



TMcraft

Setup Tutorial

Basic Development

Original Instructions

This Manual contains information of the Techman Robot product series (hereinafter referred to as the TM AI Cobot). The information contained herein is the property of Techman Robot Inc. (hereinafter referred to as the Corporation). No part of this publication may be reproduced or copied in any way, shape or form without prior authorization from the Corporation. No information contained herein shall be considered an offer or commitment. It may be subject to change without notice. This Manual will be reviewed periodically. The Corporation will not be liable for any error or omission.

 and  logo are registered trademark of TECHMAN ROBOT INC. in Taiwan and other countries and the company reserves the ownership of this manual and its copy and its copyrights.

 **TECHMAN ROBOT INC.**

Table of Contents

Revision History	3
1. Introduction	4
2. Prerequisites	7
3. TMcraft Setup Program Structure	8
3.1 Brief Explanation of TMcraft API	8
3.2 Program structure of a TMcraft Setup	10
4. Start Programming a TMcraft Setup	11
4.1 System Design	11
4.2 Create a WPF Application Project	13
4.3 Source Code	15
4.3.1 MainPage.xaml	15
4.3.2 MainPage.xaml.cs	15
4.4 Compile and Test UI	20
5. Build TMcraft Setup Package	24
6. Installation Code and Checksum	28
7. Use TMcraft Setup on TMflow	29
8. Dos & Don' ts	32

Revision History

Revision	Date	Description
1.00	2024-06-22	Original release

1. Introduction

TMcraft Setup, developed in C#/WPF, is a custom UI tailored for flow projects. Users can employ TMcraft Setup in the following scenario:

- As a control panel or the setting page of a device. For example, in a gripper setup, users can manipulate the device and set parameters as run distance of the gripper for the project. Similarly, in a lifting platform setup, users can perform the calibration of the device.
- As an application setup. Users can configure the devices, set the application parameters, and define the initialization of the project by authoring scripts to the Start Node.

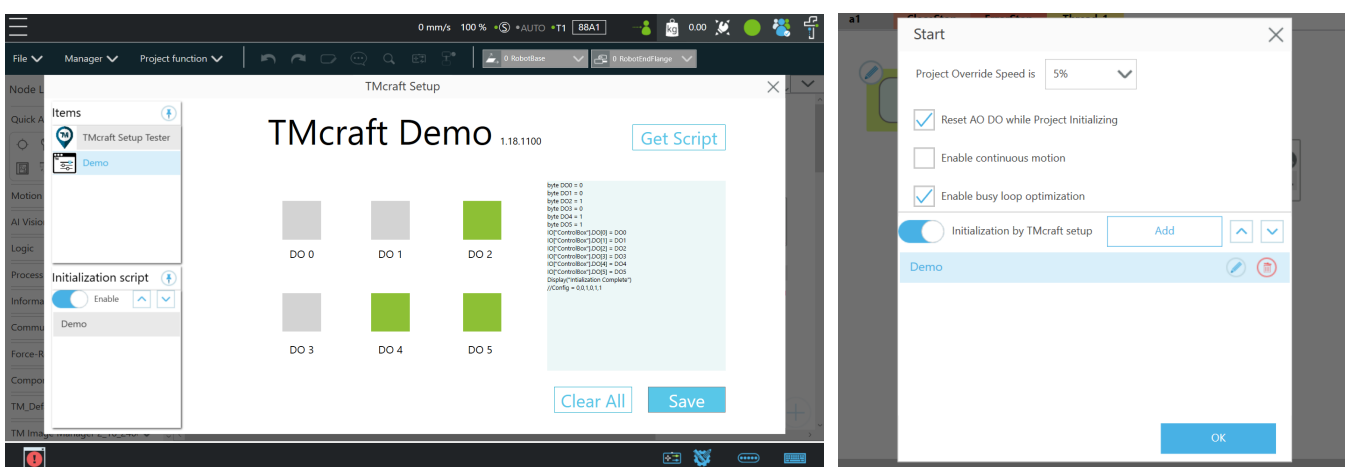


Figure 1: A Sample of TMcraft Setup

TMcraft Setup uses TMcraft (Setup) API to interact with TMflow, allowing users to manage inputs, outputs, and variables. Once configured, TMcraft Setup creates an initialization associated script that executes when the project starts.

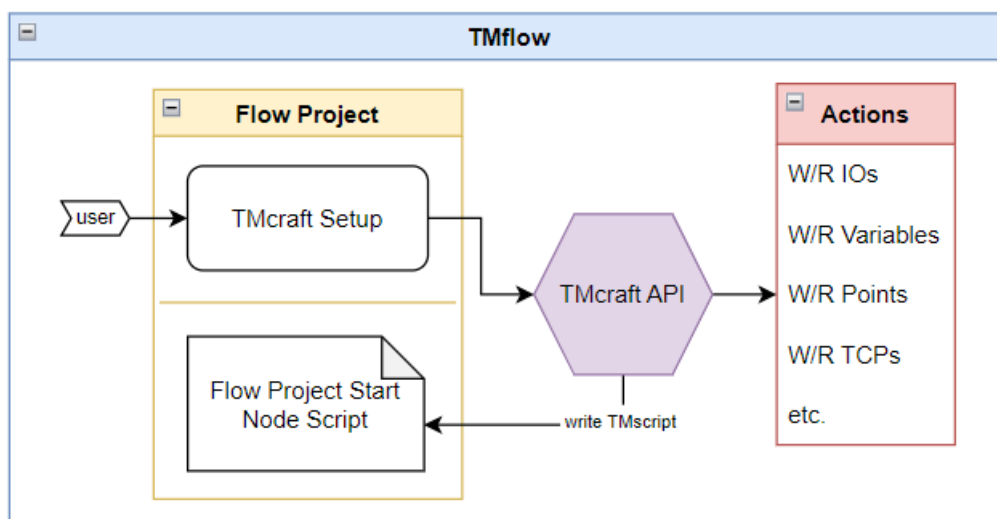


Figure 2: System architecture of TMcraft Setup within TMflow

Developers need to, at first, create a TMcraft Setup as a User Control Library in the form of a dll file, not an exe file. Then, using the TMcraft Packer from the TMcraft Development Kit, they should compile the User Control Library into an executable file and package it into a zip file with all the resource files from the source folder. Finally, they should import this TMcraft Setup zip into TMflow.

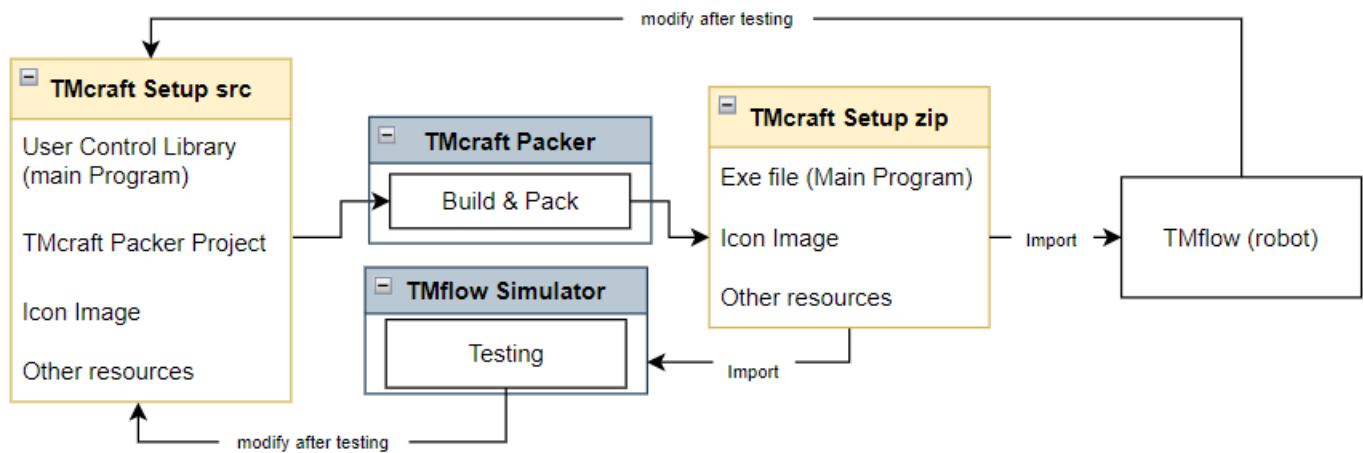


Figure 3: Development Process of a TMcraft Setup

This manual briefly explains the framework of a TMcraft Setup Program and outlines all TMcraft Setup API functions. Note that this manual may not cover all enums and additional classes in the TMcraft.dll, but the ones most relevant to the TMcraft Setup.

To determine whether the TMcraft Setup is the right fit to realize concepts of users, consider these questions:

Are users familiar with script programming?

As built with C#, developers need C# skills to develop the TMcraft Setup.

Does the application require an initialization?

Some applications might need multiple steps before production, like initializing the gripper, setting up I/O status, and verifying the connection with the SCADA system. A user-friendly interface to set up these initial processes would be helpful.

This tutorial organizes the content based on TMcraft.dll 1.18.1400 and TMflow 2.18.

- Section 2: prerequisites for developing a TMcraft Setup
- Section 3: the concept and features of the TMcraft API and TMcraft Setup Program
- Section 4: the process of how to develop a simple TMcraft Setup
- Section 5: how to complete building a TMcraft Setup Package
- Section 6: the concept of Unique Identifier, GUID
- Section 7: how to import and use TMcraft Setup on TMflow
- Section 8: the dos and don'ts

2. Prerequisites

To develop TMcraft Setup, developers should be capable of:

- Having Basic knowledge of programming with C#
- Using WPF to build UI. Understanding of the User Control concept is preferred.
- Using TM script the Programming Language

Software requirements,

- An integrated development environment for C# and WPF (such as Microsoft Visual Studio 2022)
- TMflow 2.18 or above
- .NET 6.0 and .NET SDK
- TMcraft Packer for building and packaging the TMcraft item

