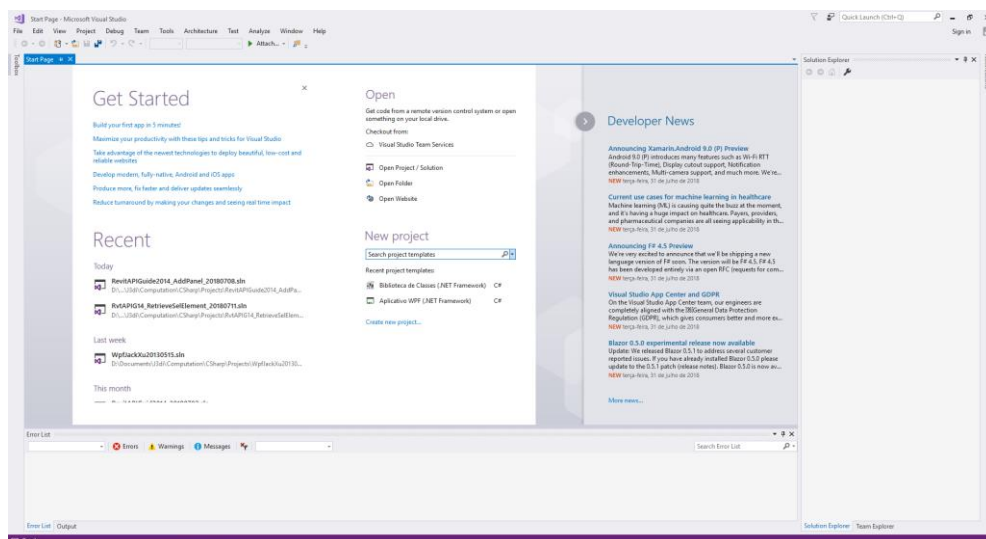


# PROGRAMAÇÃO BÁSICA DE UM ADD-IN PARA REVIT

## Visual Studio



A plataforma Visual Studio da Microsoft consiste em um ambiente de desenvolvimento integrado (IDE) onde é possível o desenvolvimento de software. É específico para o .NET Framework, onde é possível usar as linguagens Visual Basic (VB), C, C++, C# (C Sharp) e F# (F Sharp), além de outras soluções, como o desenvolvimento na área web, usando a plataforma do ASP.NET, como websites, aplicativos web, serviços web e aplicativos móveis. As linguagens que são mais usadas no Visual Studio são o VB.NET, Visual Basic.Net, e o C#, cuja pronúncia é C Sharp.

Existe uma versão do Visual Studio que pode ser instalada para uso individual, a versão Community, que pode ser adquirida para uso associado a uma conta Microsoft.

## WPF - Windows Presentation Foundation.

Uma peça muito importante em um aplicativo é sua interface. A interface é responsável por captar a entrada de dados do usuário e, após certo processamento, apresentar os resultados adequadamente. O Windows substituiu sua interface padrão, de Windows Form, que era baseado em imagens raster, matriz de pontos. O WPF, Windows Presentation Foundation, é baseado em uma

apresentação vetorial, resultando em imagens melhor definidas e apresentadas, sendo mais leve computacionalmente, dentro outros benefícios, como não distorções.

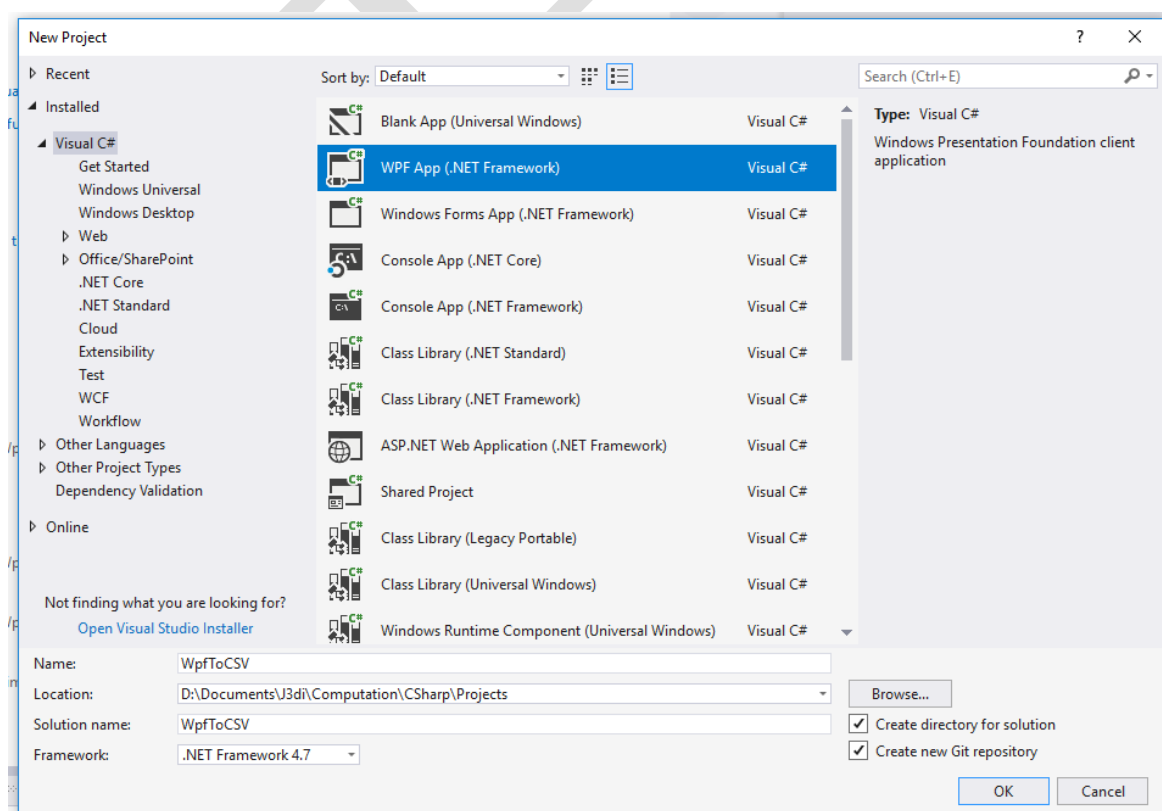
A forma de definir uma interface se dá pelo uso do XAML, pronunciado zammel em inglês ou zimel em português, e que significa eXtensible Application Markup Language. XAML é uma linguagem declarativa baseada no XML e possuiu suporte nativo para edição no Visual Studio. A interface pode ser definida arrastando-se os componentes para a janela padrão, ou pela edição textual do XAML.

## Exemplo - Projeto WPF para Gerar CSV

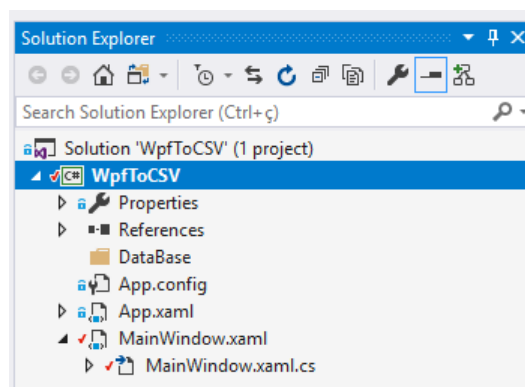
Através deste exemplo entenderemos melhor o uso desta tecnologia. Abriremos o Visual Studio e iniciaremos um novo projeto.

- Menu File – New - Project

As opções de projeto são apresentadas. Devemos optar por WPF App e optar pelo nome “WpfToCSV”.



No Solution Explorer podemos ver os arquivos criados, onde também podemos adicionar pastas para melhor organização. O arquivo “MainWindow.xaml” contém a interface do aplicativo. O arquivo “MainWindow.cs” é o arquivo da lógica do programa, que neste caso será escrita em C#. Duplo clique nestes arquivos possibilita a abertura para edição.



O foco deste trabalho não é detalhar o WPF, portanto, apresentamos o código completo sem entrar em detalhes sobre sua sintaxe. É interessante iniciar pela interface e depois promover a lógica, usando os componentes da interface. Isto dito para a codificação quando já estamos com o projeto pronto, algo que normalmente fazemos com UML.

