GUI AND APPLICATION

TOOLSET FOR NETMF

VERSION 2.5.0.0

Table of Contents

1 Introduction

Tinkr is a complete toolset for creating graphical interfaces, hosting multiple applications and rendering alpha-blended images using any .NET Micro Framework (NETMF) device with native graphic support.

GUI designs are easily created using *Tinkr* and Microsoft's free NETMF emulator. With nearly 2 dozen built-in controls and the ability to create new custom controls there is no limit to what you can create.

While running a single application is often sufficient *Tinkr* allows you to run multiple applications at the same time. And since these applications can be housed either in the device's internal storage or externally on an SD card or USB drive you can easily support more applications than your device would normally have room to house. Working with external applications also allows for quick updating of applications, simply overwrite the related .pe file and restart the device.

Tinkr supports the P2I format for alpha-blended images introduced with *Pyxis 2* and comes with an image converter for quickly turning your 32bit images into a format that can be rendered on NETMF devices, which lack native transparency support.

2 About This Document

2.1 Intended Audience

This document is intended for users with a basic knowledge of Visual Studio, C#, and the .NET Micro Framework. Users should have a working understanding of how to create NETMF applications, deploy to the Emulator, and deploy to physical devices.

*Note: A physical device is not required to test or design with Tinkr.

2.2 Change Log

October, 17th, 2013

Fixed bug in ContextMenu

Fixed rendering error with ListviewColumn

August, 23rd, 2013

Fixed bug in launching applications

Fixed rendering error with GraphicLock

Fixed Tab rendering issue

Added Shortcut control

Added ShortcutGrid control

Added Location property to all controls

Added OnItemSelected event to Listbox

Added NextChild and PreviousChild to containers

Added Keyboard support to Listbox

Added Tab & Shift Tab support to all containers

Added KeyboardShiftDown, KeyboardCtrlDown, KeyboardAltDown properties to Core

Added FlushFileSystem to Core

Added ability to load VDS forms from byte array

Added CheckChanged event to RadioButtons

July, 5th, 2013

Updated documentation and examples

Fixed setting *Expanded* property on TreeviewNodes without children

Removed some debugging code

Fixed tab location when TabDialog was inside another container

Fixed submenu item selection

Fixed item selection on Filebox

Fixed row selection on Listview

May, 5th, 2013

Added AlternateFont and AlternateBackColor to ListboxItem

Fixed color issue on TextArea

Fixed moving caret issues on TextArea

Added Home/End/Up/Down support to TextArea

Added Font property to Checkbox

Added Font property to Textbox

Updated disabled rendering on Combobox

Updated NumericUpDown to use SystemColors.BorderColor for border

Updated Progressbar to use SystemColors.BorderColor for border

Added Font property to RadioButton

Updated Textbox to use SystemColors.WindowColor for background

Added PasswordCharacter property to Textbox

Fixed rendering glitch on TabDialogs with no Tabs

Added VDS support

Updated constructor for GraphicLock

Updated constructors for Listview

Added AddColumn, ClearColumns, RemoveColumn, RemoveColumnAt to Listview

Added Font property to TextArea

Added Menus property to Appbar

Added Icons property to Appbar

Added ForeColor property to TextArea

Added AllowMaximize, AllowMinimize, AllowClose properties to Window

Added Font property to Window

Added Title property to Window

Added new constructor for ListviewColumn

March, 23rd, 2013

Added Tag property to ListboxItem

Added GetMenuByName to Appbar

Added GetIconByName to Appbar

Fixed tap event bug on ContextMenu

Fixed BringToFront typo

Fixed background image rendering on Button

February 22nd, 2012

Initial release, no changes.

3 Getting Started with GUI Applications

3.1 Initializing Tinkr

The first thing that must be done to start working with *Tinkr* is to initialize the core; this is done by calling Core.Initialize(TouchCollection touchCollection). There are two different modes to be aware of.

NativeSingleTouch uses a threaded collection method which for touches to work properly when multiple applications are running.

ManualOrMultiTouch tells *Tinkr* not to communicate with the hardware and instead must be given touch information from the application. This is most useful for devices like GHI's CP7 module which don't use the built-in touch collector.