3. TMcraft Setup Program Structure

3.1 Brief Explanation of TMcraft API

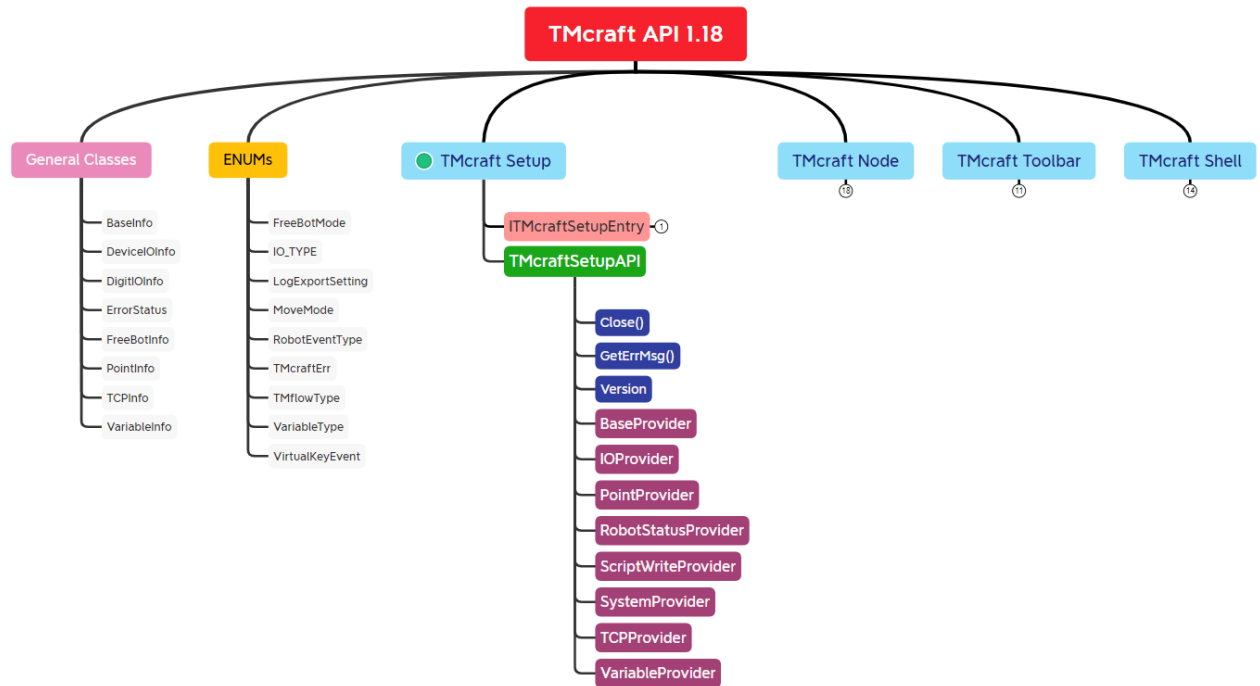


Figure 4: TMcraft API Architecture

TMcraft API is an interface that connects the TMcraft Setup with TMflow to get all sorts of robot information and request TMflow to take different actions. In addition, this API is a Dynamic Link Library file (DLL); developers should add it to program dependencies so that the program can use the functions within the API.

TMcraft.dll consists of classes, enums, and functions for every TMcraft items. Here are those most related to TMcraft Setup:

- **ITMCraftSetupEntry** is an interface that defines the program as a TMcraft Setup. Users must define the member function **InitializeSetup()**.
- **TMcraftSetupAPI** includes the functions and provider classes to build TMcraft Setup. Note that each TMcraft item comes with its own set of provider classes. Even if some share the same name, their member functions vary.
- **Providers** are a set of classes with functions associated with different interactions with TMflow frequently used among the Programs.

- **General classes** are a collection of classes, such as DeviceIOInfo, TCPInfo, and etc., used by different types of Provider functions.
- **Enum**, or enumerators, are value types defined by a set of named constants of the underlying integral numeric type used by provider functions.

Refer the table below to overview the API capabilities across various TMcraft plugins:

Capabilities \ Plugins	Node	Shell	Toolbar	Setup
Base (Add/Edit/Delete)	✓			✓
Point (Add/Edit/Delete)	✓			✓
Tool (Add/Edit/Delete)	✓	✓	✓	✓
Digital IO (Read/Write)	✓	✓	✓	✓
Analog IO (Read/Write)	✓	✓	✓	✓
Variables (New/Edit)	✓	✓	✓	✓
Vision Job (Add/Open/Delete)	✓			
Jog the robot	✓	✓		
Freebot (Set/Get)	✓	✓	✓	✓
End Button Event	✓	✓	✓	✓
Get Current Language	✓	✓	✓	✓
Get TMflow Type	✓	✓	✓	✓
TMscript on flow project (Read/Write)	✓			✓
Login/Logout/Get Control		✓		
script Project (Add/Edit/Delete)		✓		
Robot status (Error, Run, etc.)		✓	✓	
Error Event		✓		
Virtual Robot Stick		✓		
Export/Import		✓		
Variables Runtime Value (Read/Write)		✓	Read only	

Program structure of a TMcraft Setup

Refer to the sample code below for the program structure.

```
using TMcraft;

namespace TMcraftSample
{
    public partial class MainPage : UserControl, ITMcraftSetupEntry
    {
        TMcraftSetupAPI SetupUI;

        public MainPage()
        {
            InitializeComponent();
        }

        public void InitializeSetup(TMcraftSetupAPI _SetupUI)
        {
            SetupUI = _SetupUI; //connect TMflow
        }
    }
}
```

To create the foundation of a TMcraft Setup program, please remember the key points below:

1. Include TMcraft.dll as a reference. Remember to add the namespace, `using TMcraft`, onto the program.
2. Implement the interface `ITMcraftSetupEntry` and define its member function `InitializeSetup`.
3. Declare a global object based on the class `TMcraftSetupAPI`.

The rest of the program should be all sorts of event functions that can interact with TMflow through TMcraft functions.

4. Start Programming a TMcraft Setup

Developers are favored to follow the steps to develop a TMcraft Setup.

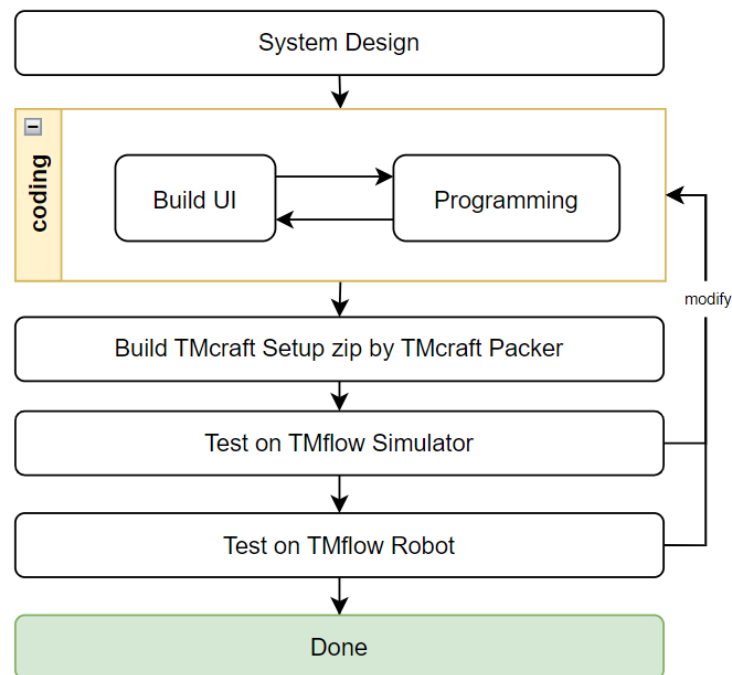


Figure 5: The Recommended Procedure for Developing a TMcraft Setup

From section 4 to section 6, users will learn how to approach these steps with a simple TMcraft Setup, *SetupDemo*.

4.1 System Design

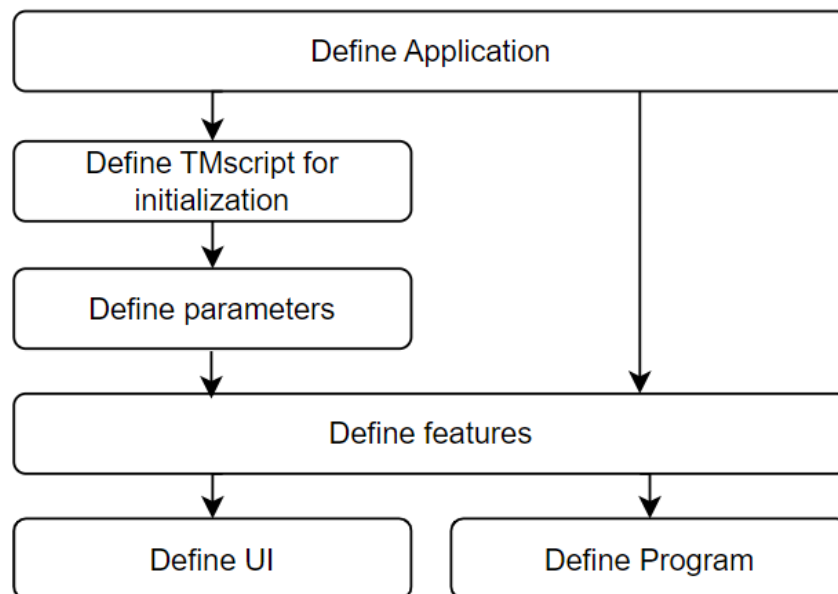


Figure 6: The Recommended Procedure of System Design

The outline of the requirements:

- The purpose of initializing control box Digital Outputs 0~5 by the settings
- Six buttons to represent the (DO) settings.
- A Textbox to show information, i.e., script and error messages.
- A button to get the script to save and show the script on the Textbox.
- A button to clear the settings and the script.
- A button to generate the initialization script by the settings.
- The project initializes the control box DO 0~5 according to the settings and displays a message "Initialization Complete" upon startup.

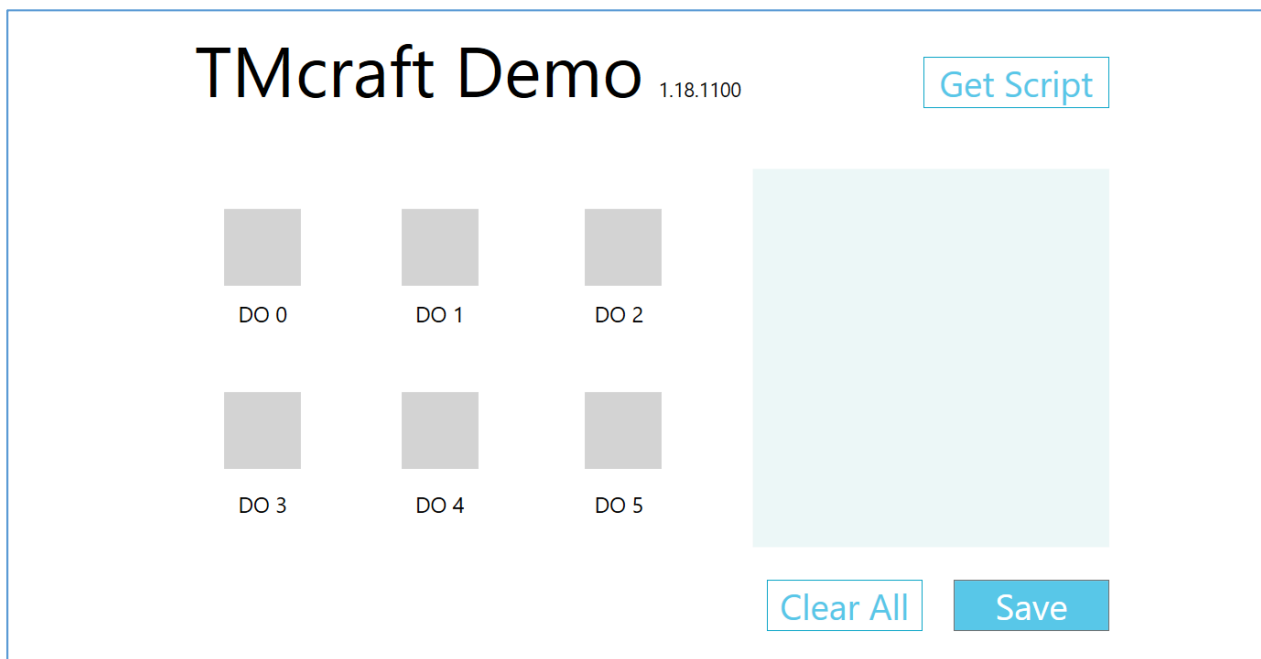


Figure 7: SetupDemo UI

The initialization script should be like this:

```
byte DO0 = [DO0 setting]
byte DO1 = [DO1 setting]
byte DO2 = [DO2 setting]
byte DO3 = [DO3 setting]
byte DO4 = [DO4 setting]
byte DO5 = [DO5 setting]
```

```
IO["ControlBox"].DO[0] = DO0
IO["ControlBox"].DO[1] = DO1
IO["ControlBox"].DO[2] = DO2
IO["ControlBox"].DO[3] = DO3
IO["ControlBox"].DO[4] = DO4
IO["ControlBox"].DO[5] = DO5
```

Note that **DO[n] setting** are the value (0 or 1) by the settings on the UI.