Código XAML:

```
<Window x:Class="WpfToCSV.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WpfToCSV"
        mc:Ignorable="d"
        Title="Creating CSV" Height="170" Width="330">
    <StackPanel>
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto"/>
                <ColumnDefinition Width="Auto"/>
                <ColumnDefinition Width="Auto"/>
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"/>
            </Grid.RowDefinitions>
        </Grid>
    </StackPanel>
</Window>
```

```

        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>
    <TextBlock Text="Nível" FontWeight="Bold" Margin="5" Grid.Column="1"
Grid.Row="0"/>
    <TextBlock Text="Material" FontWeight="Bold" Margin="5" Grid.Column="2"
Grid.Row="0"/>
    <!-->
    <TextBlock Text="Janela:" FontWeight="Bold" Margin="5" Grid.Column="0"
Grid.Row="1"/>
    <TextBlock Text="Porta:" FontWeight="Bold" Margin="5" Grid.Column="0"
Grid.Row="2"/>
    <!-->
    <TextBox Name="txtJanelaNivel" Width="100" Margin="5" Grid.Column="1"
Grid.Row="1"/>
    <TextBox Name="txtJanelaMaterial" Width="100" Margin="5" Grid.Column="2"
Grid.Row="1"/>
    <!-->
    <TextBox Name="txtPortaNivel" Width="100" Margin="5" Grid.Column="1"
Grid.Row="2"/>
    <TextBox Name="txtPortaMaterial" Width="100" Margin="5" Grid.Column="2"
Grid.Row="2"/>
    </Grid>
    <StackPanel Orientation="Horizontal" HorizontalAlignment="Right">
        <Button Name="btnLimpar" Width="51" Margin="5"
Click="BtnLimpar_Click">_Limpar</Button>
        <Button Name="btnCSV" Width="51" Margin="5"
Click="BtnCSV_Click">_CSV</Button>
        <Button Name="btnExit" Width="51" Margin="5,5,37,5"
Click="BtnExit_Click">Sai_r</Button>
    </StackPanel>
</StackPanel>
</Window>

```

Código C#:

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.IO;
using Microsoft.Win32;

namespace WpfToCSV {
    /// <summary>
    /// Interaction logic for MainWindow.xaml

```

```

/// </summary>
public partial class MainWindow : Window {

    // Variáveis globais:
    string arquivoNome;
    public string ArquivoNome { get { return arquivoNome;} set { arquivoNome =
value; } }

    public MainWindow() {
        InitializeComponent();
        ArquivoNome = "";
    }

    private void BtnExit_Click(object sender, RoutedEventArgs e) {
        this.Close();
    }

    private void BtnLimpar_Click(object sender, RoutedEventArgs e) {
        txtJanelaMaterial.Text = "";
        txtJanelaNivel.Text = "";
        txtPortaMaterial.Text = "";
        txtPortaNivel.Text = "";
        // Tip: to get value as number var = Convert.ToDouble(txtField.Text)
    }

    private void BtnCSV_Click(object sender, RoutedEventArgs e) {
        try {
            string CombinedPath =
System.IO.Path.Combine(Directory.GetCurrentDirectory(), "..\\..\\DataBase");
            SaveFileDialog saveFileDialog = new SaveFileDialog {
                Title = "Salvar CSV",
                //Alternativas
                //InitialDirectory = @"..\\..\\DataBase\\",
                //InitialDirectory = @"D\\Temp\\",
                //InitialDirectory = "D\\Temp\\",
                //InitialDirectory =
System.IO.Path.Combine(System.IO.Path.GetDirectoryName(Directory.GetCurrentDirectory())
), "..\\..\\DataBase\\"),
                InitialDirectory = System.IO.Path.GetFullPath(CombinedPath),
                Filter = "Rtf documents (*.rtf)|Txt files (*.txt)|*.txt|Csv files
(*.csv)|*.csv|All files (*.*)|*.*",
                FilterIndex = 3,
                RestoreDirectory = true
            };
            saveFileDialog.ShowDialog();
            if (saveFileDialog.FileName == "") {
                MessageBox.Show("Arquivo Inválido", "Salvar Como",
MessageBoxButton.OK);
            } else {
                ArquivoNome = saveFileDialog.FileName;
                using (StreamWriter writer = new StreamWriter(ArquivoNome)) {
                    writer.WriteLine("Elemento;Nivel;Material");
                    writer.WriteLine("Porta:;{0};{1}", txtPortaNivel.Text,
txtPortaMaterial.Text);
                    writer.WriteLine("Janela:;{0};{1}", txtJanelaNivel.Text,
txtJanelaMaterial.Text);
                }
                MessageBox.Show("Arquivo Salvo com Sucesso!", "Salvar!",
MessageBoxButton.OK);
            }
        } catch (Exception ex) {

