4/13/2019

Workflow

Contents

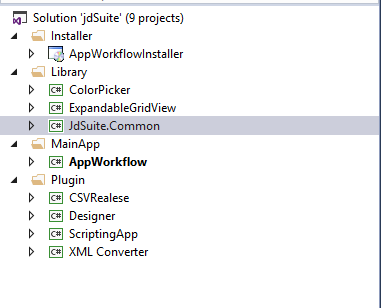
[Project Structure 2](#_Toc6086305)

[Physical File Structure 2](#_Toc6086306)

[How to add Module/Plugin in system 3](#_Toc6086307)

# Project Structure

There are four section in solution as shown below in image.



1. Installer

Installer project will reside in this folder.

1. Library

All Supported library which used in applications like Color Picker, Expandable Grid View etc.

This library is supported library of the main application

1. MainApp

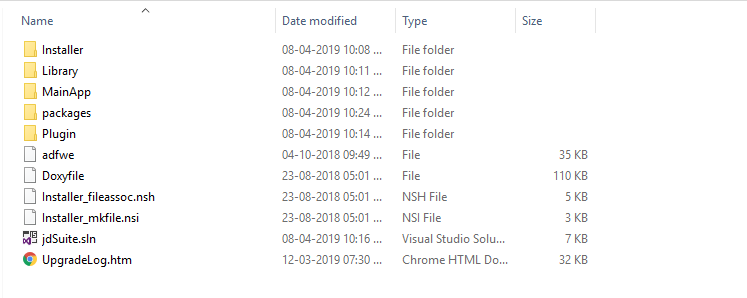
Main Application is the container which loads all plugins.

1. Plugin

All modules which meant to be integrate into main application will reside into this folder.

# Physical File Structure

As this project having different library and modules so we must manage it with proper structure.



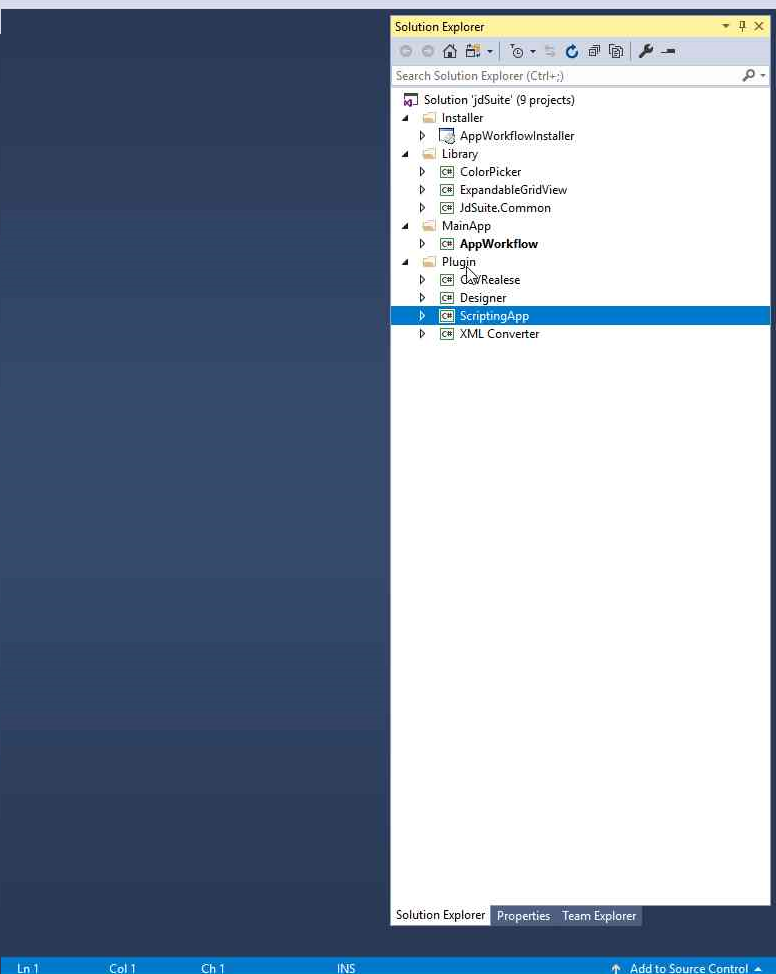
As shown in above image, the folder structure is same as project structure so it will be easy to understand and manage the things.