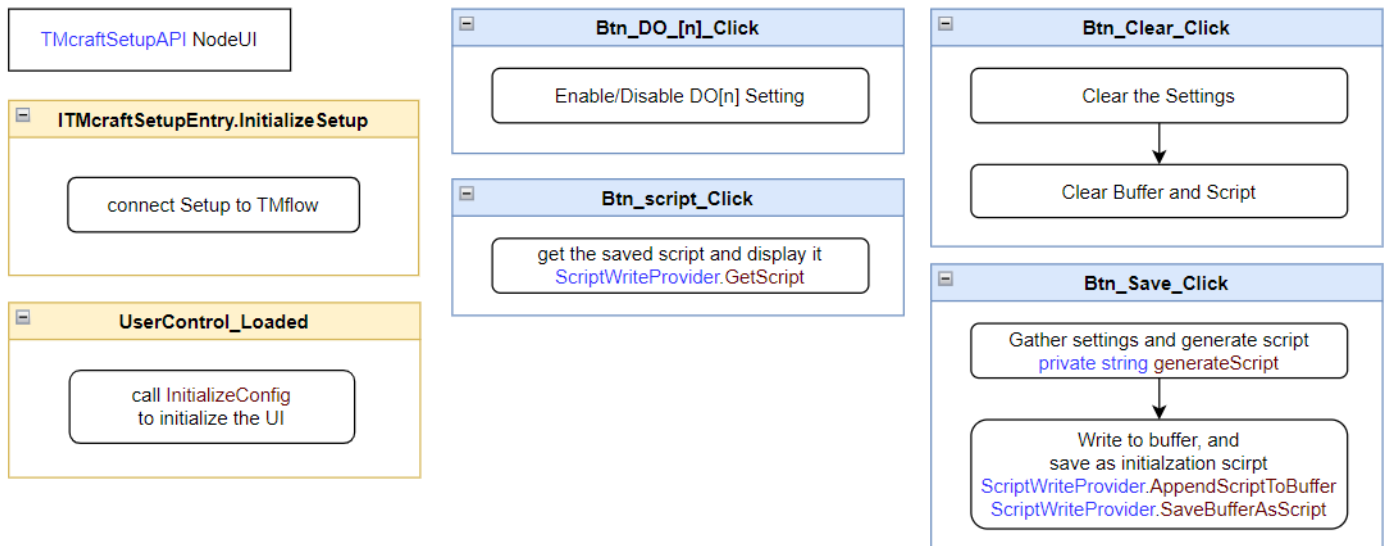
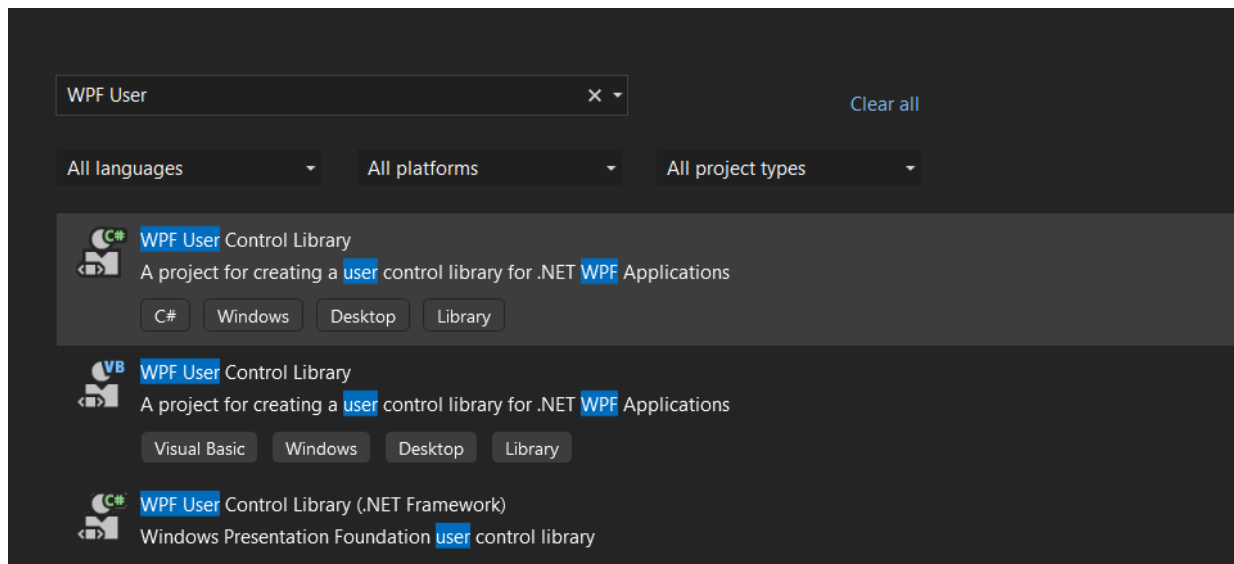
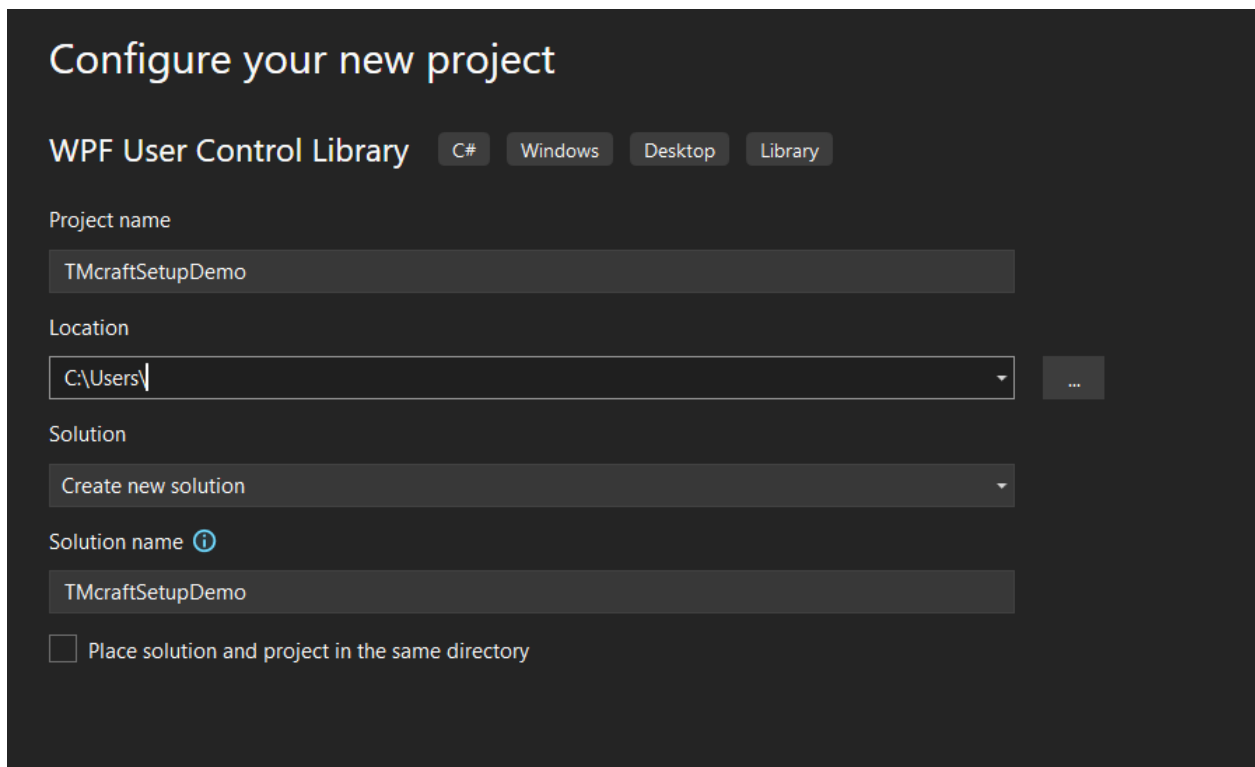


Figure 8: Program architecture of SetupDemo

4.2 Create a WPF Application Project

This section is written based on the usage of Microsoft Visual Studio 2022. First, users can create a new project with the *WPF User Control Library* template.





Configure your new project

WPF User Control Library C# Windows Desktop Library

Project name
TMcrafterSetupDemo

Location
C:\Users\...

Solution
Create new solution

Solution name ⓘ
TMcrafterSetupDemo

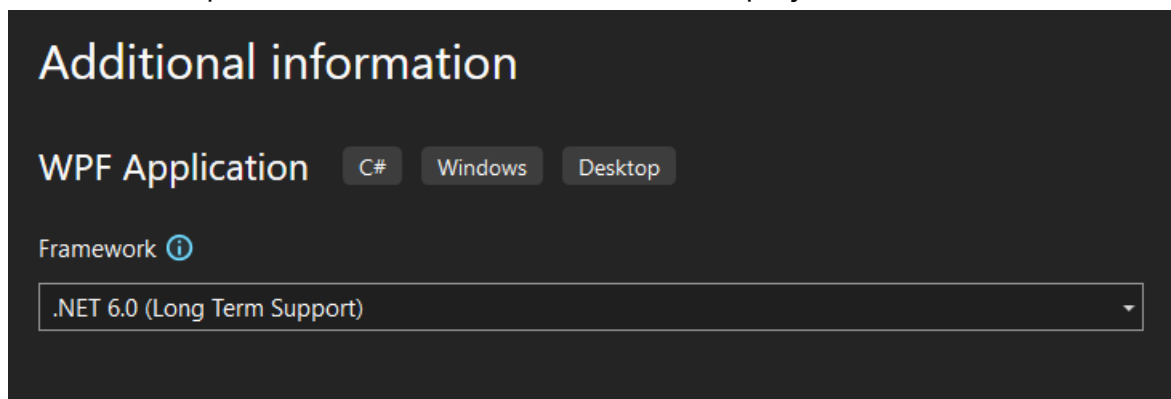
☐ Place solution and project in the same directory

Figure 9: Create a New Project with the WPF User Control Library Template

Note

NOTE:
Be reminded that it is important to use different names for the UserControl DLL and the Setup exe name to be defined later on TMcraft Packer (see Section 5).

Remember to set up with .NET 6.0 as the framework of the project.