3.2 Using GHI's CP7 Module

To aid in using the CP7, and still having full gesture support, *Tinkr* offers a class called CP7TouchHandler; which can be found under *Skewworks.Gadgeteer.CP7Helper*.

Example Usage

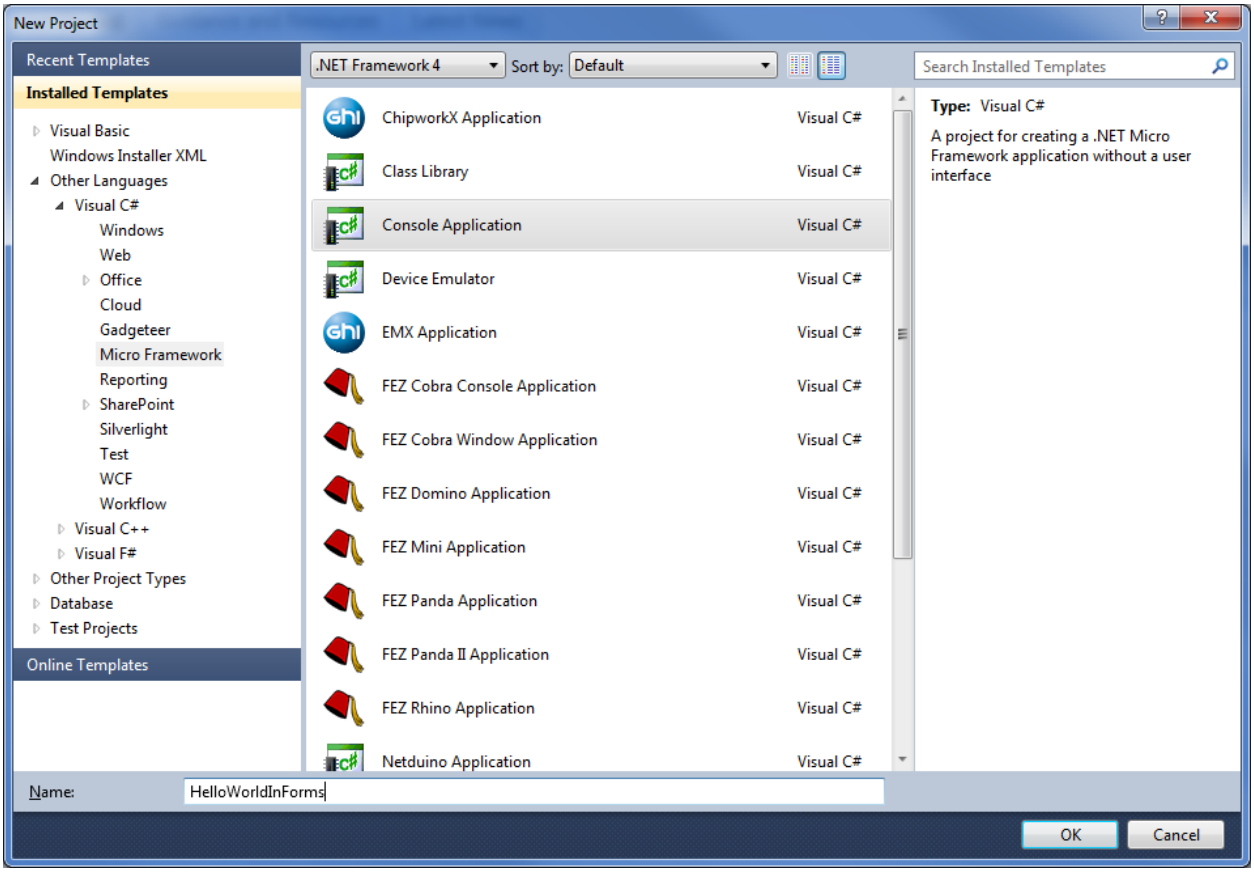
```
            Core.Initialize(TouchCollectionMode.ManualOrMultiTouch);
            CP7TouchHandler CP7 = new CP7TouchHandler(display_CP7);
            display_CP7.ScreenPressed += (Display_CP7 sender, Display_CP7.TouchStatus TT)
=> CP7.CP7Pressed(new Point(TT.touchPos[0].xPos, TT.touchPos[0].yPos));
            display_CP7.ScreenReleased += (Display_CP7 sender) => CP7.CP7Released();
            display_CP7.BackPressed += (Display_CP7 sender) => CP7.CP7BackPressed();
            display_CP7.HomePressed += (Display_CP7 sender) => CP7.CP7HomePressed();
            display_CP7.MenuPressed += (Display_CP7 sender) => CP7.CP7MenuPressed();
```

