the screen that contains localized strings for both the views and JavaScript files.
* All strings localized in this view will be wrapped in the SageRaw helper function to eliminate any unwanted characters
* No further action at this time until additional localized strings are required. This will be addressed when the partial view for the screen is developed.

5.4.1.12 Razor View – \_PaymentCodes.cshtml

This is the view for the screen that contains the generated controls. This is view that will be enhanced to display additional controls.

5.4.1.13 View Model – PaymentCodesViewModel.cs

There are no changes required unless the razor view, once developed, has new requirements for the screen.

5.4.1.14 Controller – PaymentCodesControllerInternal.cs

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

5.4.1.15 Controller – PaymentCodesController.cs

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.

5.4.1.16 Controller – FindPaymentCodesControllerInternal.cs

Internal methods are only used by a generic public controller for all Finders.

5.4.1.17 Resx File – PaymentCodesResx.resx

English resource file containing strings for all generated columns in the model in addition to descriptions from the Presentation Lists (used by Enumeration Classes).

5.4.1.18 Resx File – PaymentCodesResx.es.resx

Spanish resource file containing strings for all generated columns in the model in addition to keys only from the Presentation Lists (used by Enumeration Classes), if Spanish is selected in the Solution Wizard’s Resource Files Step.

5.4.1.19 Resx File – PaymentCodesResx.fr-CA.resx

French resource file containing strings for all generated columns in the model in addition to keys only from the Presentation Lists (used by Enumeration Classes), if French is selected in the Solution Wizard’s Resource Files Step.

5.4.1.20 Resx File – PaymentCodesResx.zh-Hans.resx

Chinese Simplified resource file containing strings for all generated columns in the model in addition to keys only from the Presentation Lists (used by Enumeration Classes), if Chinese Simplified is selected in the Solution Wizard’s Resource Files Step.

5.4.1.21 Resx File – PaymentCodesResx.zh-Hant.resx

Chinese Traditional resource file containing strings for all generated columns in the model in addition to keys only from the Presentation Lists (used by Enumeration Classes), if Chinese Traditional is selected in the Solution Wizard’s Resource Files Step.

5.4.1.22 JavaScript File – …PaymentCodesBehaviour.js

The events for the generated controls and base display logic has already been generated. This script file will be enhanced when additional controls are added to the screens Razor View.

5.4.1.23 JavaScript File – …PaymentCodesKoExtn.js

The minimal Knockout binding has already been generated. This script file will be enhanced when additional controls are added to the screens Razor View for computed fields.

5.4.1.24 JavaScript File – …PaymentCodesRepository.js

The methods for the generated controller actions have already been generated. This script file will be enhanced when additional controller actions are added.

1. Portal Menu Help

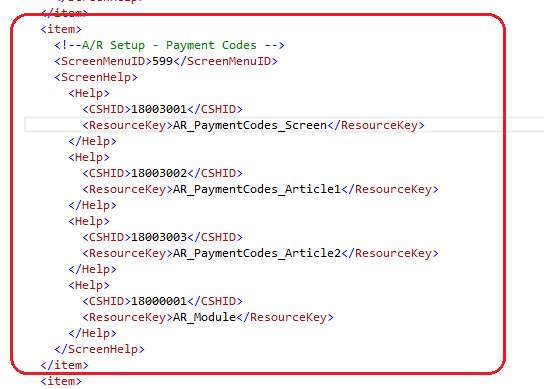
Menu Help is available via the Portal.

The current storage structure is an XML file which resides locally in the application.

* 1. Adding to the Portal Menu Help

Portal Menu Help IDs are specified by the Product Design & Experience team.

1. Locate the MenuHelp.xml file of the web project in the App\_Data folder.
2. Locate the appropriate location for the help and add the item for the A/R Payment Codes 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.

See the Appendix for a list of the code files affected by the following changes.

* 1. Payment Codes Business Entity Interface

Access the IPaymentCodesEntity.cs class and add the UpdateInactiveStatus and IsModuleActive routines to the class.

See *9.1 Payment Codes Business Entity Interface* for reference.

* 1. Payment Codes Service Interface

Access the IPaymentCodesService.cs class and add the UpdateInactiveStatus and IsModuleActive routines to the class.

See *9.2 Payment Codes Service Interface* for reference.

* 1. Payment Codes Service

Access the PaymentCodesEntityService.cs class and add the UpdateInactiveStatus and IsModuleActive routines to the class.

See *9.3 Payment Codes Service* for reference.