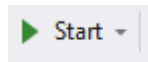
```

```

        MessageBox.Show(ex.ToString());
    }
}
}

```

O programa deve ser compilado para que um executável seja gerado, caso não seja encontrado erro grave. O botão para compilação, na parte superior sob o menu, é uma das formas para compilar o arquivo



. Devemos observar se não obtivemos erro na barra de status



. Caso tudo ocorra sem erro, a janela criada como interface é exibida e o programa pode ser testado.

## Revit API

Desenvolvimento de plug-ins com a API, Application Programming Interface, do Revit, utilizando a linguagem C#, do .Net, no Visual Studio.

Utilizaremos, como recurso, o “Revit 2019” e o “Visual Studio 2017” para desenvolver nossos exemplos. Estas duas plataformas podem ser adquiridas na versão estudante, para uso educacional.

O Visual Studio deve ser otimizado para trabalhar com C#, linguagem computacional, e utilizar o Inglês como linguagem, para melhor acompanhamento dos exemplos.

Nem todos os passos definidos são indispensáveis para o funcionamento de todos os exemplos, mas sua definição é importante para uma boa prática de programação ou sendo mesmo indispensáveis para exemplos específicos.

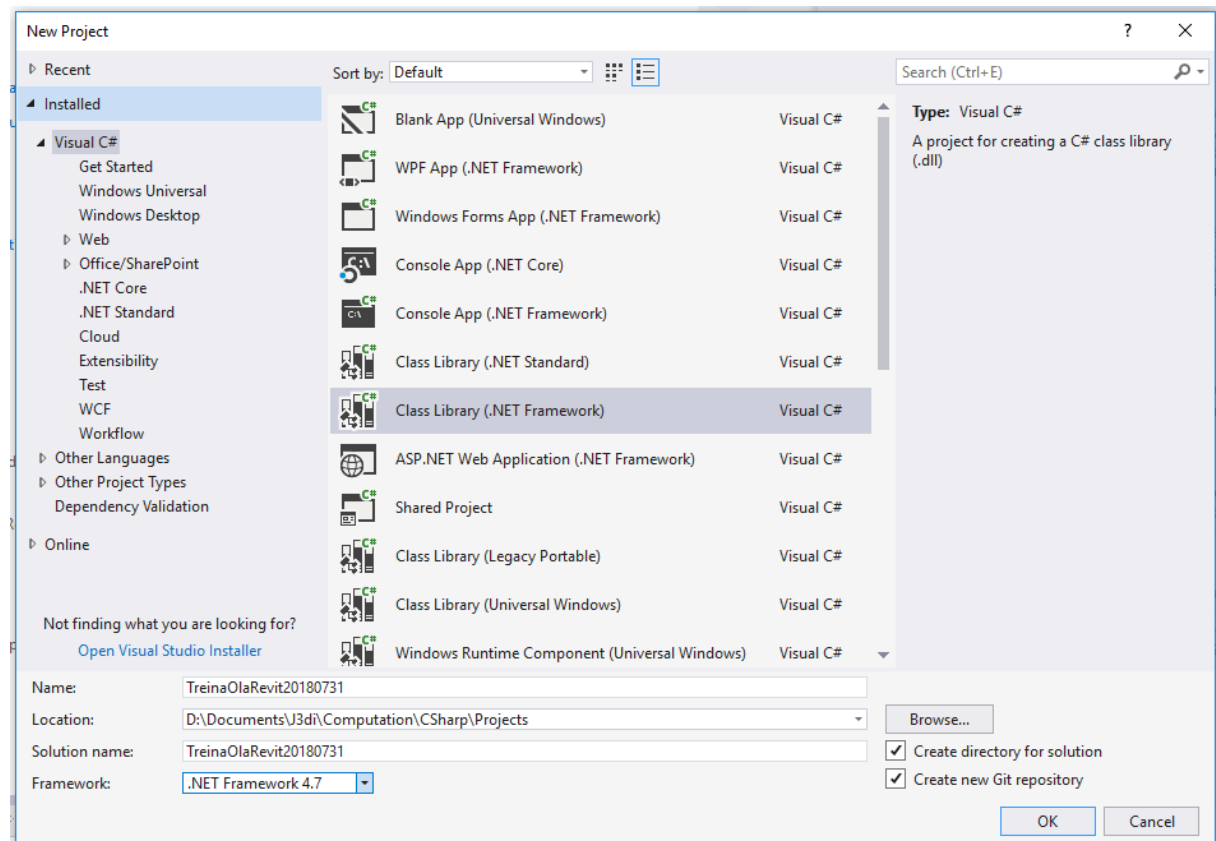
Usaremos exemplos práticos para entender os conceitos necessários para o uso da API do Revit, com a linguagem C#, na plataforma Visual Studio.

É possível encontrar ajuda específica no SDK, software development kit, da API, disponível com a instalação do pacote do Revit, sendo o instalador do SDK, que fica disponível apenas nos arquivos de instalação, encontrado em “C:\Autodesk\Revit\_2019\_G1\_Win\_64bit\_dlm\Utilities\SDK\RevitSDK.exe”. O SDK traz ajuda, bem como exemplos e arquivos de suporte para o desenvolvimento utilizando a API do Revit.

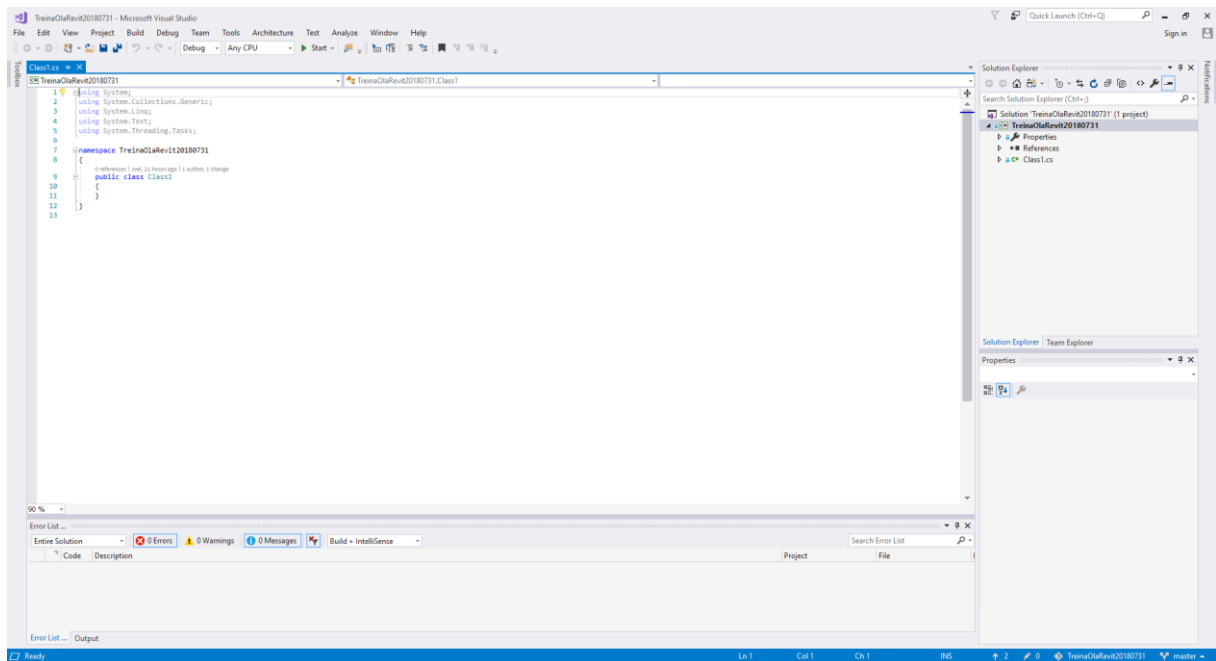
## Exemplo - Projeto "Olá Revit!"

Existem vários tipos de projetos que podem ser criados com o Visual Studio. Utilizaremos Class Library para criar um plug-in para o Revit.

- Menu File - New - Project



Uma janela para as definições do projeto possibilita escolher os padrões a serem utilizados. Optaremos por "Visual C#" para linguagem, "Class Library" para gerar uma dll, dynamic link library. O nome do projeto será "TreinaOlaRevit20180731". Voltamos para a janela principal, e o projeto criado é exibido.



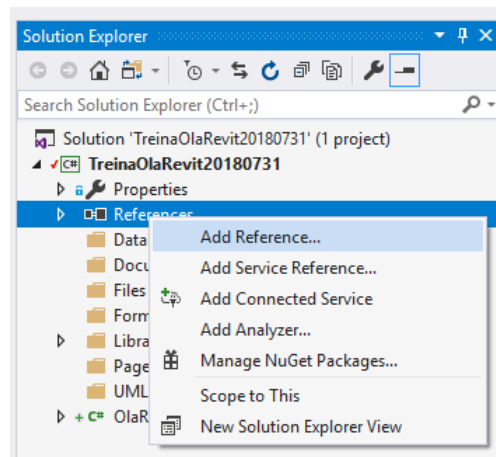
O arquivo “Class1.cs” contém nosso código C#. Iremos renomear esse arquivo para “OlaRevit.cs”. Para isso, deveremos clicar com o botão direito sobre o arquivo e escolher a opção “Renomear”. O Visual Studio deve perguntar se desejamos atualizar as referências a este nome, e devemos escolher “Sim”.

Não se deve reinventar a roda na programação. Faremos uso de bibliotecas de códigos, que contém o código necessário para reuso e montagem da nossa lógica. A API do Revit está contida em dll, dynamic link library, que são bibliotecas com a descrição do código necessário para interagir com o Revit. Precisamos adicionar referência às bibliotecas que usaremos.

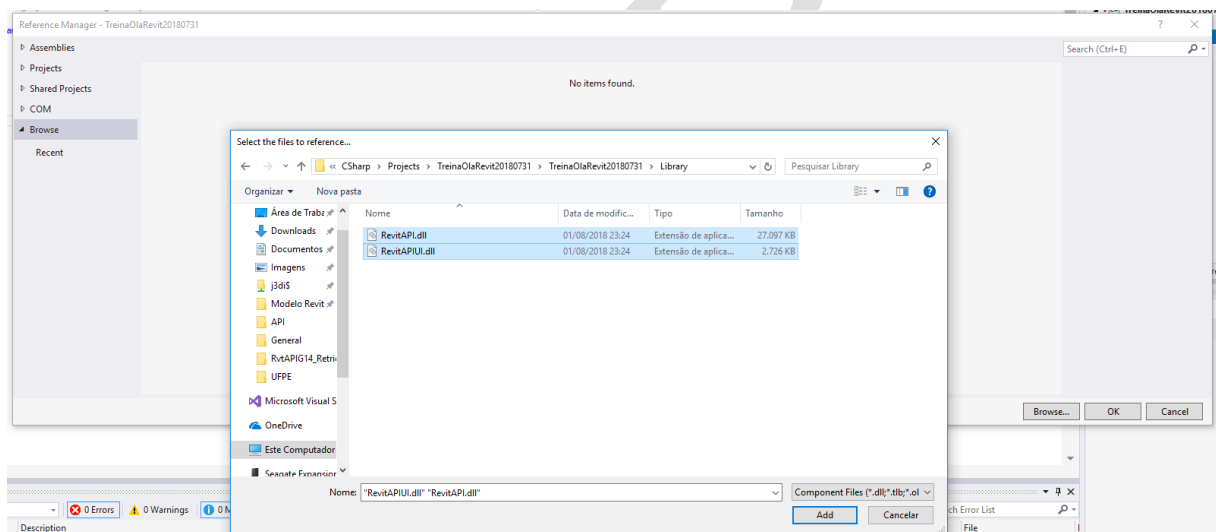
Primeiramente adicionaremos as duas dll necessárias para usar a API do Revit, sendo “RevitAPI.dll” e “RevitAPIUI.dll”. Eles estão no diretório onde nosso Revit está instalado, possivelmente “C:\Program Files\Autodesk\Revit 2019”.

Clicamos com o botão direito do mouse em References, da janela Solution Explorer. Selecionamos “Add Reference...”

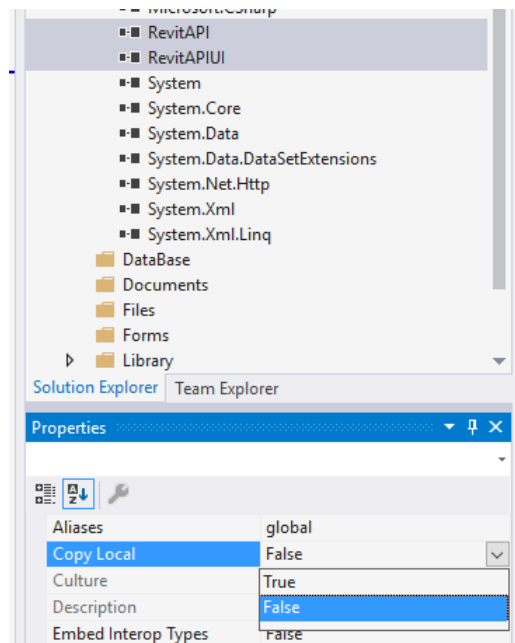




Use a opção de “Browser” para navegar até o local onde estão as dlls e adicione. Uma boa opção pode ser antes copiá-las para a pasta “Library” do projeto, isto porque mudanças de instalação do Revit podem modificar ou perder o local original destas bibliotecas da API.



Altere no Solution Explorer o parâmetro “Copy Local” para falso. Isto vai evitar que uma cópia da biblioteca seja copiada para a pasta sempre que o programa for compilado.



Agora precisamos fazer algumas modificações em nosso código C#, arquivo “\*OlaRevit.cs”, de tal forma que possa atender às exigências da API.

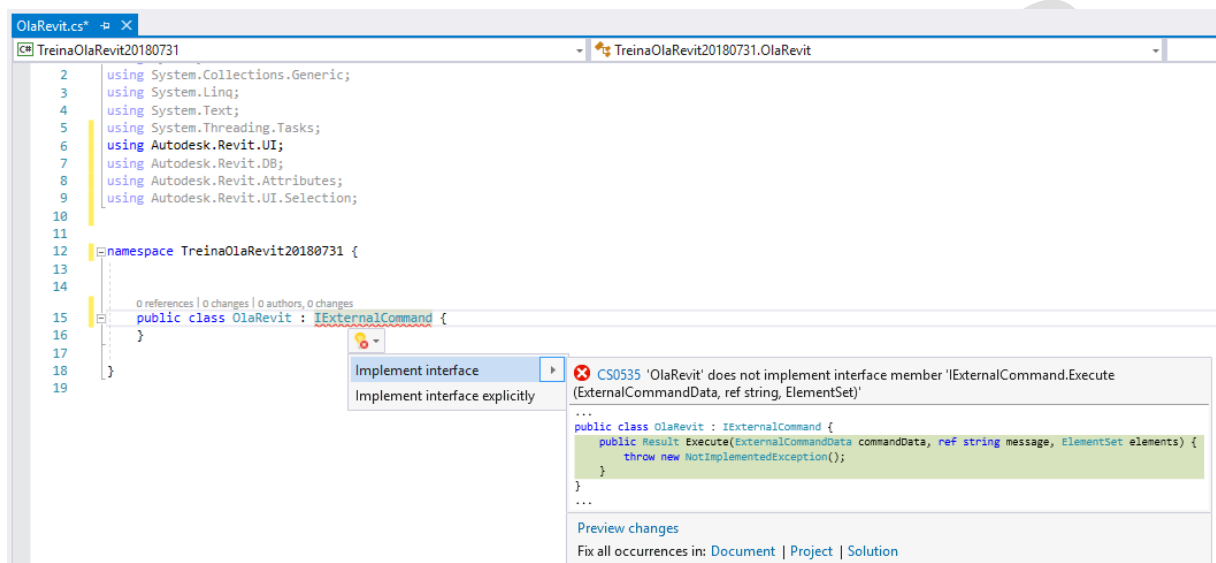
Adicionamos as bibliotecas que o código fará uso:

```
using Autodesk.Revit.ApplicationServices;
using Autodesk.Revit.UI;
using Autodesk.Revit.DB;
using Autodesk.Revit.Attributes;
using Autodesk.Revit.UI.Selection;
```

Adicionamos a directive que instrui sobre a transação. Atentar que nas atualizações a opção Automática não é mais suportada:

```
[Transaction(TransactionMode.Manual)]
```

Precisamos implementar a interface `IExternalCommand`, e neste caso faremos uso de uma funcionalidade bem interessante do Visual Studio. Adicione o código “: `IExternalCommand`” logo após à nossa definição da classe. O Visual Studio irá nos possibilitar definir a implementação desta interface, bastando usar a opção que irá aparecer sob a definição, um ícone de uma lâmpada com a seta que selecionaremos, e que aparece quando deixamos a seleção sobre o código recém adicionado, conforme imagem a seguir.



Um método, “Execute”, será adicionado, e, uma vez que ele retorna “Result”, precisamos dar esta saída ao nosso código. Podemos comentar o código que lança a exceção de não implementação do código com “//”. Usaremos a saída dentro de uma estrutura “try ... catch”, para tratar exceções.

Este nosso código apenas exibirá uma mensagem no Revit com uso de janela de texto normal, o que pode ser feito com o código “`TaskDialog.Show("Revit", "Olá Revit!");`”.

Nosso código completo ficou:

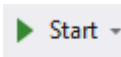
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Autodesk.Revit.UI;
using Autodesk.Revit.DB;
```

```
using Autodesk.Revit.Attributes;
using Autodesk.Revit.UI.Selection;

namespace TreinaOlaRevit20180731 { // NameSpace.

    [Transaction(TransactionMode.Manual)]
    public class OlaRevit : IExternalCommand { // Classe que irá dar origem à dll para
    uso no Revit.
        public Result Execute(ExternalCommandData commandData, ref string message,
        ElementSet elements) { // Método de entrada que implemtna a interface necessária da
        API.
            try {
                TaskDialog.Show("Revit", "Olá Revit!"); // Caixa de diálogo que será
                executada se tudo der certo.
                return Result.Succeeded; // Retorno que tudo ocorreu bem.
            } catch (Exception ex) { // Tratamento de exceção.
                message = ex.Message;
                TaskDialog.Show("Error!", message);
                return Result.Failed; // Retorno de erro.
                //throw; // Código a ser comentado.
            }
            //throw new NotImplementedException(); // Código a ser comentado.
        }
    }
}
```

Devemos então compilar o arquivo

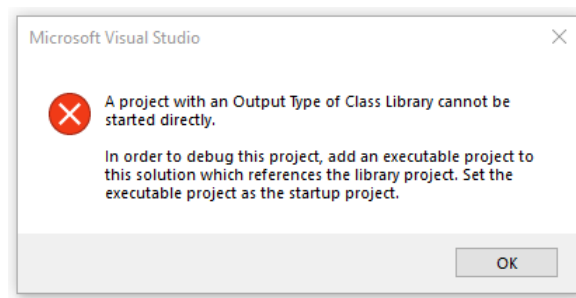


. E observar se não obtivemos erro na barra de status

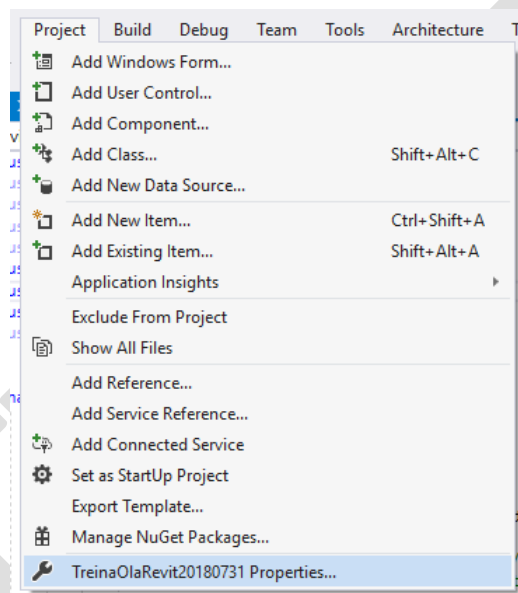
0 Errors | 0 of 2 Warnings

. Uma tela de advertência pode ser apresentada, visto que não estamos criando um programa, e não atribuímos a um aplicativo, o que pode ser configurado para usar o Revit, mas ainda não o faremos aqui.