3.3 Hello World (Pure NETMF)

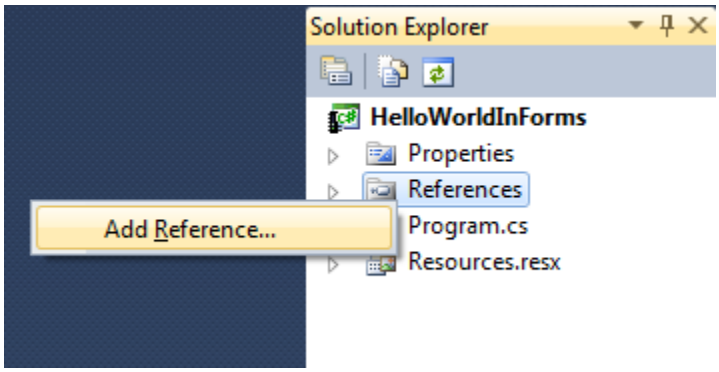
Unlike NETMF applications you may be used to *Tinkr* doesn't rely on WPF, nor do you need to *Flush* a Bitmap in order to update your screen. Instead, you will need to create and activate a Form.

You can have as many Forms as you wish however, only one can be active at a time. Activating a Form is simple and you can switch between Forms with a single line of code.

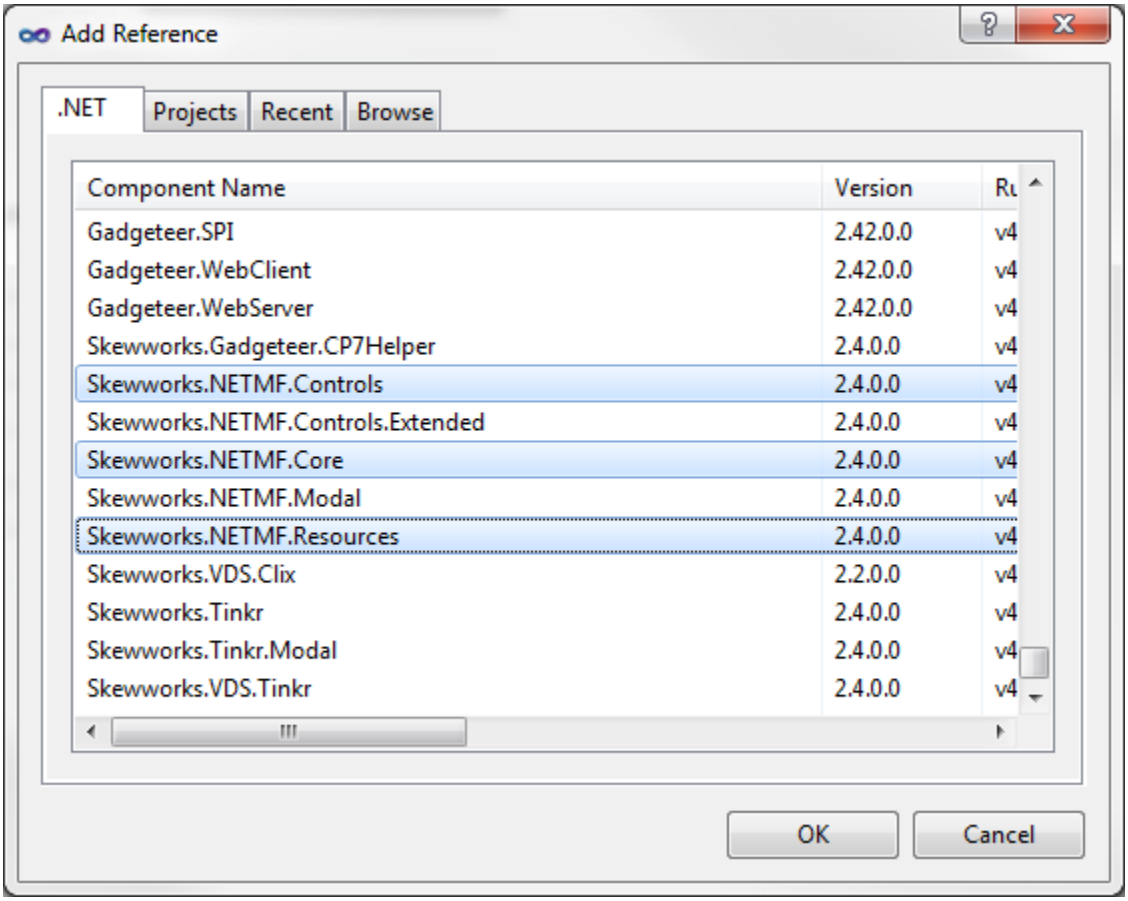
To begin, let's create a new Micro Framework Console Application project in Visual Studio; we'll name it HelloWorldInForms.