* 1. Payment Codes Repository

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

See *9.4 Payment Codes Repository* for reference.

* 1. Payment Codes Model

Access the PaymentCodes.cs class and add the RegularExpression annotation to the PaymentCode property.

See *9.5 Payment Codes Model* for reference.

* 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.

See 9.6 *Payment Codes View Model* for reference.

* 1. Payment Codes Internal Controller

Access the PaymentCodesControllerInternal.cs class and add the private constants PaymentAndProcessingModuleId and PaymentProcessing.

Add the UpdateInactiveStatus routine to the class.

Modify the GetViewInfo routine and add the FilteredPaymentTypes routine to the class.

See *9.7 Payment Codes Internal Controller* for reference.

* 1. Payment Codes Controller

Access the PaymentCodesController.cs class and add the UpdateInactiveStatus routine to the class.

See *9.8 Payment Codes Controller* for reference.

* 1. Payment Codes Partial Razor View

Access the \_PaymentCodes.cshtml file and add the remaining controls to the screen.

See *9.9 Payment Codes Partial Razor View* for reference.

* 1. Payment Codes Repository JavaScript

Access the …PaymentCodesRepository.js file and add the updateInactiveStatus method.

See *9.10 Payment Codes Repository JavaScript* for reference.

* 1. Payment Codes Knockout Extension JavaScript

Access the …PaymentCodesKoExtn.js file and add the new computed properties for Status, InactiveDate, and DateLastMaintained.

See *9.11 Payment Codes Knockout Extension JavaScript* for reference.

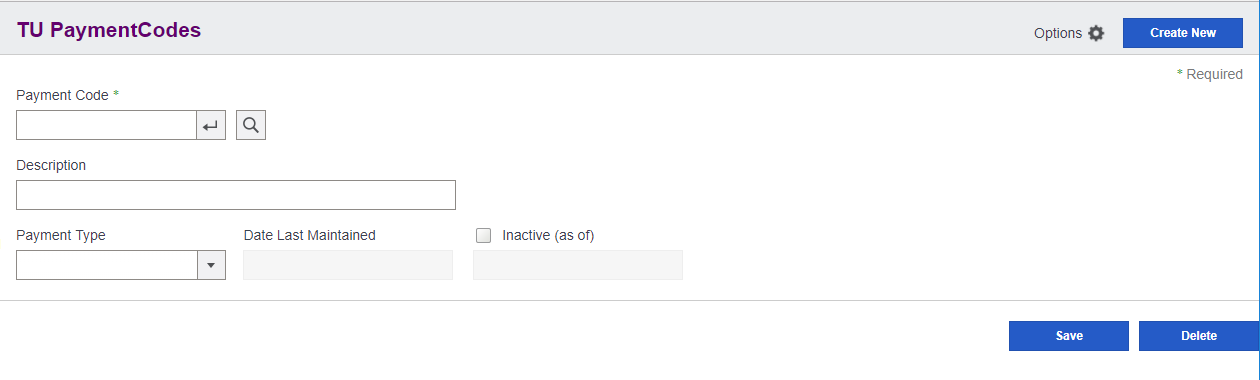
* 1. Payment Codes Behaviour JavaScript

Access the …PaymentCodesBehaviour.js file and add the business logic for the new screen controls.

See *9.12 Payment Codes Behaviour JavaScript* for reference.

1. Review the Screen

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



1. Appendix: Complete Code Files

This section contains completed code files for most code files. The highlighted text is code that was manually entered in the Specific Changes section (that is, code that is not generated by the wizard).

* 1. Payment Codes Business Entity Interface
     1. IPaymentCodesEntity.cs

|  |
| --- |
| // 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
     1. IPaymentCodesService.cs

|  |
| --- |
| // 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 Service
     1. PaymentCodesEntityService.cs

|  |
| --- |
| // 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
     1. PaymentCodesRepository.cs

|  |
| --- |
| // 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
     1. PaymentCodes.cs

|  |
| --- |
| // 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
     1. PaymentCodesViewModel.cs

|  |
| --- |
| // 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
     1. PaymentCodesControllerInternal.cs

|  |
| --- |
| // 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
     1. PaymentCodesController.cs

|  |
| --- |
| // 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
     1. \_PaymentCode.cshtml

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
     1. ...PaymentCodesRepository.js

|  |
| --- |
| /\* 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
     1. …PaymentCodesKoExtn.js

|  |
| --- |
| /\* 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
     1. …PaymentCodesBahviour.js

|  |
| --- |
| /\* 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();  }  });  }); |