How to use the VS extension

To support you developing plugins for Smartstore we've created a VS extension which will add the skeletal structure of a plugin to your Smartstore solution.

How to install

You can find the extension by going to *VisualStudio > Extensions > Manage Extensions > Visual Studio Marketplace > Search for SmartStore > Download*

Or you can download it *here* and install via double click.

How to create a plugin

Make a right click on the plugins node in your solution and choose *Add > New Project*. Now select the Smartstore project template. The physical location to store the project on your disc must be the plugins directory of your Smartstore repository.

Settings

Project name

Defines the name of the project. This will be used as the assembly name and the Smartstore plugin system name which is used to identify the plugin within Smartstore.

Plugin name

Defines the name of the project. The name should be choosen very carefully as it is used in various places as object names like names for controller, service, domain, etc.

Author name

Defines the name of the author as it is displayed in the plugin list in the backend of the shop admins who are using the plugin.

Plugin type

Defines the type of the plugin you're about to create.

- General: Creates a plugin with the specified options.
- Payment: Creates a payment plugin. Therefore the plugin derives from PaymentPluginBase and implements the necessary methods (ProcessPayment, PostProcessPayment, GetAdditionalHandlingFee, CanRePostProcessPayment, GetPaymentInfoRoute).
- Widget: Creates a widget plugin. Therefore the plugin derives from IWidget & IConfigurable and implements the necessary methods (GetWidgetZones, GetDisplayWidgetRoute, GetConfigurationRoute). Also the action method for public display as well as the corresponding view (PublicInfo) will be added.
- Feed: Creates a feed plugin. Therefore a provider class which derives from ExportProviderBase will be added along with ProfileConfiguration.cshtml.

Plugin group

Defines the plugin group in which the plugin is displayed in the plugin list in the backend of the shop.

Plugin is licensable

Defines whether the plugin is licensable. If this option is active the plugin will be decorated with the corresponding attribute.

Plugin has own data context (EF Migration)

Adds a custom data context to the plugin. This option should be used if you're planing to use your own domain objects in your plugin. The option also adds a rudimentary service interface and service class to manage the domain object.

Plugin integrates into backend by its own tabs

Use this option if the plugin should manage additional data which will be gathered in dependency of existing entities like Product, Category or Topic. The option will add code which integrates a tab into the configuration details pages of the aforementioned entities.

Shop-Admin can edit Widgetzone

This option will add code to the configuration page of the plugin where the shop admin can choose the widgetzone where the public output of the plugin (the widget) shall be displayed.

Add admin menu

Adds a class to the plugin which derives from AdminMenuProvider and thus can add menu items to the existing admin navigation.

Add filter

Adds a barebone filter class along with the corresponding registration for Autofac (in the DependencyRegistrar class) to the plugin.

Add hook

Adds a barebone hook class to the project.

Add task

Adds a barebone task class to the project along with the code which will add the task to the list of sheduled tasks upon plugin installation.

Add validator

Adds refernces of the mandatory libraries for validation (FluentValidation) to the project and adds a small sample validator to the plugin configuration model.

Add permission provider

Adds a permission provider to the plugin.

Add event consumer

Adds a barebone event consumer class to the project.

Add starter (IApplicationStart)

Adds a barebone starter class to the project which derives from IApplicationStart.

Add starter (IPreApplicationStart)

Adds a barebone starter class to the project which derives from IPreApplicationStart.

Add starter (IPostApplicationStart)

Adds a barebone starter class to the project which derives from IPostApplicationStart.

Add sample configuration properties

Adds properties to the configuration view of the plugin. This section provides best practices implementations for common properties like Media, HTML-Editor, etc.