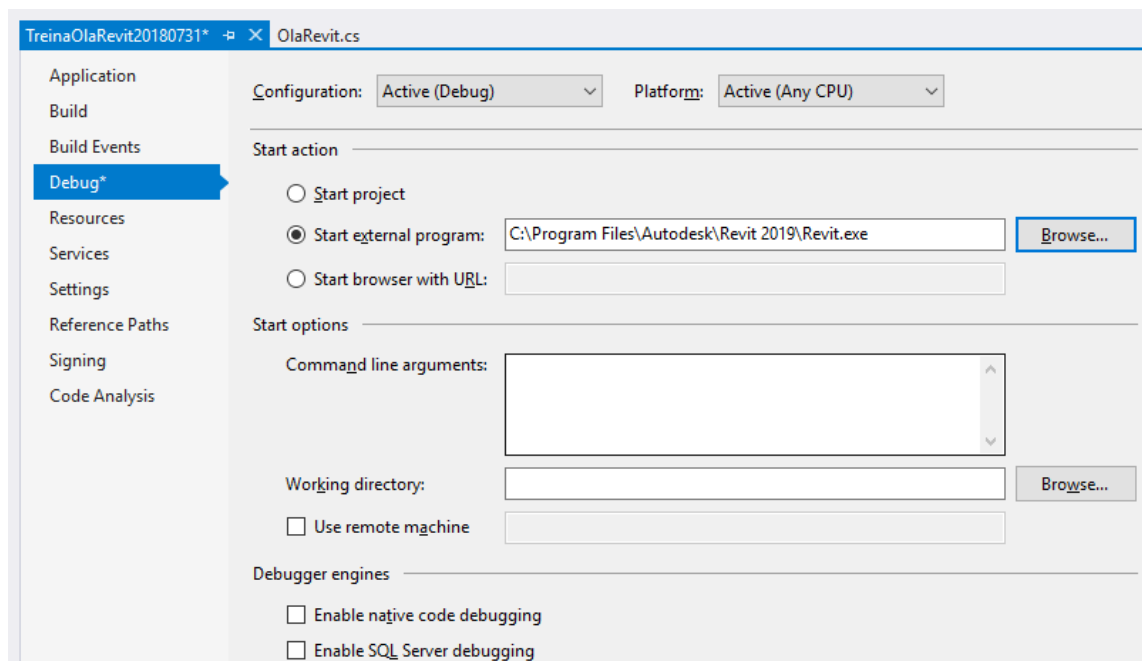
Quando executamos o projeto desta forma, teremos a advertência de que a saída não pode ser executada diretamente, visto ser uma dll, e que precisaríamos de um programa externo para isso. Para testarmos imediatamente a biblioteca, podemos então configurar o Revit como o programa para testar a saída.



O programa de saída deve ser configurado a partir do menu “Project”, submenu “Properties...”, que nos remeterá à tela de configuração.



Devemos então a aba “Debug”, e então optar por “Start external program:”. O botão “Browser” nos permite selecionar o programa que será executado quando compilarmos o projeto, no caso o Revit. Navegue até a pasta onde o Revit foi instalado e selecione o executável “Revit.exe”. Um provável caminho será “C:\Program Files\Autodesk\Revit 2019”.



Sempre que compilarmos o projeto o Revit será inicializado, caso não tenhamos algum erro a depurar. A biblioteca, o arquivo dll, será criado na pasta “bin\Debug\” do nosso projeto. Esta é a dll que consiste em nosso plug-in e que será apontada para que o Revit possa fazer uso do nosso código, algo que é feito com um manifesto.

## Elaboração de Manifesto que Habilita o Plugin

O Revit, ao iniciar, checará se existe um arquivo, que denominamos manifesto, na sua pasta padrão de plug-ins. Caso exista um arquivo nesta pasta, o Revit irá interpretar. Esta é a forma de permitirmos o uso de plug-ins no Revit. Caso tudo ocorra bem com a interpretação, seremos avisados do seu carregamento, e teremos que optar entre não carregar a dll, até porque está é uma possibilidade de entrada para vírus, carregar apenas uma vez, ou sempre carregar. Quando optamos por carregar sempre, o Revit não nos dará avisos de carregamento deste plug-ins nas próximas inicializações. Um exemplo de manifesto está na área de download da eTlipse, site <https://www.etlipse.com/>, que se refere ao uso do AddInManager, um aplicativo que possibilita carregar diretamente plug-ins do formato .dll.

Criaremos o arquivo “TreinaOlaRevit20180731.addin”, que conterá as informações necessárias para interpretação na inicialização do Revit. Este arquivo precisará estar na pasta de plug-ins do Revit,

em geral, "C:\Users\[XXXX]\AppData\Roaming\Autodesk\Revit\Addins\2020\", onde "[XXXX]" é o nome do usuário. A pasta AppData pode ser encontrada mais facilmente pelo atalho "%AppData%".

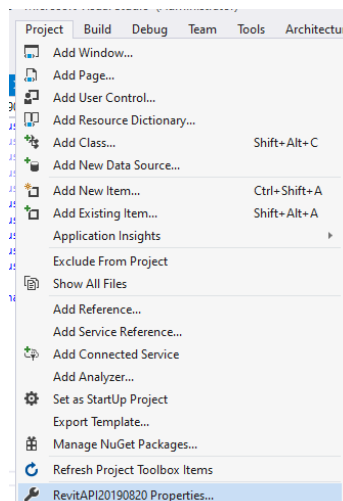
Nosso arquivo ficará conforme modelo abaixo. Os parênteses usamos aqui apenas para comentar, mas obviamente não devem estar contidos em nosso manifesto.

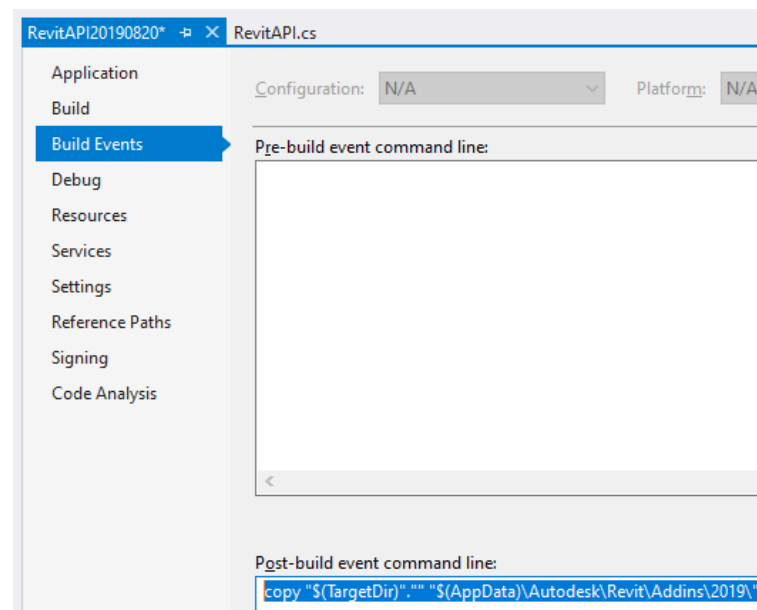
```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<RevitAddIns>
  <AddIn Type="Command">
    <Name>SampleApplication</Name>
    <Assembly>C:\Temp\RevitPlugins\ TreinaOlaRevit20180731.dll</Assembly> (Arquivo e
Local onde copiaremos nossa dll, resultado do arquivo compilado)
    <AddInId>BC05297E-11B0-4CCC-A0F4-35AFC0DD12E7</AddInId> (Código único de Id do
Plugin, que pode ser gerado no Visual Studio)
    <FullClassName> TreinaOlaRevit20180731.0laRevit</FullClassName> (NameSpace e nome
da classe que criamos)
    <VendorId>ETlipse</VendorId>
    <VendorDescription>Edson Andrade, Rogerio Lima, Joel Diniz</VendorDescription>
  </AddIn>
</RevitAddIns>
```

O Id do plug-in deve ser único, e pode ser gerado no próprio Visual Studio, em Tools > Create GUID > 5. Este é um número gerado com algoritmo especial para ser improvável a coincidência.