Developer need to take care that in future if he/she creates any project then it must be placed in specific folder based on need so by that way system will be manageable.

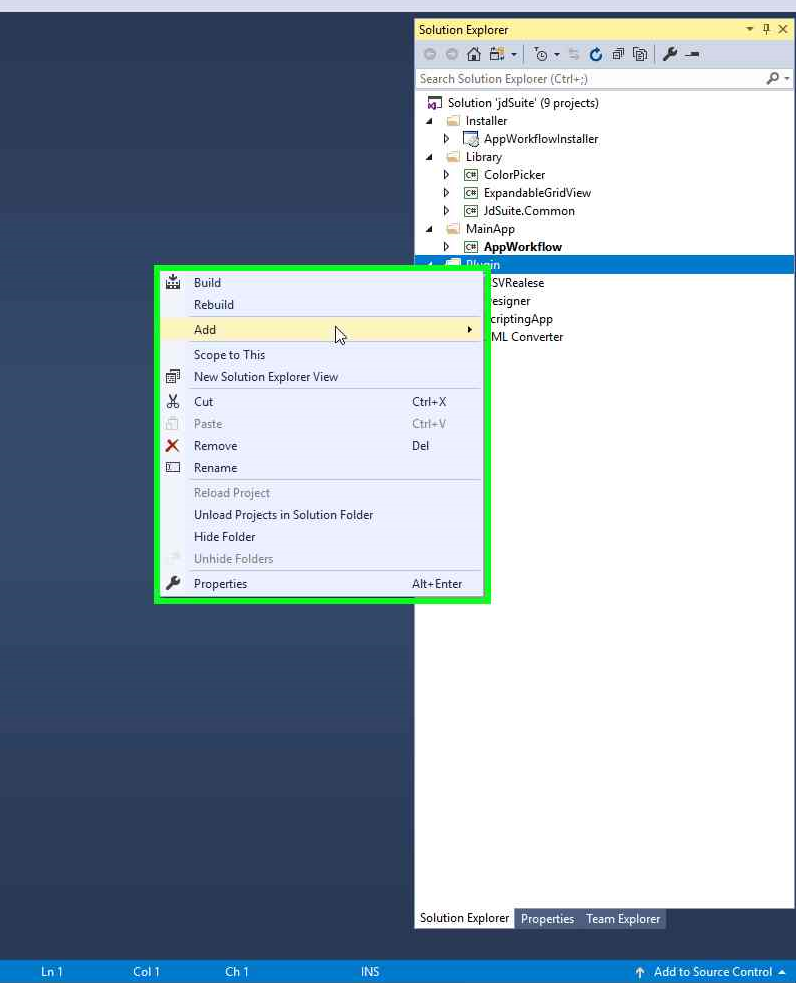
# How to add Module/Plugin in system

Developer can develop individual plugin and integrate into system easily for that they need to perform below steps:

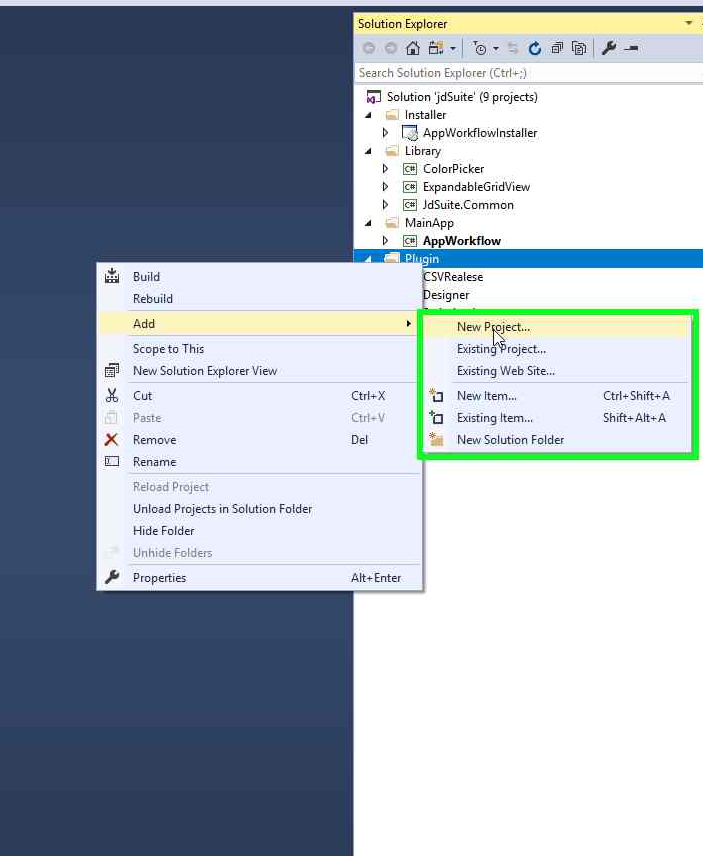
**Step 1:** Right click in "Plugin" folder of “jdSuite”.

Step 1 screenshot.

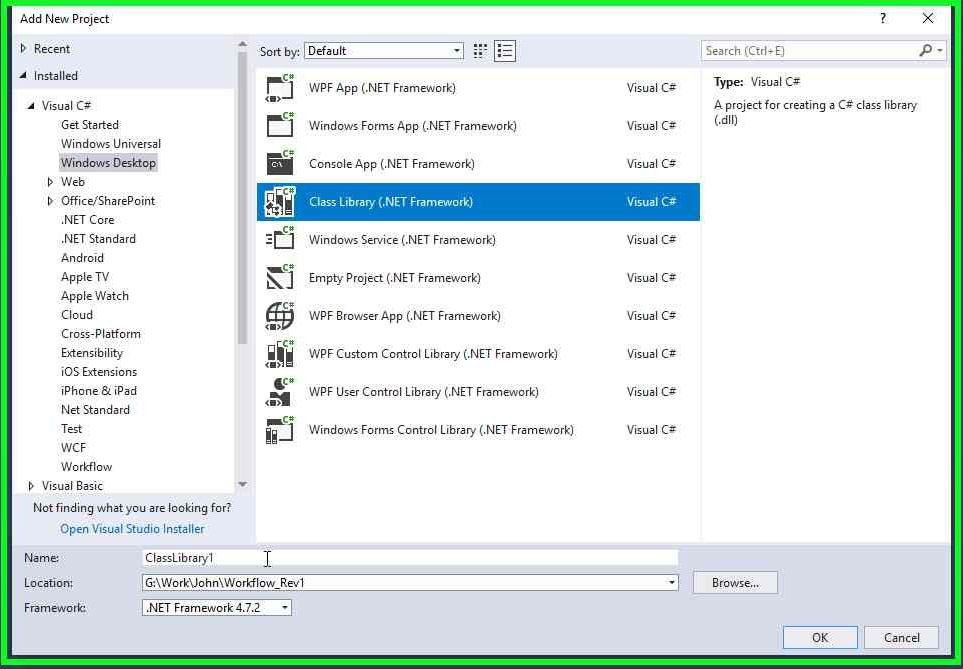
**Step 2:** Click on "Add (menu item)".

Step 2 screenshot.