Additional information

WPF Application C# Windows Desktop

Framework ⓘ
.NET 6.0 (Long Term Support)

Figure 10: .NET 6.0 Setup as the Framework

Once opened the project in the editor, add the TMcraft.dll as the reference.

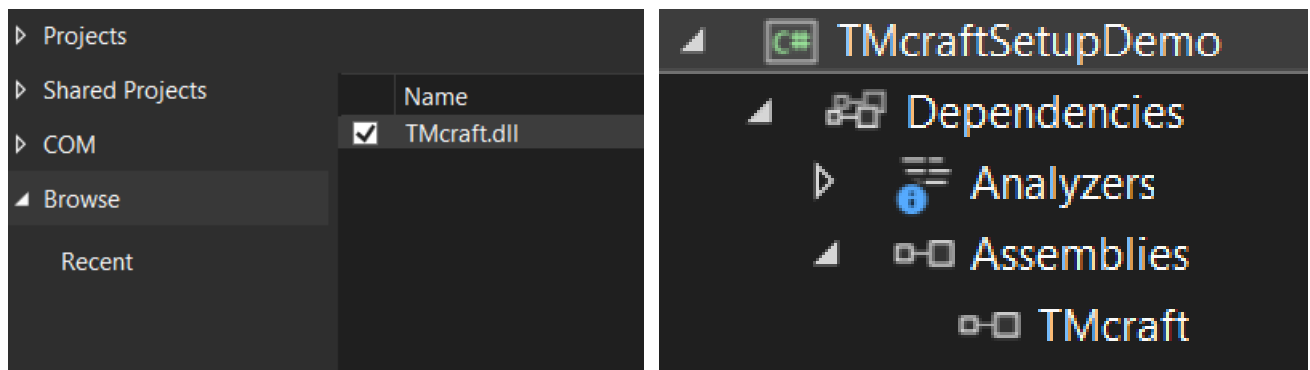


Figure 11: Add TMcraft.dll as a Reference

4.3 Source Code

This section focuses on the key aspects of the TMcraftSetupDemo source code. For the complete source code, please refer to TMcraft Development Kit_2.18\General Examples\[Setup] Setup Demo

4.3.1 MainPage.xaml

MainPage.xaml defines the UI. Here are a few points necessary:

- `x:Class="TMcraftSetupDemo.MainPage"`, used as one of the required parameters to build the executable file of the TMcraft Setup. Refer to section 5 for more details.
- `Loaded="UserControl_Loaded"`: defines the actions taken during UI initialization.
- Assign button click event functions to the corresponding UI buttons:

Button Name	Description	Click event function
Btn_DO_[n]	Set the target value of Digital Output [n].	Btn_DO_[n]_Click
Btn_script	Get the initialization script and display it.	Btn_script_Click
Btn_Clear	Clear all settings, buffer, and script.	Btn_Clear_Click
Btn_Save	Save the settings as the initialization script.	Btn_Save_Click

4.3.2 MainPage.xaml.cs

MainPage.xaml.cs defines the program of the TMcraft Setup; here are a few points necessary:

- At first, add TMcraft.dll onto the program reference and the interface ITMcraftSetupEntry.


```

using TMcraft;

namespace TMcraftSetupDemo
{
    /// <summary>
    /// Interaction logic for MainPage.xaml
    /// </summary>
    4 references
    public partial class MainPage : UserControl, ITMcraftSetupEntry
    {

```

- Declare the following global object/variable
 - A **TMcraftSetupAPI** object **setupUI** for calling all sorts of API functions through the program
 - Six Boolean variables for the settings of the target digital output.
 - A string variable for the script template (half of the initialization script)
 - others

```

public partial class MainPage : UserControl, ITMcraftSetupEntry
{
    TMcraftSetupAPI setupUI;
    bool DO_0 = false; bool DO_1 = false; bool DO_2 = false;
    bool DO_3 = false; bool DO_4 = false; bool DO_5 = false;

    string scriptTemplate = "IO[\"ControlBox\"] DO[0] = DO0\r\nIO
    [\"ControlBox\"] DO[1] = DO1\r\nIO[\"ControlBox\"] DO[2] = DO2\r\nIO
    [\"ControlBox\"] DO[3] = DO3\r\nIO[\"ControlBox\"] DO[4] = DO4\r\nIO
    [\"ControlBox\"] DO[5] = DO5\r\nDisplay(\"Initialization Complete\");

    SolidColorBrush Color_Disable = new SolidColorBrush();
    SolidColorBrush Color_Enable = new SolidColorBrush();

```

- Implement the member function **InitializeSetup()** in **ITMcraftSetupEntry**. This function activates when the Setup Program begins and interacts with TMflow.

```

0 references
public void InitializeNode(TMcraftNodeAPI _nodeUI)
{
    NodeUI = _nodeUI; //connect TMflow
}

```

- Define the **UserControl_Loaded** function. This function primarily retrieves the saved configuration and sets up the UI through **InitializeConfig()**. Further details will follow later in this section.

```

string str_Version = TMcraftSetupAPI.Version;
Label_Version.Content = str_Version;

if (setupUI == null || setupUI.RobotStatusProvider == null)
{
    //MessageBox.Show("No Connection with TMflow...");
    TextBlock_MsgBox.Text = "No Connection with TMflow...";
    return;
}

InitializeConfig();

```

- Define the six click event functions, **Btn_DO_[n]_Click**, to assign the target value for the Digital Output.

```

private void Btn_DO_0_Click(object sender, RoutedEventArgs e)
{
    if (DO_0)
    {
        Btn_DO_0.Background = Color_Disable;
        DO_0 = false;
    }
    else
    {
        Btn_DO_0.Background = Color_Enable;
        DO_0 = true;
    }
}

```

- Define the click event, **Btn_script_Click**, to retrieve and display the saved initialization script of the current TMcraft Setup in the Textbox.

```

if (setupUI == null || setupUI.RobotStatusProvider == null)
{
    TextBlock_MsgBox.Text = "No connection with TMflow...";
    return;
}
else
{
    setupUI.ScriptWriteProvider.GetScript(out script);
    TextBlock_MsgBox.Text = script;
}

```

- Define the click event, **Btn_Clear_Click**, to clear the settings (disabled by default), the script buffer, and the initialization script.

```

Btn_DO_0.Background = Color_Disable; DO_0 = false;
Btn_DO_1.Background = Color_Disable; DO_1 = false;
Btn_DO_2.Background = Color_Disable; DO_2 = false;

setupUI.ScriptWriteProvider.ClearBuffer();
Thread.Sleep(50);
setupUI.ScriptWriteProvider.SaveBufferAsScript();

```

- Define the **Btn_Save_Click** click event. It first calls the **generateScript()** function, which

collects the settings and creates the corresponding script. The script consists of two parts.

- (1) Assign the byte value to represent the target status of the Digital Outputs.

```
private void generateScript(out string _str_Script, out Dictionary<string,
string> _setupConfig)
{
    string str_Script = string.Empty;
    Dictionary<string, string> setupConfig = new Dictionary<string, string>
    ();

    if (DO_0)
    {
        str_Script += "byte DO0 = 1" + Environment.NewLine;
        setupConfig.Add("DO_0", "1");
    }
    else
```

- (2) The template script, which set the Digital Outputs.

```
str_Script += scriptTemplate + Environment.NewLine;

_str_Script = str_Script;
_setupConfig = setupConfig;
```

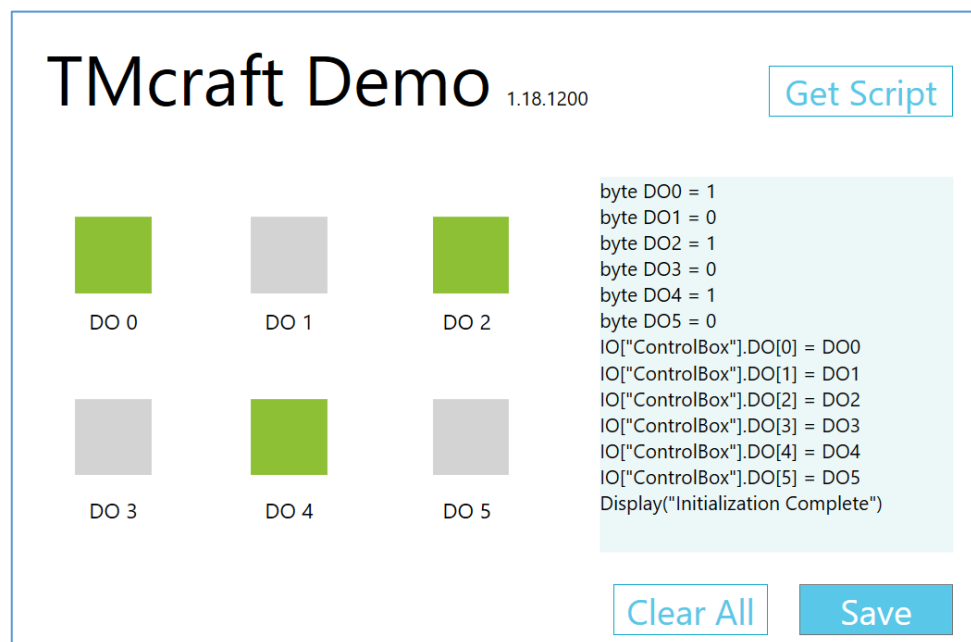


Figure 12: an example of script generated according to the settings

Also, the `generateScript()` function saves the settings into a Dictionary type and outputs them along with the script string to the `Btn_Save_Click()` function.

Next, it empties the script buffer, adds the script to the buffer, and stores the buffer as

an initialization script. Remember, the buffer clears when the Setup Program closes. To finish, it saves the project configurations by executing `DataStorageProvider.SaveData()`.

```
//Inscribe initialization script
setupUI.ScriptWriteProvider.ClearBuffer();
setupUI.ScriptWriteProvider.AppendScriptToBuffer(script);
Thread.Sleep(50);
setupUI.ScriptWriteProvider.SaveBufferAsScript();

//Save the setting configuraiton
setupUI.DataStorageProvider.SaveData(setupConfig);
```

- The `InitializeConfig()` function starts by retrieving the data storage for the current TMcraft Setup from the project. It then checks the size of the returned Dictionary. If the Dictionary is empty, indicating a new installation, it calls `EmptyConfig()` to set the UI to its default state. If the Dictionary size is 6, it calls `setupConfig()` to configure the UI based on the saved settings.

```
public void InitializeConfig()
{
    try
    {
        if(setupUI == null || setupUI.DataStorageProvider == null)
        {
            TextBlock_MsgBox.Text = "TMflow not connected...";
            return;
        }
        else
        {
            Dictionary<string, object> setupConfig;
            setupUI.DataStorageProvider.GetAllData(out setupConfig);

            switch(setupConfig.Count)
            {
                case 0:
                    TextBlock_MsgBox.Text = "No intialization script yet.";
                    EmptyConfig();
                    return;
                case 6:
                    SetConfig(setupConfig);
                    return;
                default:
                    TextBlock_MsgBox.Text = "Invalid Number of data: " + setupConfig.ToString();
                    return;
            }
        }
    }
}
```

```

public void SetConfig(Dictionary<string, object> setupConfig)
{
    object Config_0; object Config_1; object Config_2;
    object Config_3; object Config_4; object Config_5;

    try
    {
        if (setupConfig.ContainsKey("DO_0"))
        {
            setupConfig.TryGetValue("DO_0", out Config_0);
            switch (Config_0.ToString())
            {
                case "0":
                    Btn_DO_0.Background = Color_Disable;
                    DO_0 = false;
                    break;
                case "1":
                    Btn_DO_0.Background = Color_Enable;
                    DO_0 = true;
                    break;
                default:
                    TextBlock_MsgBox.Text = "Invalid DO_0 value: " + Config_0.ToString();
                    return;
            }
        }
    }
}

```

Note**NOTE:**

There are some instructions recommended when using the TMcraft API functions:

1. Check if the [TMcraftSetupAPI](#) object is null or not
2. Use a [uint](#) object to get the result of the API function (replied by TMflow)
3. Check if the result is 0 or not; if not, call [GetErrMsg\(\)](#) to get the corresponding error message
4. Be aware that errors might not only originate from TMflow but also from the API itself. These errors are represented as [TMcraftErr](#) objects and can be retrieved by calling [GetErrMsg\(\)](#).