Next we need to add references to *Tinkr*; to do this, right-click on the "References" folder to the right and select "Add Reference".



Next select the *Tinkr* references as seen below as well a **Microsoft.SPOT.Graphics**



Now we're going to add some code, so you'll need to open your Program.cs file and place the following code inside.

```
using System;
using System.Threading;

using Microsoft.SPOT;

using Skewworks.NETMF;
using Skewworks.NETMF.Controls;
using Skewworks.NETMF.Resources;

namespace HelloWorldInForms
{
    public class Program
    {

        public static void Main()
        {
            // Initialize Tinkr in normal mode
            // This allows for native touch collection
            Core.Initialize(TouchCollection.NativeSingleTouch);

            // Create the form
            Form frmMain = new Form("frmMain");

            // Create a label
            Label lbl1 = new Label("lbl1", "Hello World!", Fonts.Droid12, 4, 4);

            // Add label to the form
            frmMain.AddChild(lbl1);

            // Activate the form
            Core.ActiveContainer = frmMain;

            // Make sure application doesn't exit
            Thread.Sleep(Timeout.Infinite);
        }
    }
}
```

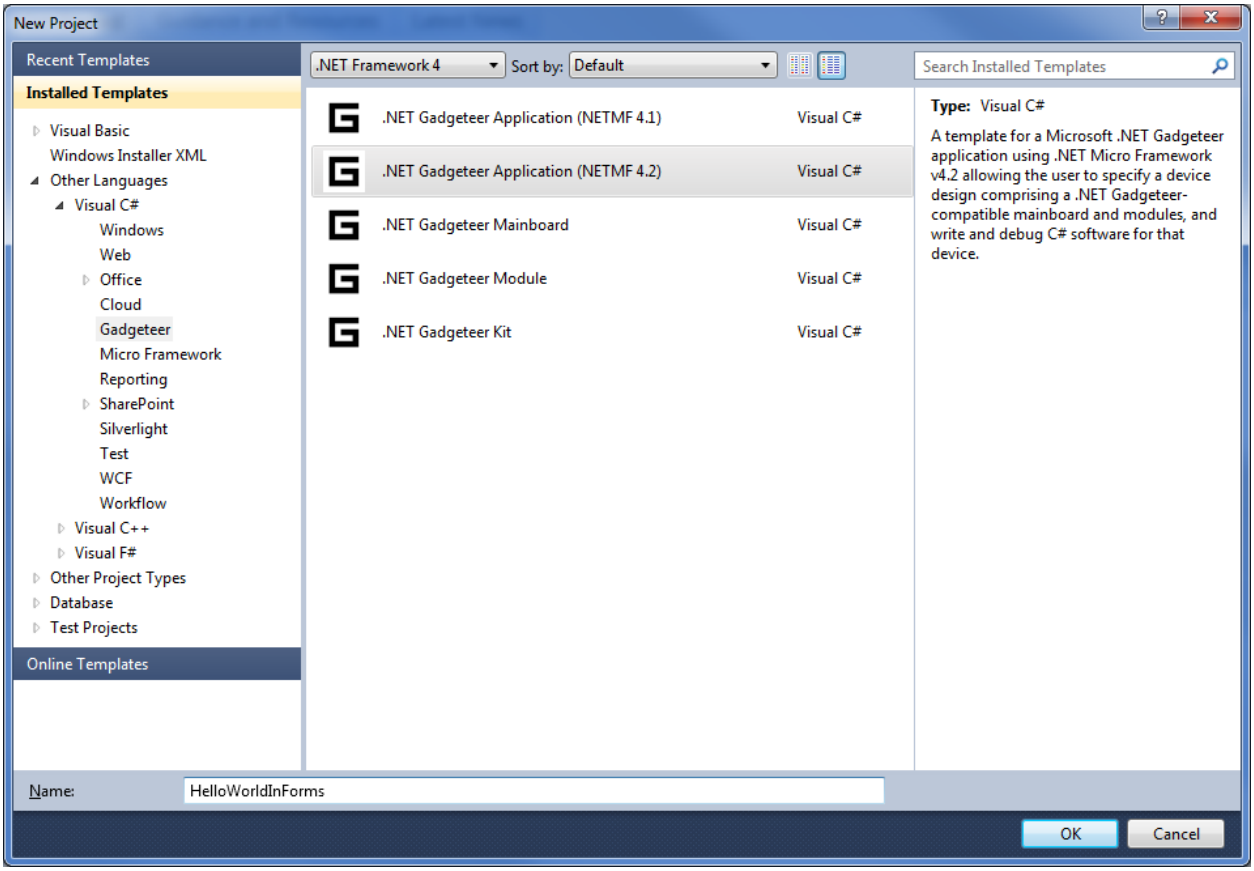
Once your Program.cs has been modified you can press F5 or "Debug -> Start Debugging" to launch your application in the Emulator, which will look like the screen below.