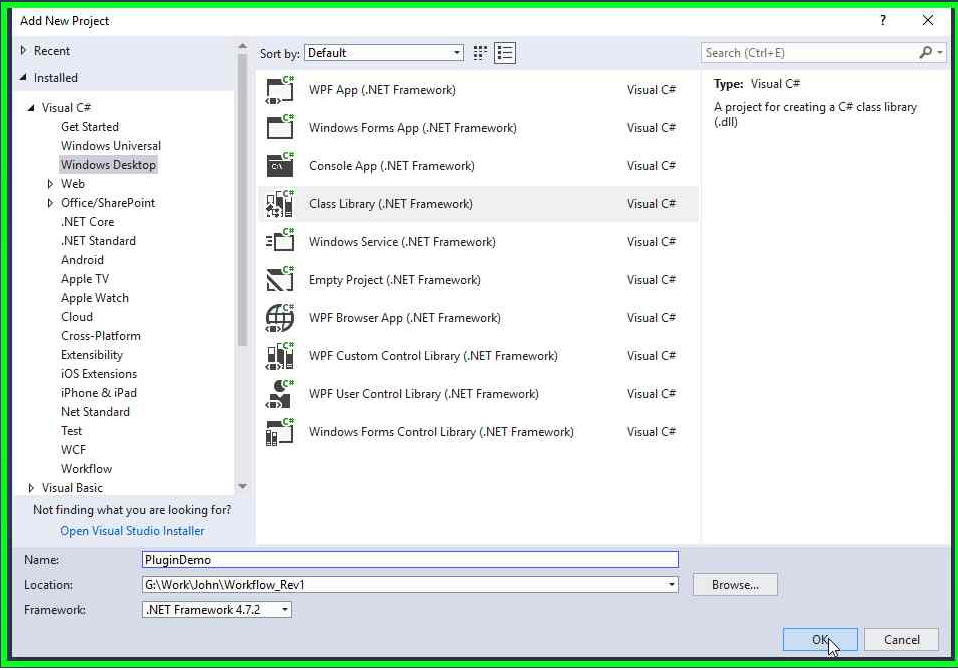
**Step 3:** Click on "New Project...".

Step 3 screenshot.

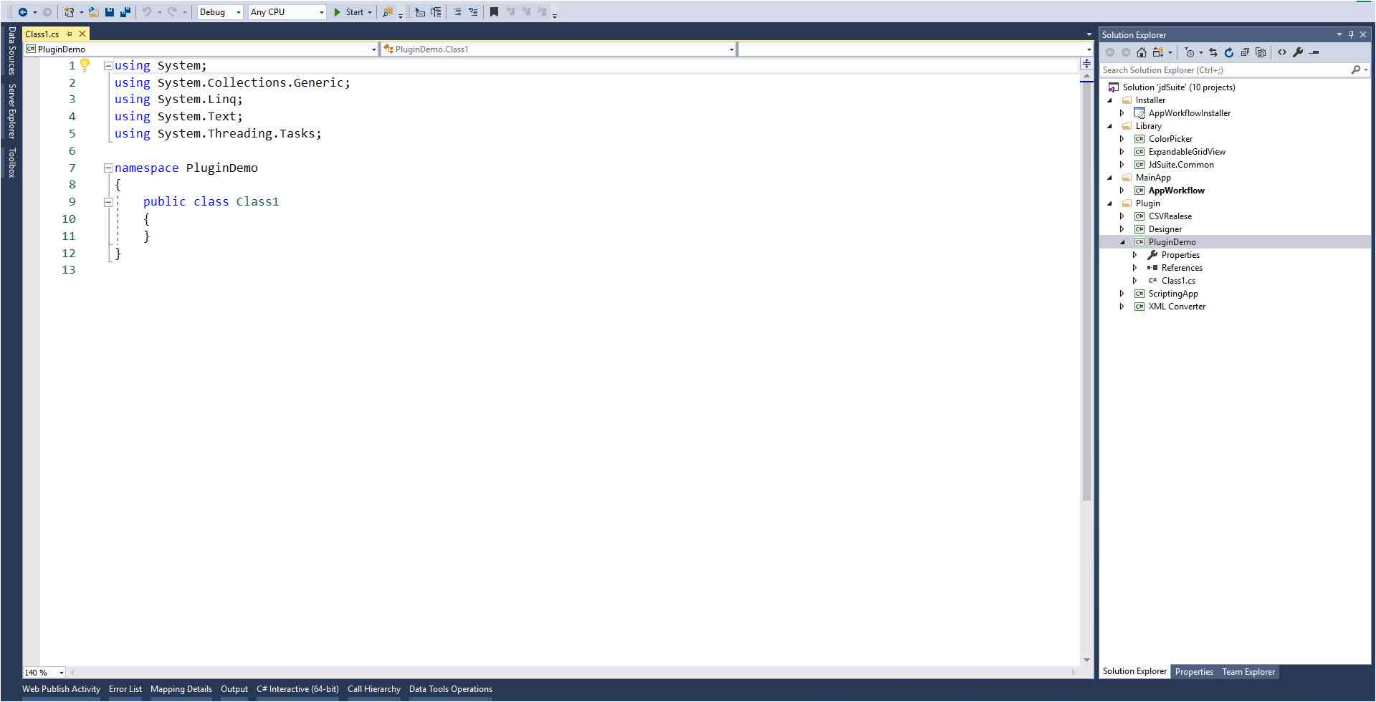
**Step 4:** Click on "Class Library (.NET Framework)" in "Add"