4.4 Compile and Test UI

Once the source code is ready, compile the code and generate the DLL file. Be reminded to double-check items below beforehand.

1. Right-click on the DLL Project on the Solution Explorer to open the Properties page.
2. On the Properties\ **Application** page, check the parameters. The most important is to make sure the **Output type** is **Class Library** and **Target framework** is **.NET 6.0**.

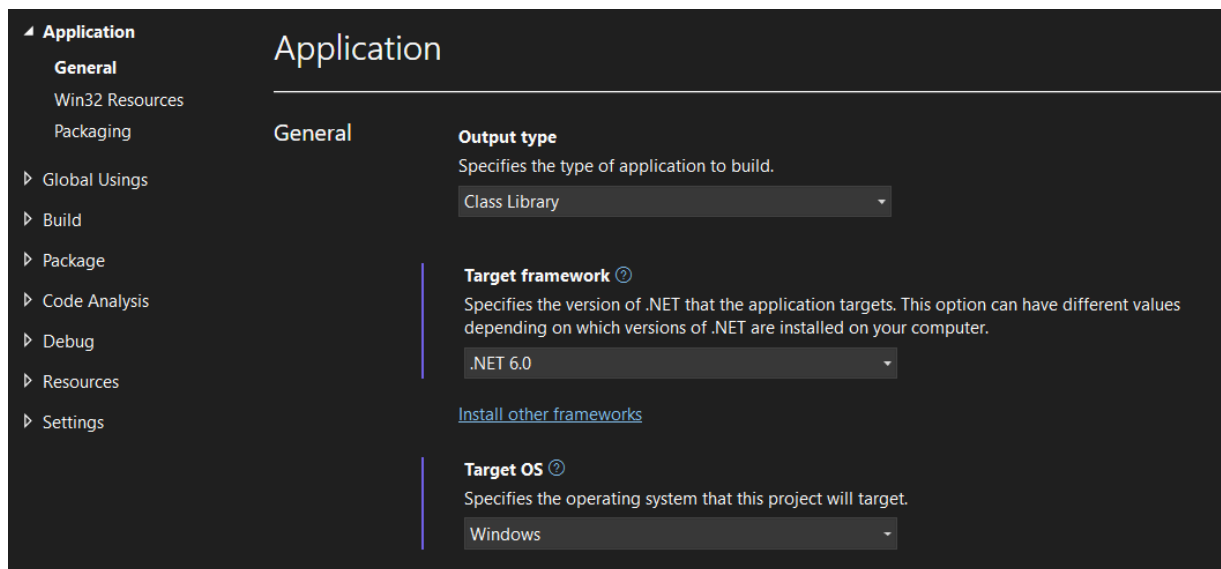


Figure 13: Set the Application Property before Building the UserControl Library

Before constructing the TMcraft Setup Package with the UserControl DLL, users should test the UI functioning at the very first.

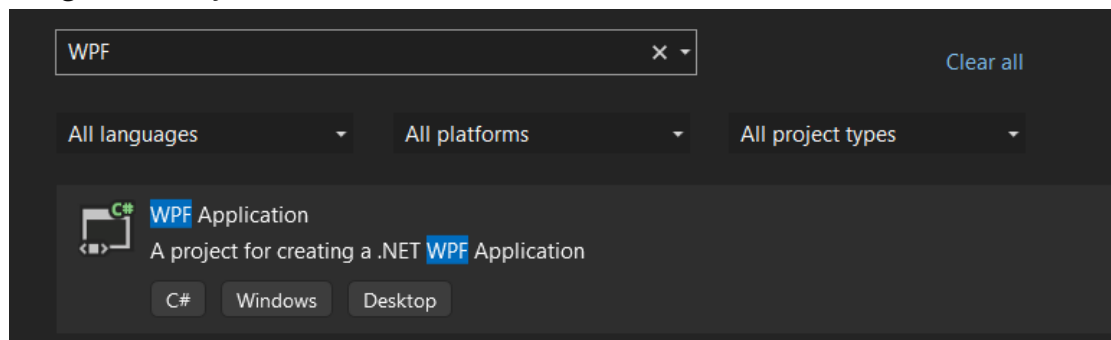


Figure 14: Create a WPF Application for UI Testing

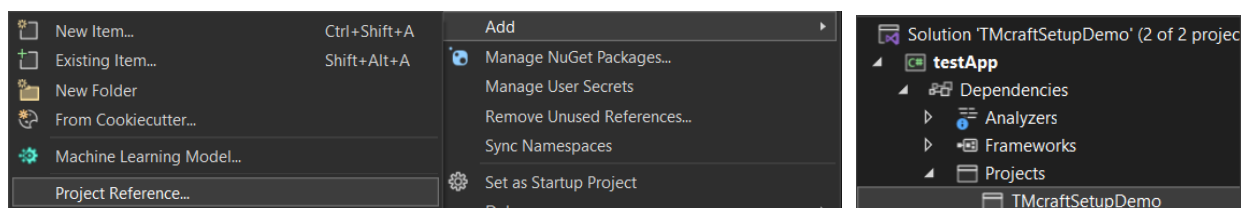


Figure 15: Add the TMcraftSetupDemo User Control as Reference

The application source code is simple, i.e., to define a UserControl object with TMcraftSetupDemo .MainPage (namespace of the Setup UI) and add it to the Grid. (Remember to name it first on the xaml file or the Properties page of the Grid)

```

using TMcrafterSetupDemo;

namespace testApp
{
    /// <summary> Interaction logic for MainWindow.xaml
    2 references
    public partial class MainWindow : Window
    {
        0 references
        public MainWindow()
        {
            InitializeComponent();

            UserControl setupUI = new MainPage();
            gd_Main.Children.Clear();
            gd_Main.Children.Add(setupUI);
        }
    }
}

```

Figure 16: Source code of the setup testing application

Set the testing application as the Startup Project before compiling the code. Then, run the program to test the Setup UI.

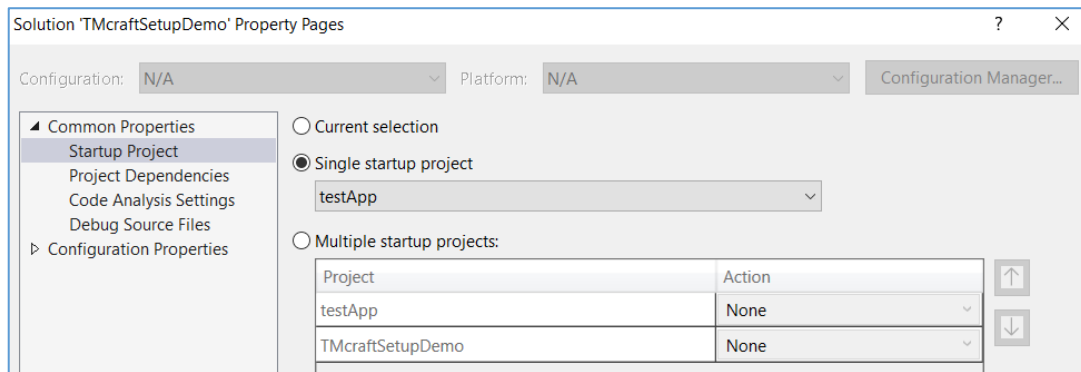


Figure 17: Set up the Startup Project

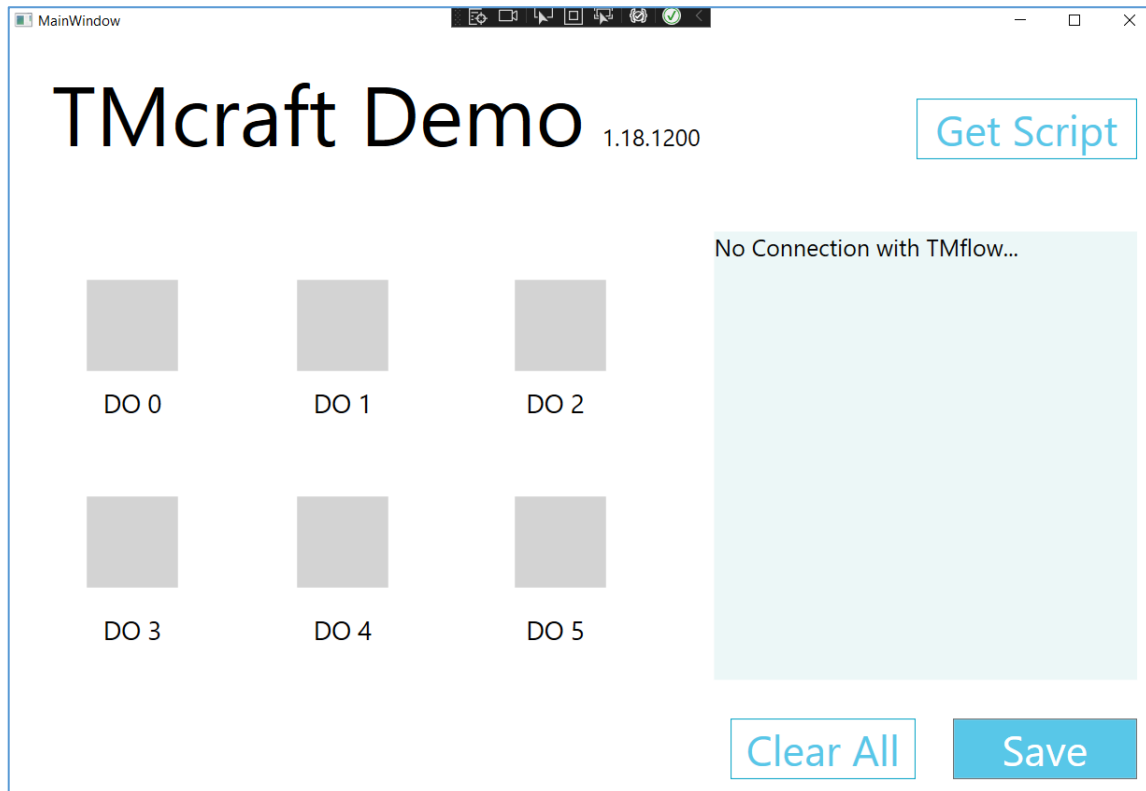


Figure 18: Test the Setup UI

5. Build TMcraft Setup Package

The previous section described how to build the TMcraft Setup Program. Before using it on TMflow, it must be packaged using the TMcraft Packer.