É possível automatizar a cópia do Manifesto para pasta adequada do Revit, que entende os plug-ins a serem executados. Devemos acessar, no Visual Studio, o menu **Project**, submenu **Properties**. Na janela de propriedades do projeto, na aba Built Events, caixa Post-build, colocamos a directiva abaixo, que dependerá do Revit em uso, no exemplo o 2019.

copy "\$(TargetDir)". "" "\$(AppData)\Autodesk\Revit\Addins\2019\"



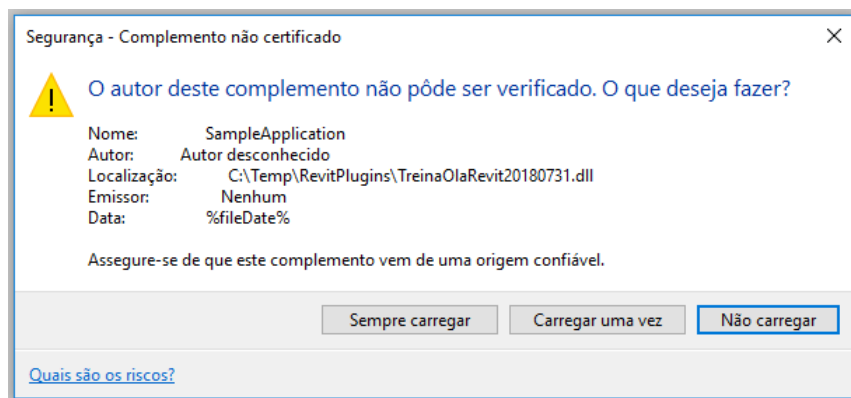


Conforme citado, o Revit irá adicionar como plugin os arquivos processados na pasta citada de plugins. Estes arquivos basicamente consistem no addin, que precisa estar corretamente configurado apontando para a *dll*, e a *dll* em si que irá conter o código do plugin. Caso opte por sempre abrir o plugin, no seu primeiro acesso, estes arquivos irão permanecer na pasta padrão, e não será mais lançada a pergunta sobre seu processamento, o que passará a ocorrer automaticamente. Na fase de teste é portanto interessante optar pelo carregamento temporário. Caso precise desinstalar seu plugin que optou por sempre carregar basta deletar ou mover os arquivos da referida pasta.

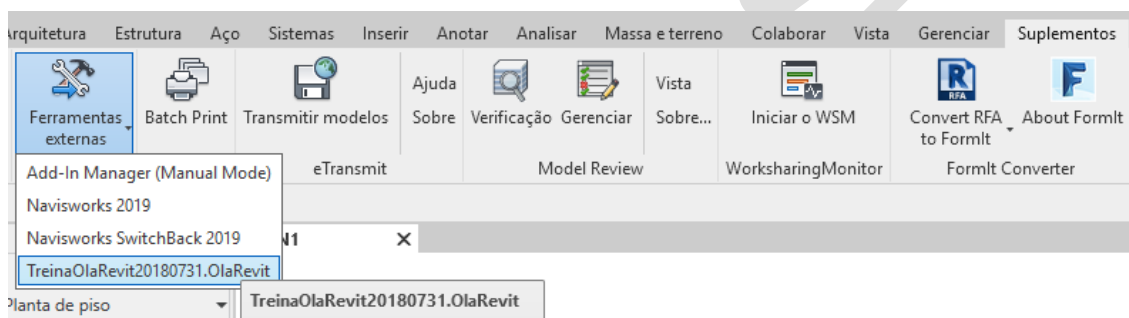
## Executando o plug-in

Inicializamos o Revit, e obteremos o aviso que descrevemos. Para teste pode ser interessante ainda não optar por aceitar sempre executar o plug-in.

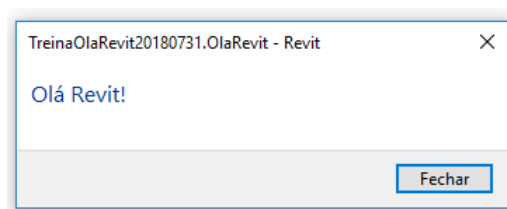




O plug-in que criamos estará disponível em Suplementos > Ferramentas externas. Este será o padrão, mas é possível criar novos itens e menus.



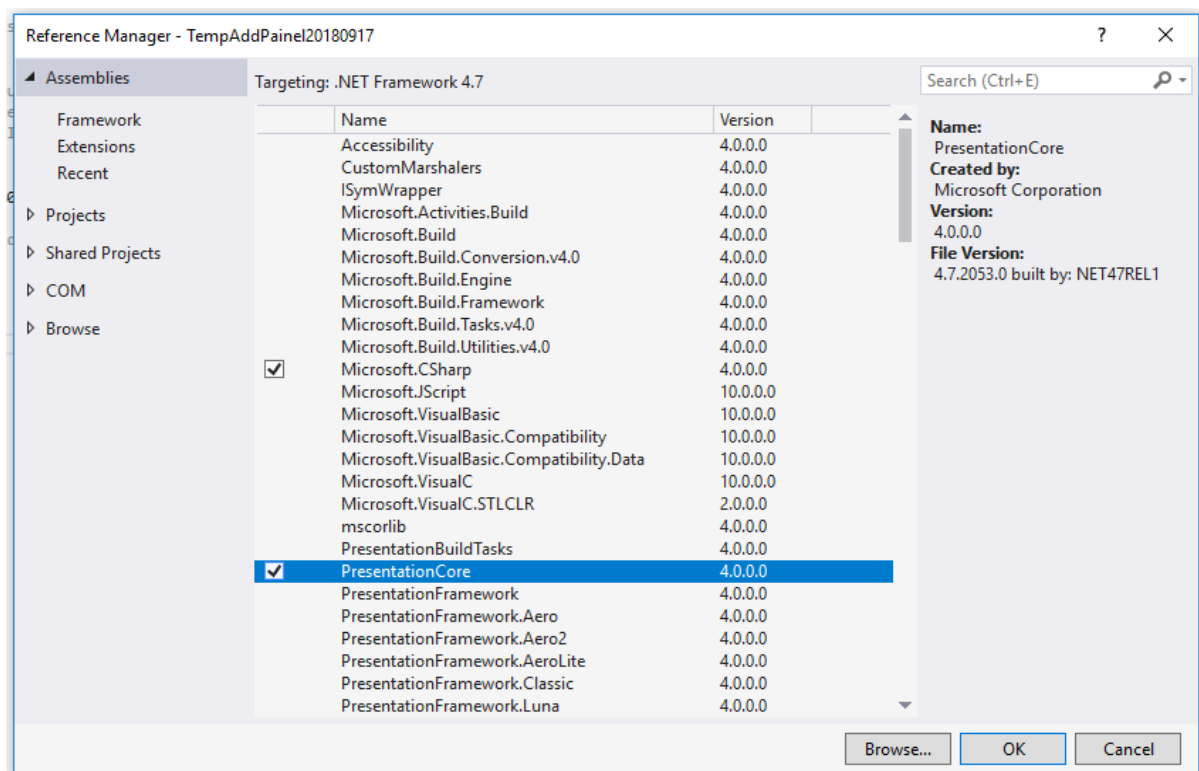
Conforme pretendido, a execução, com o clique no item, nos remeterá à tela de diálogo com texto que programamos.