Step 4 screenshot.

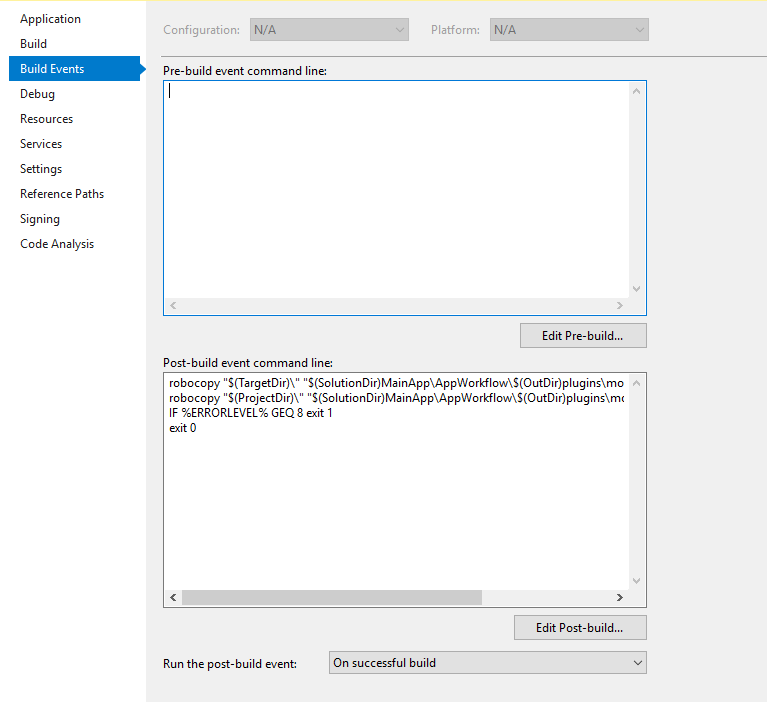
**Step 5:** Provide path upto the plugin folder of jdSuite project in location and provide proper module name in Name.

Step 5 screenshot.

**Step 6:**After successful performed steps you can see your module is added into jdSuite.

Step 6 screenshot.

**Step 7:** Right Click on project and Go To properties -> Debug Events. Past below content in Post build event command line text box:



**Content:**

robocopy "$(ProjectDir)res\\" "$(TargetDir)res\\" /MIR

robocopy "$(TargetDir)\" "$(SolutionDir)MainApp\AppWorkflow\$(OutDir)plugins\modules\$(ProjectName)\\" /E

robocopy "$(ProjectDir)\" "$(SolutionDir)MainApp\AppWorkflow\$(OutDir)plugins\modules\$(ProjectName)\\" config.xml

IF %ERRORLEVEL% GEQ 8 exit 1

exit 0

**Step 8:** Add project reference of JdSuite.Common in newly created project.

**Step 9:** Add Module.cs file in the project.

Step 10: Add three attributes on Module class.