First, prepare a source folder with the following items.

- The UserControl Library file (dll) of the Setup UI
- All files from the TMcraft API (1.18 or above) folder from the development kit
- The Icon Image
- Other resource files, such as:
 - Other reference files used by the Program
 - Files used by the Setup, such as media, documents, etc.

Note

NOTE:

Suggest putting all files from the `..\bin\Debug` folder into the source folder.

Next, open TMcraft Packer (1.18 or above) and create a project. Select **Setup** as the target TMcraft item and complete the Project Settings. The system saves the project as a `.tmcraft` file at the specified file path in the **Location** field.

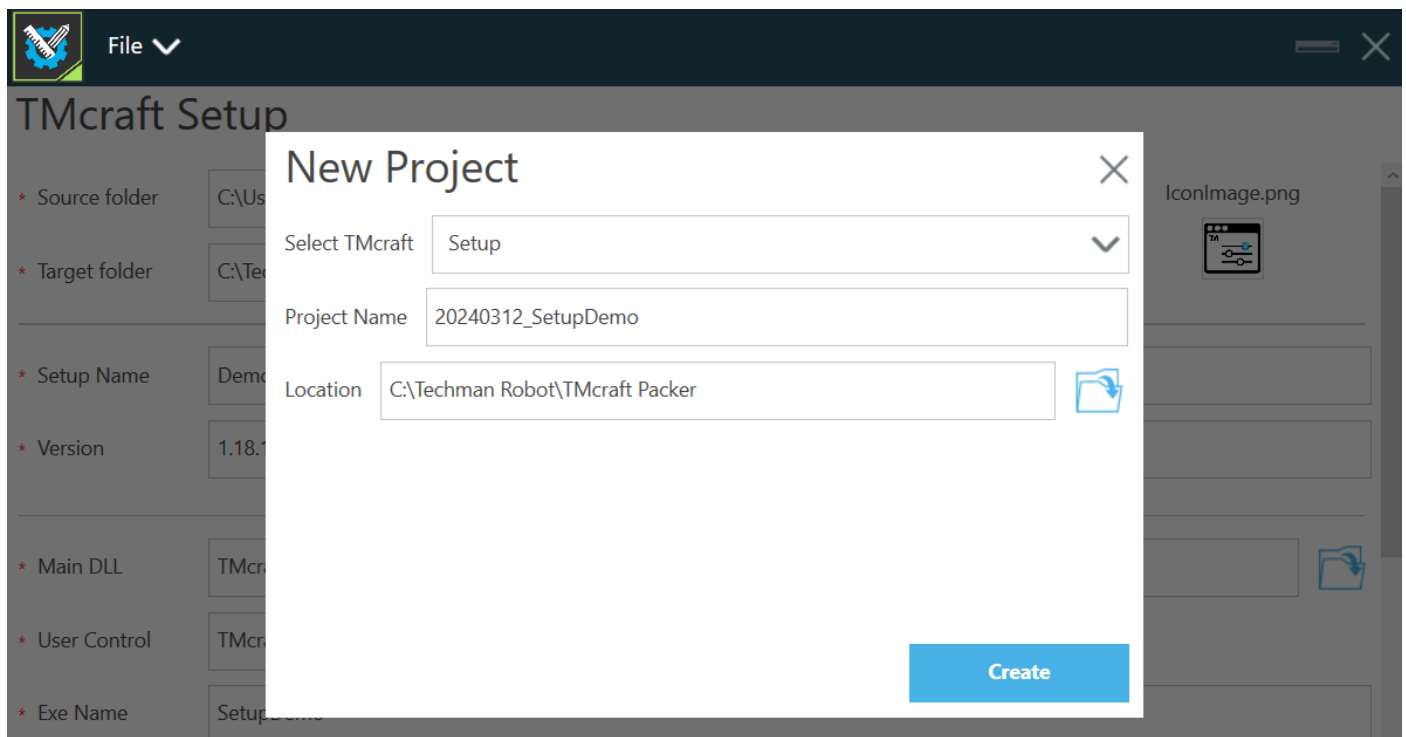


Figure 19: Create a TMcraft Packaging Project

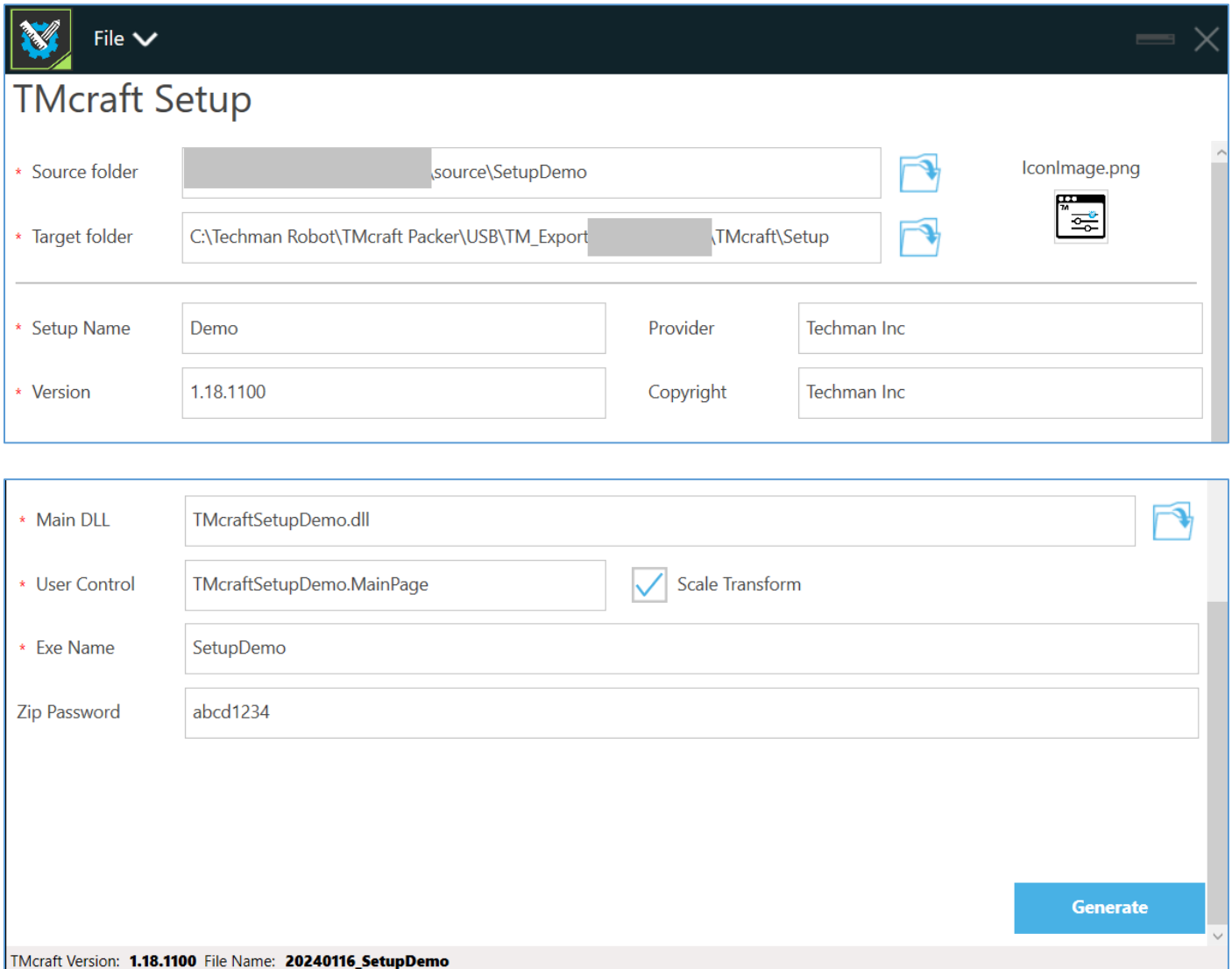
**IMPORTANT:**

Ensure that TMcraft Packer is running in the environment with .NET 6.0 installed.

After opening the project, users can see a form of parameters required to build and package the TMcraft Setup. Refer to the following for explanations.

Parameters	Description	Requisite
Source folder	Path of the source folder	<input type="radio"/>
Target folder	Path of the target folder. For any new projects, the default target folder is <code>..\TMcraft Packer\USB\TM_Export\[PC Name]\TMcraft\Setup</code>	<input type="radio"/>
Setup Name	Name of the Setup, which is shown on the list of Flow Project\Project function\TMcraft Item	<input type="radio"/>
Provider	Name of the developer or the company providing this Setup, which is shown on the TMcraft Management Page	
Version	The version of the TMcraft Setup, which is registered onto the exe file and also shown on the TMcraft Management Page	<input type="radio"/>
Copyright	Copyright of the TMcraft Setup, which is registered onto the exe file	
Main DLL	File name of the User Control Library of the TMcraft Setup (as in Section 4)	<input type="radio"/>
User Control	The startup user control from the Main DLL. Since there might be several user controls within the User Control Library, it is necessary to define which one is used as the Main User Control. The format of this parameter should be <code>[Namespace].[UserControlName]</code>	<input type="radio"/>
Exe Name	Defines the name of the executable file generated. Be reminded that it should NOT be identical to the User Control Library file name.	<input type="radio"/>
Scale Transform	Denotes if the TMcraft Setup would change the scale automatically by the resolution of the Control Box (1366 x 768 pixels). Disable the transformation if confident in the defined width and height.	<input type="radio"/>
Zip Password	Developers can define the password of the zip file. The password length should be between 6 and 256 characters of the non-case-sensitive Latin alphabet and numbers.	

※ Complete all parameters labeled with a red star in the TMcraft Packer GUI before proceeding.



The screenshot shows the TMcraft Setup application window. The title bar is dark blue with a 'File' menu and window controls. The main area is white with a title 'TMcraft Setup'. It contains several input fields and checkboxes. The 'Source folder' is set to 'source\SetupDemo'. The 'Target folder' is 'C:\Techman Robot\TMcraft Packer\USB\TM_Export\TMcraft\Setup'. The 'Setup Name' is 'Demo', 'Version' is '1.18.1100', 'Provider' is 'Techman Inc', and 'Copyright' is 'Techman Inc'. The 'Main DLL' is 'TMcraftSetupDemo.dll'. The 'User Control' is 'TMcraftSetupDemo.MainPage' with a checked 'Scale Transform' checkbox. The 'Exe Name' is 'SetupDemo' and the 'Zip Password' is 'abcd1234'. A blue 'Generate' button is at the bottom right. A status bar at the bottom shows 'TMcraft Version: 1.18.1100' and 'File Name: 20240116_SetupDemo'.

