Build DNNrocket Module

# Introduction

This will describe how to create an administration interface for DNNrocket.

# Start

Create Visual Studio project in the /DesktopModules/DNNrocketModules. With a “.Net Standard” class library.

Rename class library to “StartConnect.cs”, just to give a standard name across modules. (CamelCase)

Add existing projects “DNNrocket API” and “Simplisity” (under ..\DNNrocket) and set a reference.

The StartConnect class should inherit from “DNNrocketAPI.APInterface”. And implement the abstract methods.

The DNNrocketAPI project has example “TestForm” and “TestList” to get you started. But other, real life projects would be better examples.

In after build copy the modules:

copy "$(ProjectDir)$(OutDir)$(TargetFileName)" "$(ProjectDir)..\..\..\bin\$(TargetFileName)"

copy "$(ProjectDir)$(OutDir)$(AssemblyName).pdb" "$(ProjectDir)..\..\..\bin\$(AssemblyName).pdb"

using DNNrocketAPI.Componants;

using Simplisity;

using System.Collections.Generic;

namespace RocketCatalog

{

public class StartConnect : DNNrocketAPI.APInterface

{

private SimplisityInfo \_postInfo;

private SimplisityInfo \_paramInfo;

private RocketInterface \_rocketInterface;

private SystemLimpet \_systemData;

private Dictionary<string, string> \_passSettings;

private string \_editLang;

private string \_currentLang;

private string \_nextLang;

private SessionParams \_sessionParams;

private int \_portalId;

private UserParams \_userParams;

public override Dictionary<string, object> ProcessCommand(string paramCmd, SimplisityInfo systemInfo, SimplisityInfo interfaceInfo, SimplisityInfo postInfo, SimplisityInfo paramInfo, string langRequired = "")

{

var strOut = ""; // return nothing if not matching commands.

paramCmd = InitCmd(paramCmd, systemInfo, interfaceInfo, postInfo, paramInfo, langRequired);

var rtnDic = new Dictionary<string, object>();

switch (paramCmd)

{

case "rocketserver\_test":

strOut = "TEST";

break;

}

// -----------------------------------------------------------------------

// if we have changed language, reset the editlang. The \_nextLang is defined on the "InitCmd" function.

if (\_nextLang != \_editLang) DNNrocketUtils.SetEditCulture(\_nextLang);

// -----------------------------------------------------------------------

rtnDic.Add("outputhtml", strOut);

return rtnDic;

}

public string InitCmd(string paramCmd, SimplisityInfo systemInfo, SimplisityInfo interfaceInfo, SimplisityInfo postInfo, SimplisityInfo paramInfo, string langRequired = "")

{

\_postInfo = postInfo;

\_paramInfo = paramInfo;

\_currentLang = DNNrocketUtils.GetCurrentCulture();

// -----------------------------------------------------------------------

// Change of language.

// \_nextlang is used for returning data.

// \_editlang is used to save the data and reset to \_nextLang at end of processing in "ProcessCommand" method.

\_editLang = DNNrocketUtils.GetEditCulture(); // set editlang from url param or cookie

\_nextLang = \_paramInfo.GetXmlProperty("genxml/hidden/nextlang");

if (\_nextLang == "") \_nextLang = \_editLang; // default to editLang

DNNrocketUtils.SetNextCulture(\_nextLang); // set the next langauge to a cookie, so the "EditFlag" razor token works.

// -----------------------------------------------------------------------

\_systemData = new SystemLimpet(systemInfo);

\_rocketInterface = new RocketInterface(interfaceInfo);

if (\_paramInfo.GetXmlPropertyBool("genxml/hidden/reload"))

{

var menucmd = \_userParams.GetCommand(\_systemData.SystemKey);

if (menucmd != "")

{

paramCmd = menucmd;

\_paramInfo = \_userParams.GetParamInfo(\_systemData.SystemKey);

var interfacekey = \_userParams.GetInterfaceKey(\_systemData.SystemKey);

\_rocketInterface = new RocketInterface(systemInfo, interfacekey);

}

}

else

{

if (\_paramInfo.GetXmlPropertyBool("genxml/hidden/track")) \_userParams.Track(\_systemData.SystemKey, paramCmd, \_paramInfo, \_rocketInterface.InterfaceKey);

}

var securityData = new SecurityLimet(\_portalId, \_systemData.SystemKey, \_rocketInterface, -1, -1);

paramCmd = securityData.HasSecurityAccess(paramCmd, "rocketecommerce\_login");

return paramCmd;

}

}

}

# ProcessCommand

This is the abstract class that needs to be inhertited. It is the entry point for the API provider. The API controller “DNNrocketAPI.ProcessAPI” passed the data to this method, which then must process the call.