[Export(typeof(IModule)), PartCreationPolicy(CreationPolicy.NonShared)]

[ExportMetadata("ModuleCategory", IModuleCategory.DATA\_INPUTS)]

[ExportMetadata("ModuleType", typeof(Module))]

**Step 11:**User can set any category from below:

1. DATA\_INPUTS
2. DATA\_MANIPULATION
3. DESIGNS
4. IMPOSITION
5. OUTPUTS
6. MISC

**Step 12:** Add Icon into Resource file of the system and name it as “Icon”.

**Step 13:** InheritBaseModuleinto Module class. Implement four methods in module class. Override Execute method to run the plugin.

publicclassModule :BaseModule

{

protectedoverridestringModuleName{ get { return"<Module Name>"; } }

protectedoverridestring DisplayName { get { return"<Display Name>"; } }

protectedoverride Bitmap Icon { get { return"<Image>"; } }

protectedoverridestring ToolTip { get { return"<tool tip>"; } }

publicoverrideobject Execute(Workflow workflow)

{

returnbase.Execute(workflow);

}

}

**Step 14:**To Add Output Node in Box you need to perform below code in constructor of module.

Node = newOutputNode(this, "<Display Name>", "<Extension>");

AddOutputNode(Node);

**Step 15:**To Add Input Node in Box you need to perform below code in constructor of module.

Node = new InputNode(this, ""<Display Name>", "<Extension>");

AddInputNode(Node);

**Step 16:**below is the state object which we transfer from one module to another module by connection

[Serializable]

publicclassModuleState

{

publicModuleState() { }

[XmlAttribute]

publicstring InputPath { get; set; } = String.Empty;

[XmlAttribute]

publicbool InputIsSchema { get; set; } = false;

[XmlAttribute]

publicstring TextEncoding { get; set; } = String.Empty;

public Field Schema { get; set; }

}

# How state management works?

In current System, in each module we maintain states of each connector. Each connectorhasits own identity and state.

To Initialize state you need to check by below code if exist then do not reinitialize state.

if (this.\_node.State == null)

this.\_node.State = newModuleState();

We maintain state for only output nodes. So, if any module hasoutput nodes in that module, we can produce state and that state can be utilize by connected module via that connection. To Access that state

Var State = ((BaseOutputNode)Node.GetConnector())?.State;

By this way we can access state by real time so any change occurs in state then it will automatically be reflected in another system.

# How .flo file save logic works?

WorkflowScrollViewer

WorkflowCanvas

Connections

CanvasModules

Connection

CanvasModule

OutputNode

InputNode

BaseModule

State

OutputNodes

InputNodes

1. Custom serialization is implemented in system where Manually WriteXML and ReadXML methods needs to be implemented.

# Working with files (caching)

Working with files is done using JdSuite.Common.FileProcessing namespace.

WorkflowFile is the main entity to work with. It contains xml representation of file in RootNode property. Workflow objects is created by WorkflowFileFactory. All file processing should be done using WorkflowFile object to avoid multiple loading the same file and reduce memory usage.

State already has WorkflowFile object in DataFile property. Just don’t forget to copy it from previous module to next one (always in CanRun() method):

this.InputNode.State.DataFile = ((OutputNode)this.InputNode.Connector).State.DataFile;

And after modifying Xml structure don’t forget to save it (if needed) using

void SaveAsXml()

or

void SaveAsXml(string fileName)

if you need to save it to another file.

# Validation

To validate existing WorkflowFile object just run the method

bool ValidateUsingSchema(Field schema, int totalNodeCount, out List<string> errors)

passing current Scema object (Field class), totalNodeCount (get from Indexer.GetTotalNodeCount()).

The errors list will contain error messages (if there will be any).

WorkflowFile object also have validation progress event to notify about validation progress.

You can use it like this:

workflowFile.OnValidationProgressChange += (s, e) => { workInfo.UpdateProgress(this, e); };

Don’t forget to unsubscribe after validation.