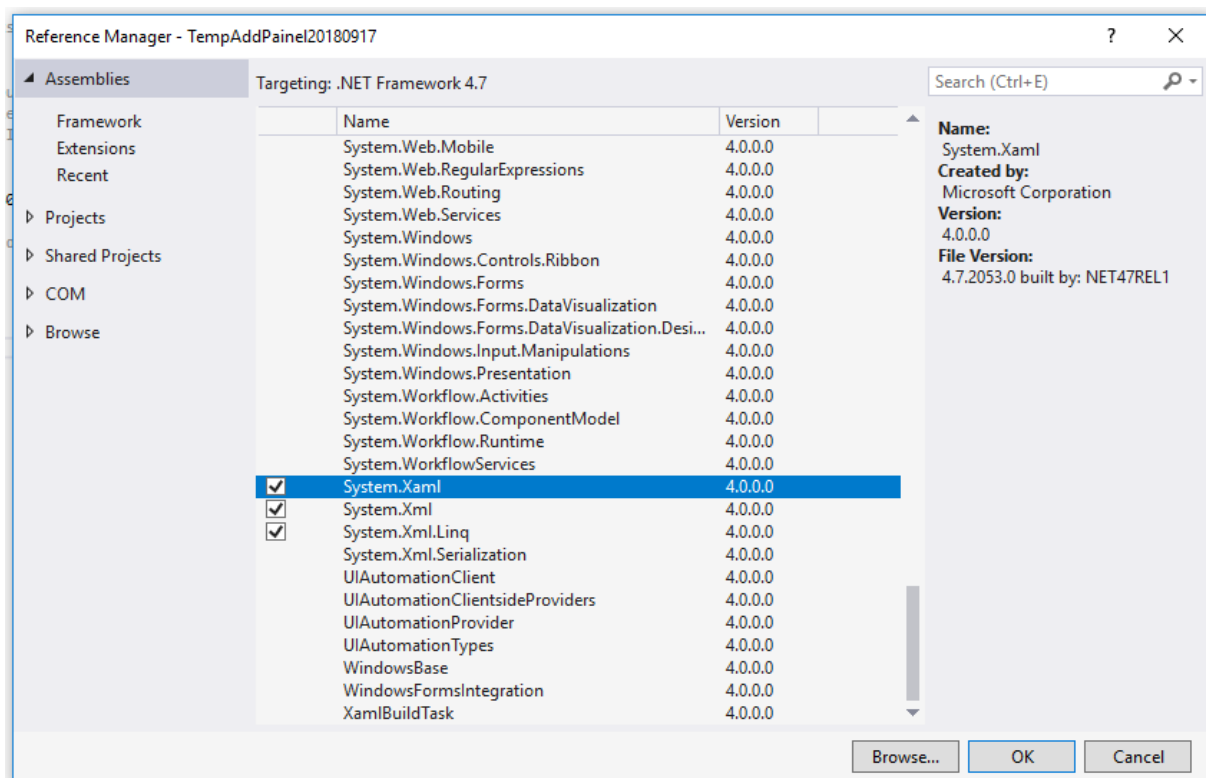


## Personalizar Ribbon Painel

Para criação de plugin é possível personalizar o painel Ribbon do Revit e configurar ícones que direcionem o usuário aos plug-ins personalizados.

Criamos um projeto no Visual Studio, de nome “AddPanel”, conforme no primeiro exemplo. Desta vez precisaremos adicionar mais referências. Devemos adicionar referência a “PresentationCore”. O mesmo deve ser feito para adicionar “System.Xaml”.





Renomeamos “Class1.cs” para “CsAddPanel.cs”, a partir de Solution Explorer, botão direito do mouse sobre o arquivo, em seguida opção renomear. Abrimos o arquivo “CsAddPanel.cs” para editá-lo conforme abaixo. Diferente da primeira aplicação, que era baseada em Command, está é baseada em Application, e contém dois métodos abstratos, OnStartup() e OnShutdown(). É preciso definir uma imagem de ícone e usar seu caminho no código, bem como usar o caminho da dll, conforme código a seguir. O botão criado deve apontar para a dll que se pretende executar quando este for acionado no Revit. O arquivo de manifesto deve ser criado de forma semelhante ao feito anteriormente, desta vez com tipo “Application”.

O código completo fica:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Autodesk.Revit.UI;
using Autodesk.Revit.DB;
using Autodesk.Revit.Attributes;
using Autodesk.Revit.UI.Selection;
using System.Windows.Media.Imaging;

namespace RevitAPIGuide2014_AddPanel_20180708{
```

```
[Transaction(TransactionMode.Manual)]
public class CsAddPanel : Autodesk.Revit.UI.IExternalApplication {

    Result IExternalApplication.OnShutdown(UIControlledApplication application) {

        return Result.Succeeded;
    }

    Result IExternalApplication.OnStartup(UIControlledApplication application) {

        // Add new ribbon panel
        RibbonPanel ribbonPanel = application.CreateRibbonPanel("NewRibbonPanel");

        // Create a push button in the ribbon panel "NewRibbonPanel"
        // the add-in application "HelloWorld" will be triggered when button is
pushed

        PushButton pushButton = ribbonPanel.AddItem(new
PushButtonData("HelloWorld", "HelloWorld",
@"C:\Temp\RevitPlugins\RevitAPIGuid2014_20180702.dll",
"RevitAPIGuid2014_20180702.HelloWorld")) as PushButton;

        // Set the large image shown on button
Uri uriImage = new Uri(@"D:\Temp\Temp_32x32.png");
BitmapImage largeImage = new BitmapImage(uriImage);
pushButton.LargeImage = largeImage;