Usually this processing is done with a switch command, which takes the “paramCmd” and executes the code related to that command.

# Themes

Create a “Themes>-config- w3>1.0>default” folder. This is where the razor template will be kept. Sub-Folders of “config-w3\1.0\css” and “config-w3\1.0\js” can be created if required.

# Razor Templates

Create a razor template under the “Themes>w3-config>1.0>default”, this will be your default template. The name can be anything you want. Usually there will be multiple razor templates per interface.

# Admin.html

The “admin.html” will be the admin start point of the module system and will be defined in the DNNrocket systemadmin record as “Admin URL”.

When linking to DNN, you need the tabid and moduleid passed as url parameters, so they can be passed to the API code for DNN module security.

If creating a DNN module. The “<moduleName>” in the DNN manifesto must match the “systemprovider\_systeminterface”

<moduleName>rocketecommerce\_paymentform</moduleName>

This is how the DNN modules are linked to the DNNrocket API system. (See RocketMod for an example)

\*\*\*WARNING\*\*\* DNN security is linked to the admin by using the TabId and ModuleId. (Plus the DNNrocket roles). Do NOT forget to do this DNN security link for modules.

# SideMenu.cshtml

In the admin.cshtml we have a link to the sidemenu.cshtml. This is optional.

The SideMenu.cshtml is saved to the “/Theme/condif-w3/1.0/default” folder. And the admin.html should be adjusted as required.

@inherits DNNrocketAPI.render.DNNrocketTokens<Simplisity.SimplisityRazor>

@using DNNrocketAPI;

@using System

@using System.Linq

@using System.Xml

@using Simplisity;

@using DNNrocketAPI.Componants;

@AddProcessData("resourcepath", "/DesktopModules/DNNrocket/api/App\_LocalResources/")

@AddProcessData("resourcepath", "/DesktopModules/DNNrocket/RocketMod/App\_LocalResources/")

<div class="w3-row w3-padding-small w3-theme-d5 w3-border-bottom w3-padding">

<div class="w3-col w3-large"><i class='fas fa-rocket' style='font-size:36px'></i>&nbsp;Rocket Mod</div>

<div class="w3-col w3-hide-large w3-button w3-display-topright" onclick="w3\_close()" style="cursor:pointer;width:40px;" title=""><i class='fas fa-caret-left' style='font-size:36px'></i></div>

</div>

@RenderSideMenu(Model, "DNNrocket/RocketMod", "RocketMod", true, false, true)

The template Renders the side menu using a standard code, that builds the menu from the system interfaces.

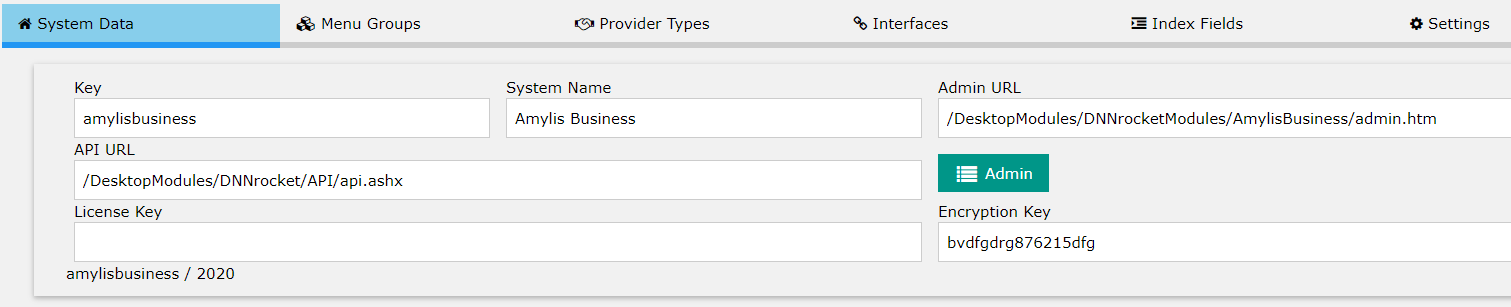
# RESX resource

Resource files can be created, by convention these files are placed under a “App\_LocalResources” folder. The resx file is the same name as the theme or project, again by convension.