* Source folder	source\SetupDemo	IconImage.png
* Target folder	C:\Techman Robot\TMcraft Packer\USB\TM_Export\TMcraft\Setup	
* Setup Name	Demo	Provider: Techman Inc
* Version	1.18.1100	Copyright: Techman Inc
* Main DLL	TMcraftSetupDemo.dll	
* User Control	TMcraftSetupDemo.MainPage	<input checked="" type="checkbox"/> Scale Transform
* Exe Name	SetupDemo	
Zip Password	abcd1234	

Generate

TMcraft Version: 1.18.1100 File Name: 20240116_SetupDemo

Figure 20: Set up the Packaging of TMcraft Setup

Once all requirements are ready, click **Generate**. It might take several minutes to package, and users can check the current progress on the Page (Users will see an error message if anything goes wrong). After packaging the TMcraft Setup successfully, users should see the successful status label.

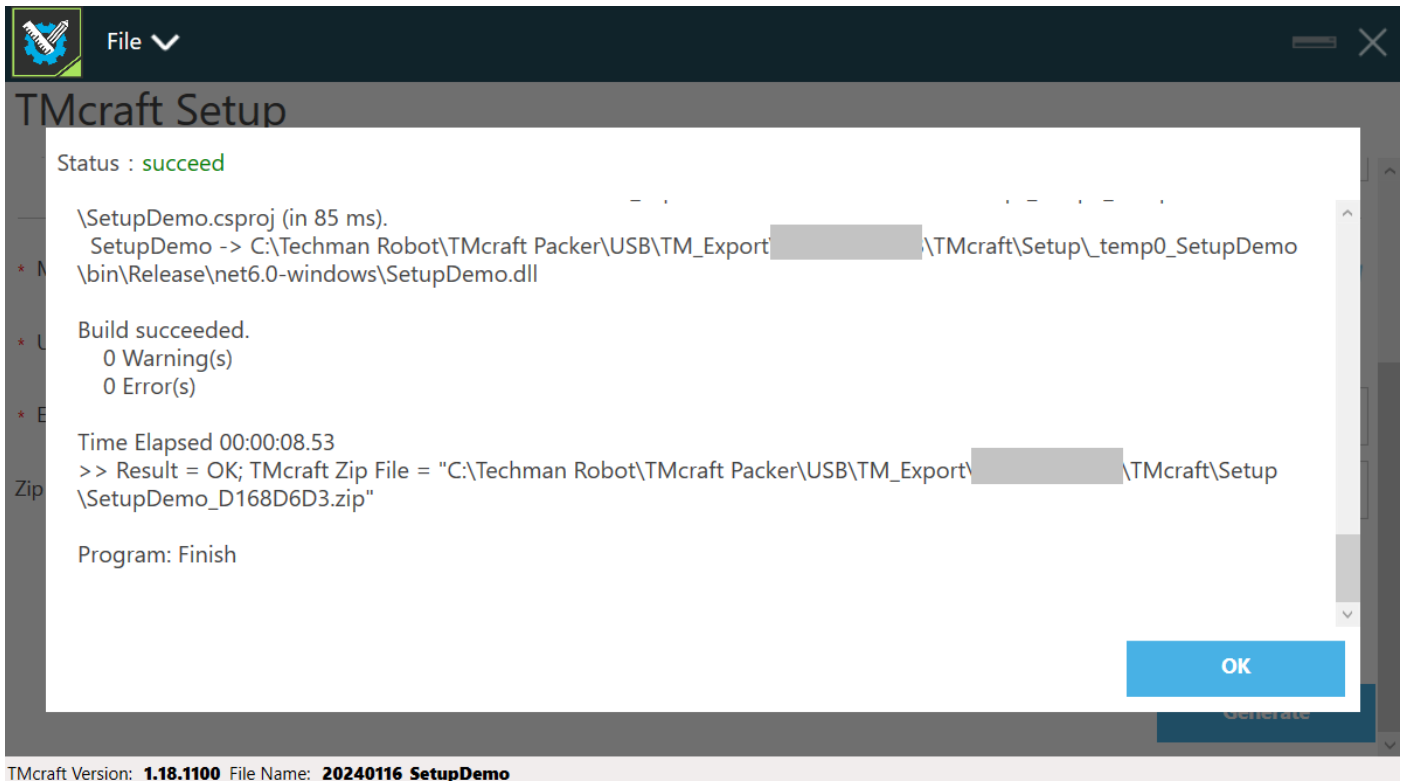


Figure 21: The Message Shown After Packaged Successfully

Finally, users can find the TMcraft Setup zip file named after *[Exe Name]_[Checksum]* in the target folder.

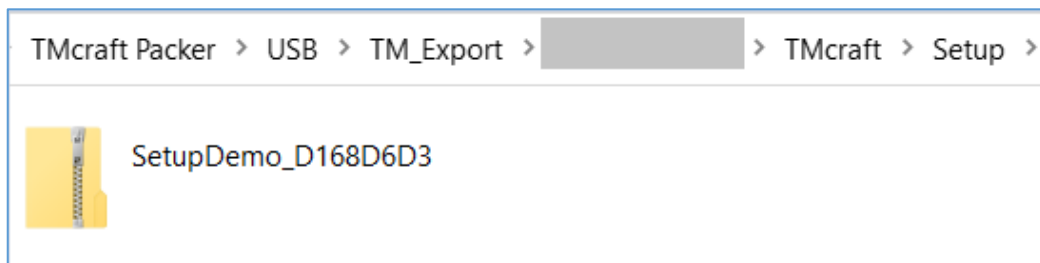


Figure 22: TMcraft Setup zip at Target folder

6. Installation Code and Checksum

To protect the rights and interests of developers, each TMcraft Packer Project has its own encrypted Installation Code (GUID-based). Any TMcraft items generated by this project will share the same Installation Code and be able to replace one another on the same robot. Additionally, items generated from two different TMcraft Packer Projects (without identical Installation Codes) cannot replace each other, even if they share the same name and configuration. Therefore, developers should keep their packer project safe.

On the other hand, TMcraft Packer will also generate a checksum for each TMcraft item based on the binary footprint of the files (exe and dll) within the source folder and the Installation Code. The TMcraft item saves the checksum onto the configuration file. When importing the TMcraft item, TMflow will calculate a checksum by the same method; if these two checksums are not identical, TMflow will block the importing.

Developers can announce the checksum, allowing end users to verify it on the TMcraft Management Page to ensure the product's authenticity.

7. Use TMcraft Setup on TMflow

To use a TMcraft Setup on both TMflow (robot) and TMflow Simulator, import the TMcraft Setup Package to a robot by placing the zip file in the following path:

[Storage Drive]/TM_Export/[Folder]/TMcraft/Setup

Plug the storage device into the robot and navigate to **System > Import/Export > Import**. Note that there is a new category: **TMcraft**. Select **TMcraft > Setup**, choose the required TMcraft Setup zip file, and click **Import**.

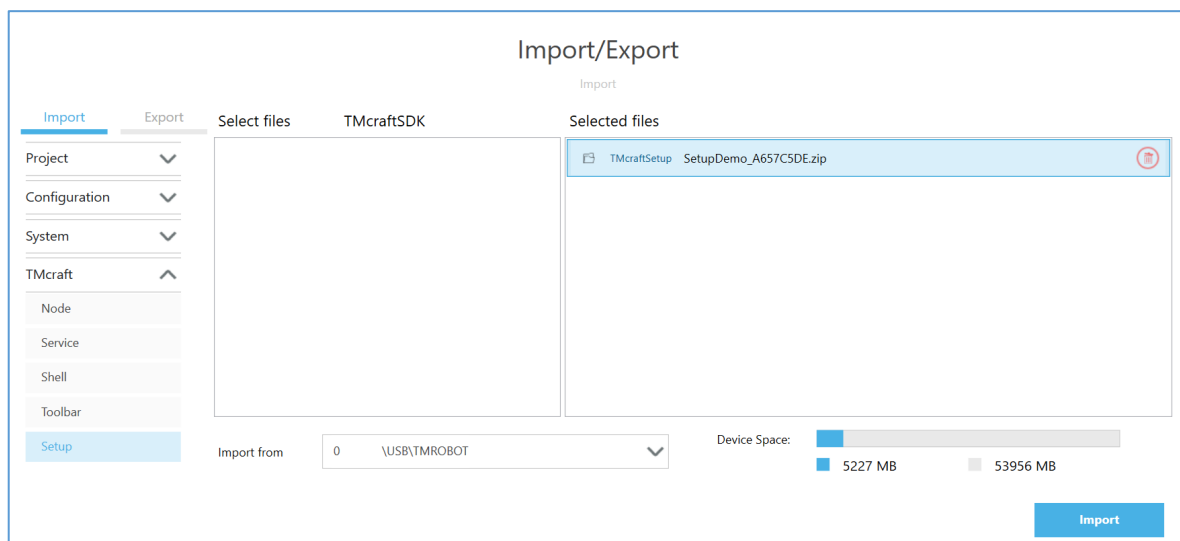


Figure 23: Import TMcraft Setup zip

Next, navigate to **Setup > Configuration > TMcraft Management > Setup**. Users can see the imported TMcraft Setup shown in the table with the following information:

- **Name:** the executable file name of the TMcraft Setup
- **Provider:** as defined in Chapter 5
- **Version:** as defined in Chapter 5
- **Checksum:** a number calculated based the binary footprint of the files (exe and dll) in the TMcraft Setup folder. Developers may publish this for identification purpose to their end-users.

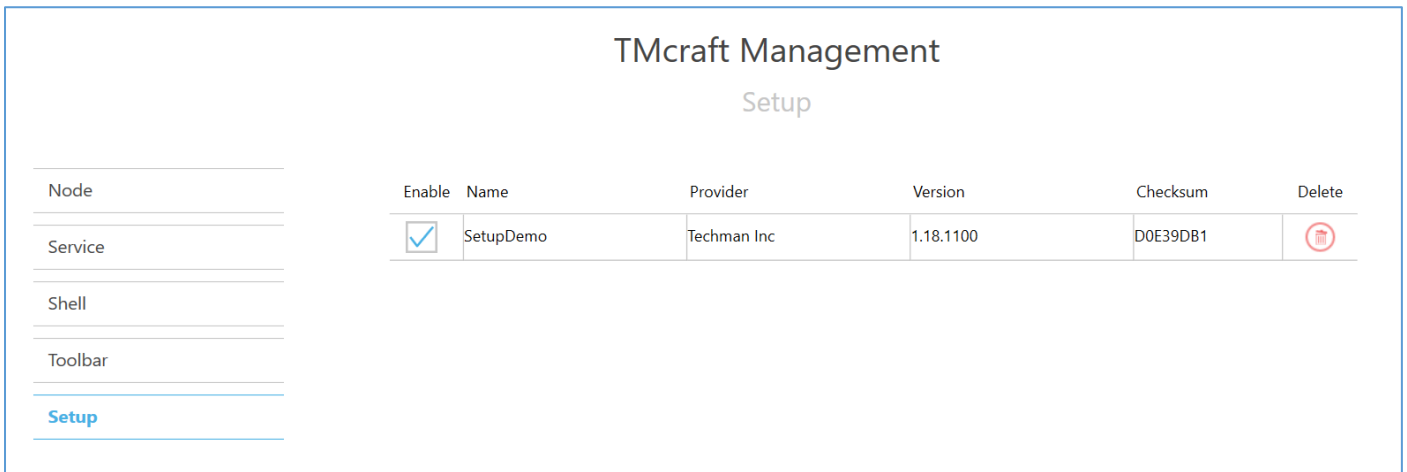


Figure 24: Enable the TMcraft Setup on TMcraft Management

Create or open a flow project, and navigate to **Project function > TMcraft Setup**. Users can choose **Demo** in the TMcraft Setup item list, and the UI will show on the right. After finishing the settings, users can click **Clear All** to reset the settings or click **Save** to generate the script and inscribe as initialization. After generating the script, users can notice that an item, **Demo**, appears on the list of Initialization scripts.