        return Result.Succeeded;
    }
}
}
```

O arquivo de manifesto será:

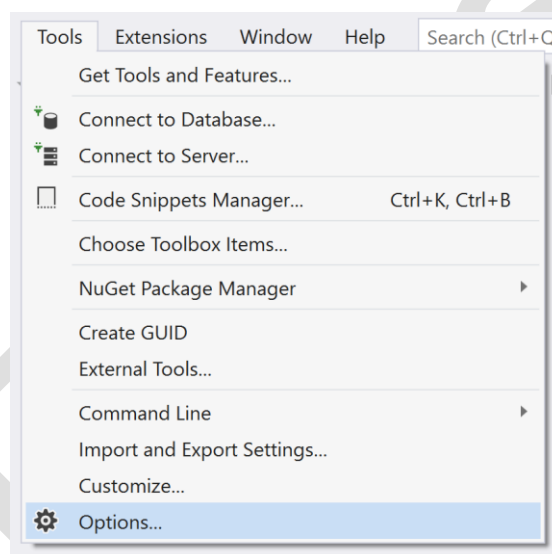
```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<RevitAddIns>
  <AddIn Type="Application">
    <Name>SampleApplication</Name>
    <Assembly>C:\Temp\RevitPlugins\RevitAPIGuide2014_AddPanel_20180708.dll</Assembly>
    <AddInId>BC05297E-11B0-4CCC-A0F4-35AFC0DD12E7</AddInId>
    <FullClassName>RevitAPIGuide2014_AddPanel_20180708.CsAddPanel</FullClassName>
    <VendorId>ETlipse</VendorId>
    <VendorDescription>Edson Andrade, Rogerio Lima, Joel Diniz</VendorDescription>
  </AddIn>
</RevitAddIns>
```

## Revit API com WPF - Template eTlipse.

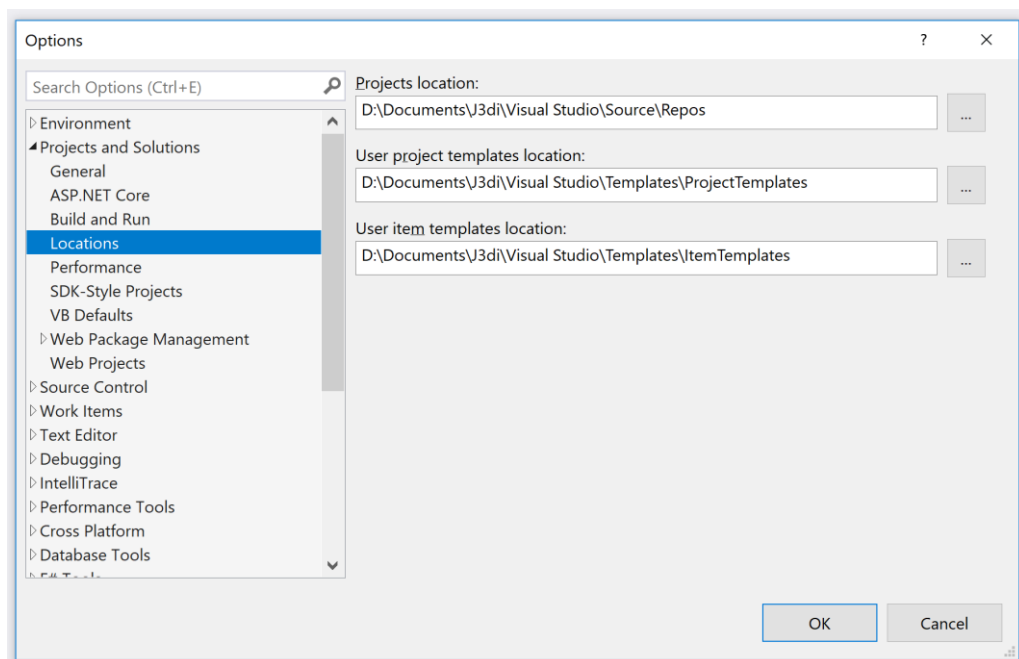
Algumas configurações são necessárias para fazer uso de interface WPF com a API do Revit, algumas bem específicas e não tão usuais. A eTlipse disponibiliza um template específico para esta necessidade, facilitando o uso deste importante recurso da plataforma .Net.

O template da eTlipse pode ser baixado na página da eTlipse para ser utilizado no Visual Studio e facilitar o uso e configuração de aplicação WPF utilizando a API do Revit. O arquivo do template consiste em um arquivo comprimido em formato .zip que deve ser colocado no diretório apropriado. Importante verificar o diretório padrão utilizado pela versão em uso do Visual Studio.

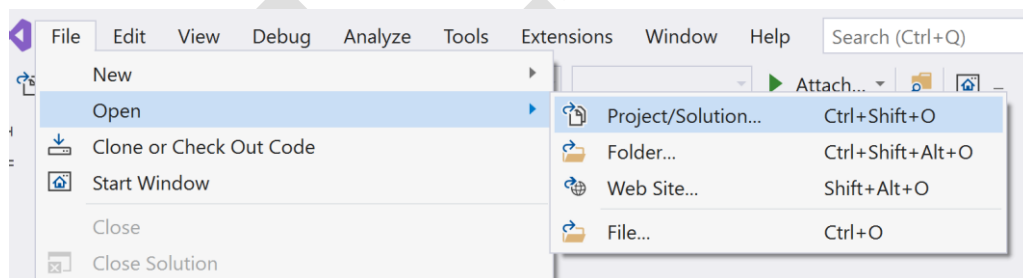
A configuração do caminho utilizado para carregar os templates está registrada no menu de configuração. Utilize a opção “Options...” do menu “Tools”.



Na janela de opções devemos escolher o item “Projects e Solutions”, e, como subtópico desta opção, teremos o item “Locations”. Será exibida as opções de caminho em “Projects Location” nesta mesma janela, onde devemos observar o item “User Project templates location”.




O arquivo de configuração deverá ser colocado dentro desta página para ser carregado como template pelo Visual Studio. Feita a cópia para a pasta correta, teremos a possibilidade de criar um novo projeto baseado no template escolhido. Um novo projeto podemos iniciar a partir do item “Projeto” da submenu “New” do menu “File”.





Podemos observar que o Visual Stúdio irá apresentar o template da eTlipse como uma das opções para novo projeto, conforme figura a seguir. Precisamos selecionar a opção exibida “TL Revit 2020 WPF Addin” e clicar em “Next”.


## Create a new project

All languages All platforms All project types

 WPF App (.NET Framework) C#

 TL Revit 2020 WPF Addin  
eTlipse Revit 2020 Add-In WPF project for an application with multiple commands in a modeless dialog.  
C# WPF Revit Add-In

 WPF Library (.NET Core)  
Windows Presentation Foundation client application

 Revit Extension WPF  
Revit Extension WPF

Not finding what you're looking for?  
[Install more tools and features](#)

Next

Na tela que segue deveremos escolher o nome do projeto e da solução que iremos construir, bem como o caminho onde armazenaremos nossa solução. Escolhidas as opções, clicamos em “Create” para criar nossa solução.

## Configure your new project

TL Revit 2020 WPF Addin C# WPF Revit Add-In

Project name

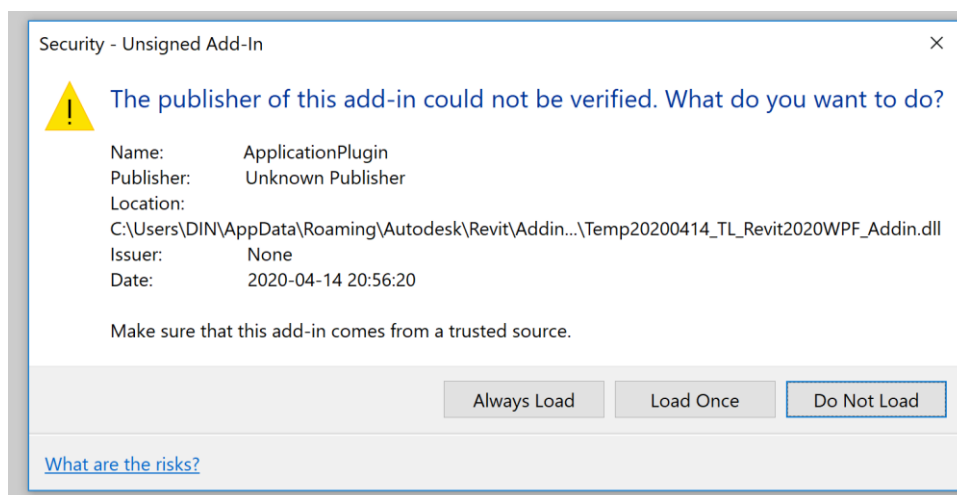
Location  
 ...

Solution name ⓘ

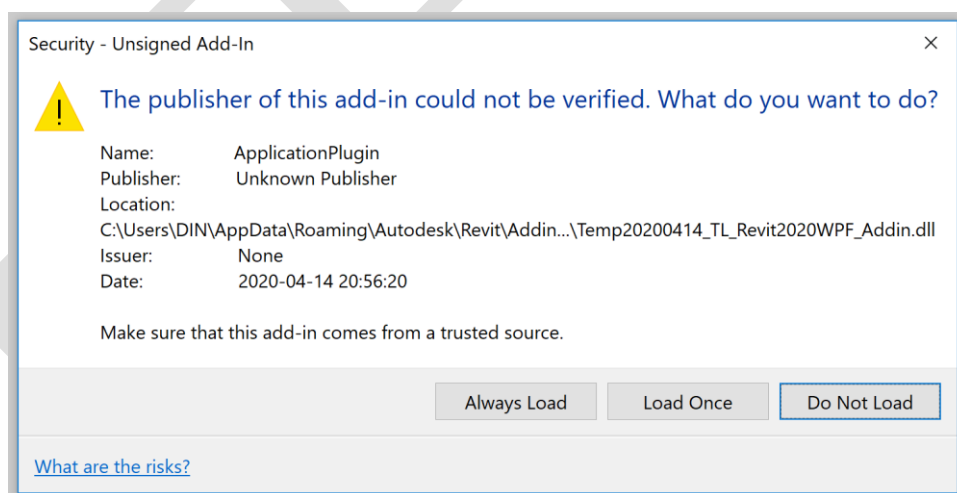
☐ Place solution and project in the same directory

Back Create

O projeto está criado e será aberto para edição. Este template traz um exemplo a partir do qual é possível fazer as devidas personalizações para a solução que desejar criar. O código é amplamente comentado para que possa facilitar sua edição.



Uma vez que tenha construído sua aplicação sobre o template, poderá então compilar sua solução. O projeto irá abrir o Revit e lançar sua solução. Importante notar que, por proteção do Revit, será exibida a tela inicial para autorizar o uso da aplicação, que poderá ser apenas para uma vez, para sempre, ou pode não permitir o carregamento. Finalmente pode abrir um projeto no Revit e a solução criada estará no menu que terá o nome configurado na solução.





O ícone projetado estará disponível para lançar sua solução. A janela projetada será lançada ao clicar no ícone do seu aplicativo.