# Creating a System

To create a system use “/Desktopmodules/dnnrocket/adminsystem.html”.

Add new system. (Example Below)



**Key**: Unique key for system.

**System Name**: Display name

**Admin URL**: start point of admin interface.

**API url**: API interface url. Usually this will be the rocket API, which will then redirect to the correct system. “/Desktopmodules/dnnrocket/api/rocket/action”.

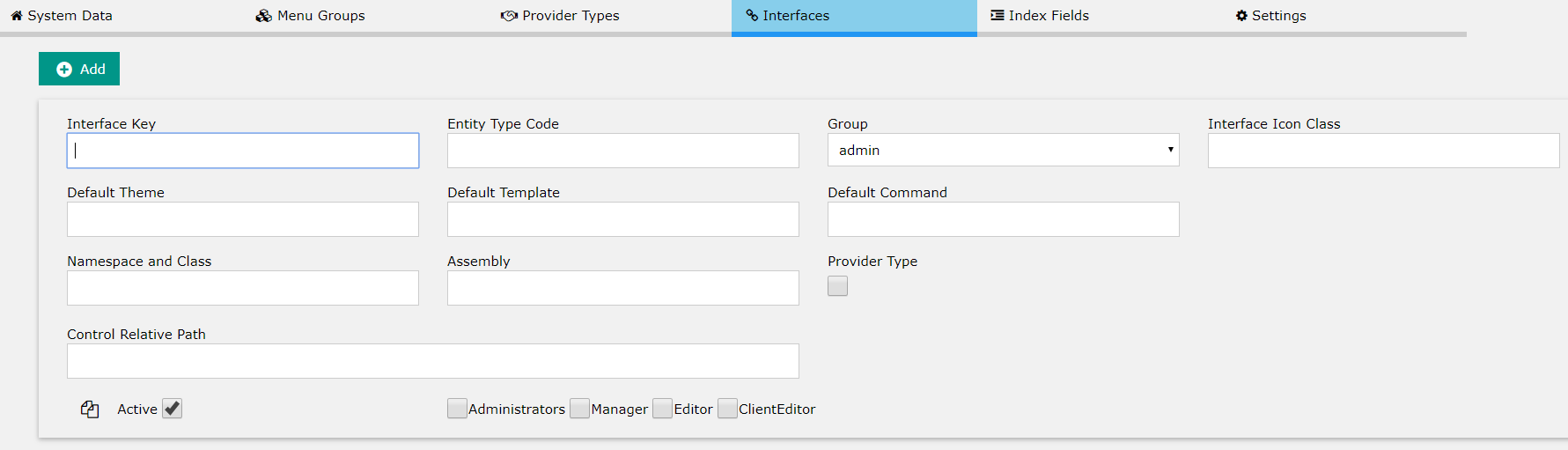
**Licence Key**: License.

**Encryption Key**: This is used to encrypt data in the system and is required.

# Menu Groups

Menu groups appear on the DNNrocket side menu in the admin panel. The side menu is a 2nd level structure, usually the menu groups are the first level, the interfaces are the second level. However, if you do not select a group on the interface, the interface becomes a 1st level link.

# Admin Interface



**Interface Key**: Unique key for the interface.

**Entity Type Code**: The data entity TypeCode, which is the database TypeCode that simplicity uses.

**Group**: Menu Group.

**Interface Icon Class**: Icon.

**Default Theme**: Theme folder which is used by default. The is a single folder under the “Themes” folder of the “Control Relative Path”.

**Default Template**: Default Template. Usually the starting template.

**Default Command**: Default Command. Usually the starting command.

**Namespace and Class**: The namespace and class that the provider will use. This is the namespace and class of the DNNrocket provider.

**Assembly**: The assembly name of the DNNrocket provider. This is used to initiate the assembly.

**Provider Type**: Selected provider type, system defined.

**Control Relative Path**: The relative path to the “Themes” folder and the interface root folder.

**Security**: The default security roles for the interface.