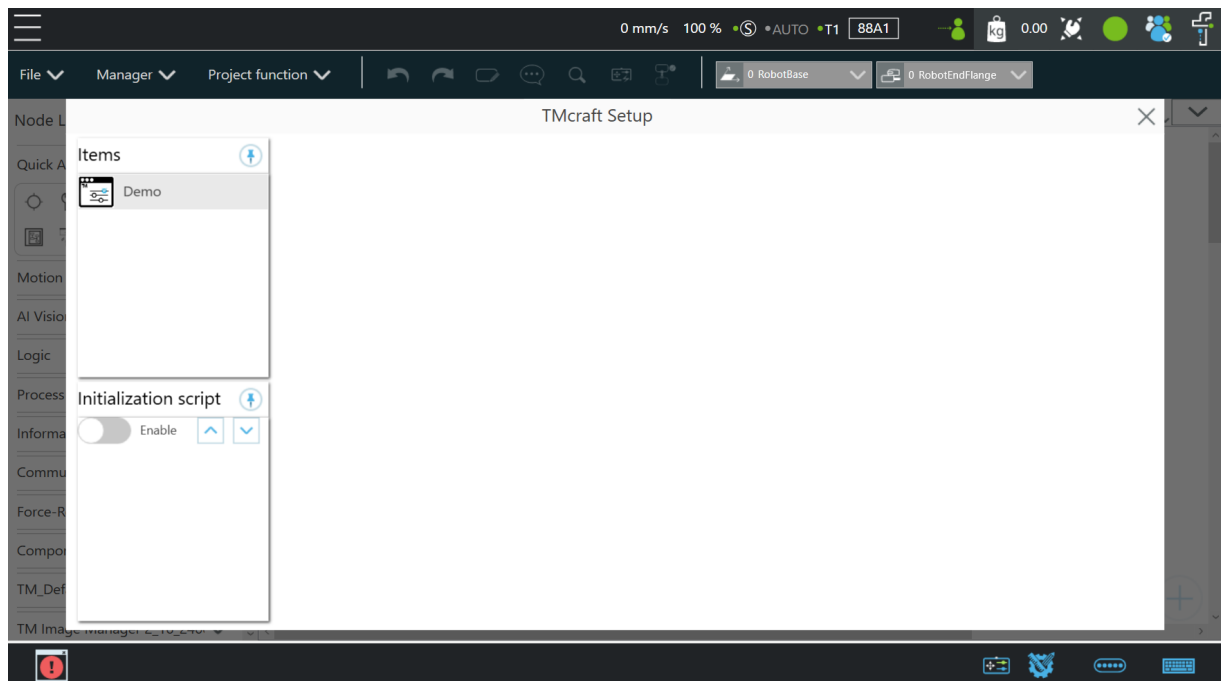


Figure 25: Project Function\TMcraft Setup Page

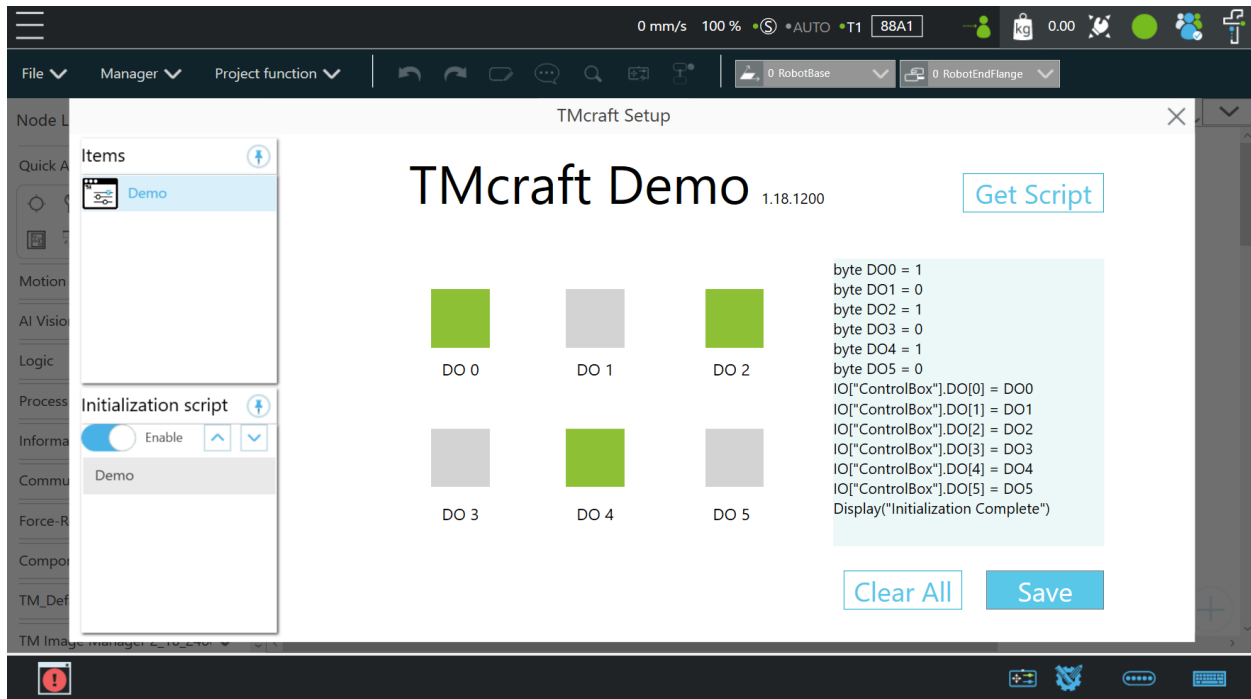


Figure 26: Finish the settings and generate the associated initialization script

Remember to slide the switch above the list to enable the Initialization script. Once the project starts, the script will execute immediately.

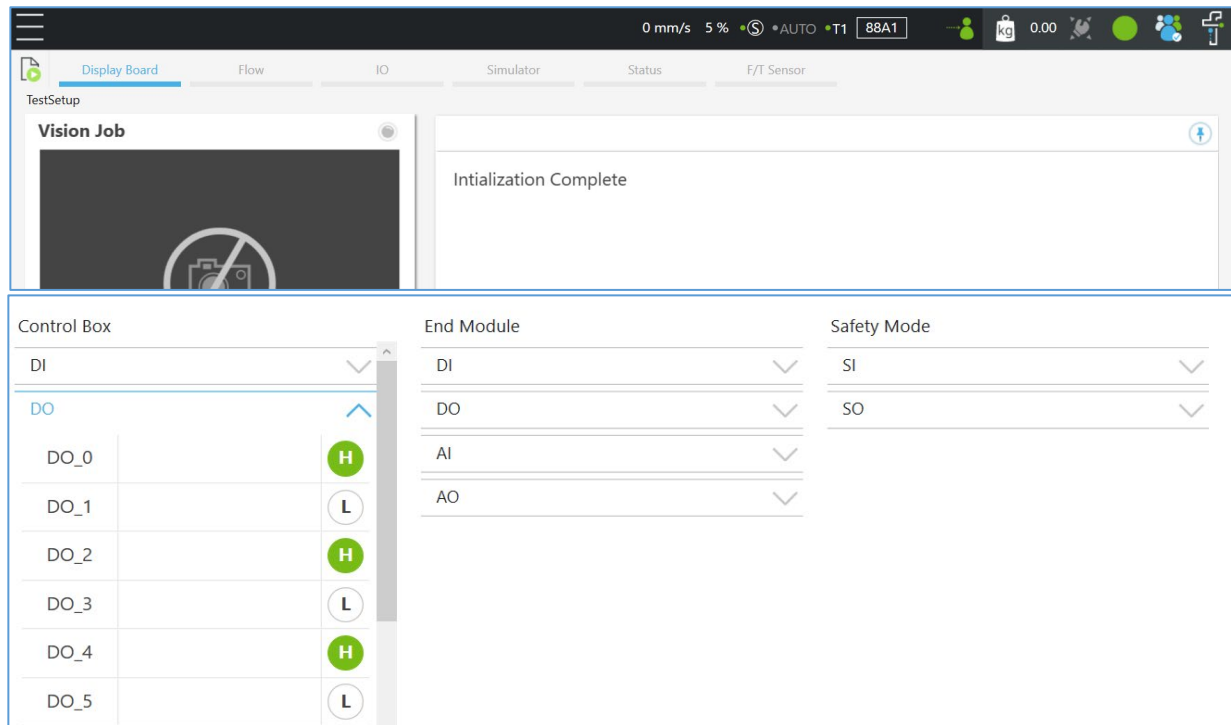


Figure 27: The Result of the Setup Demo initialization script

8. Dos & Don'ts

- About program reference/libraries:

If using NuGet to install and manage libraries within the TMcraft program, ensure the library files are accessible for placement in the TMcraft packaging source folder. Accomplish this by adding or modifying the property <CopyLocalLockFileAssemblies> to true within the csproj file. After compiling the project, users should find all reference files within the bin folder.

- About resource path definition:

When using resources in the TMcraft Program, it is necessary to use an absolute path (`(pack://application:../{Name of the User Control Library Project}; component/Resources/{String Resource file})`) instead of the relative path to access resources; otherwise, it will not work.

- Suggestions on how to use TMcraft API functions:

1. Check if the `TMcraftSetupAPI` object is null or not
2. Use a `uint` object to get the result of the API function (replied by TMflow)
3. Check if the result is 0 or not; if not, call `GetErrMsg()` to get the corresponding error message
4. Note that, instead of TMflow error, there might also have error from the API itself; this error is represented as a `TMcraftErr` object and returned by calling `GetErrMsg()`.

- Safety Assessment is necessary when using RobotJogProvider

If a TMcraft Setup utilizes RobotJogProvider functions for motion control, developers are responsible for ensuring a single point of control in compliance with ISO 10218-1.

- About writing files into Control Box through TMcraft items

Robot System will deny the following access from TMcraft executable, including: (1) system shutdown and (2) accessing to drives C and D, except for the following folders:

- D:\...TMflow\TMcraft\
- D:\...TMflow\XmlFiles\
- D:\...TMflow\TextFiles\

Also it is not allowed to create any folders by the program.

- Log

The TMcraft Program should save log files within its folder, which allows end-users to export the TMcraft zip, including the log files, for developer analysis.

- Before packaging with TMcraft Packer:

- ✓ Place all API relevant files from the TMcraft Development Kit into the source folder.
- ✓ Place all files from the bin folder of the TMcraft Project into the source folder
- ✓ Ensure TMcraft Packer is running in the environment with .NET 6.0 installed.