3.4 Hello World (Gadgeteer)

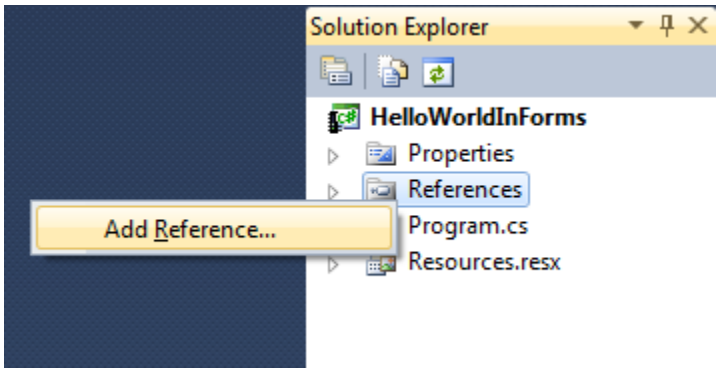
Unlike NETMF applications you may be used to *Tinkr* doesn't rely on WPF, nor do you need to Flush a Bitmap in order to update your screen. Instead, you will need to create and activate a Form.

You can have as many Forms as you wish however, only one can be active at a time. Activating a Form is simple and you can switch between Forms with a single line of code.

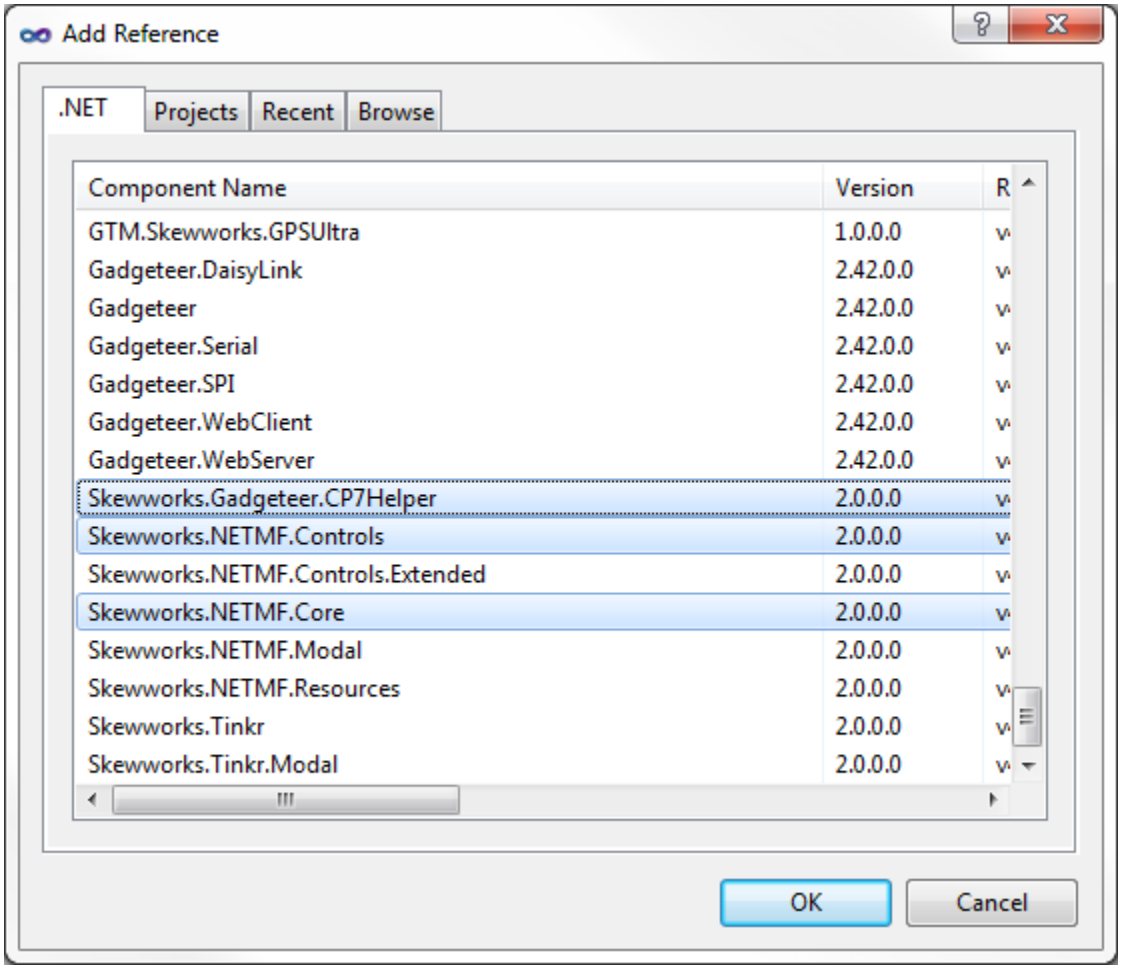
To begin, let's create a new .NET Gadgeteer Application project in Visual Studio; we'll name it HelloWorldInForms.



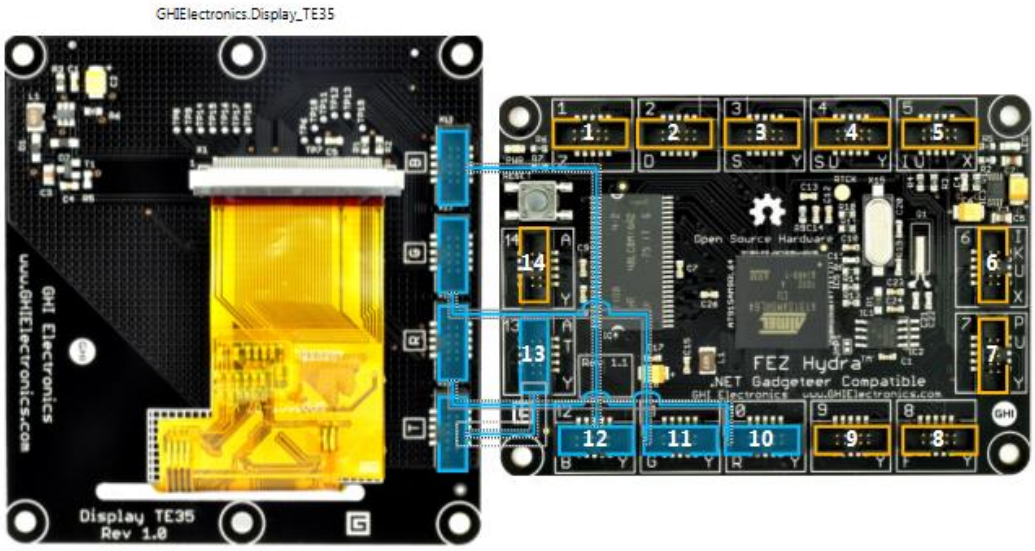
Next we need to add references to *Tinkr*. To do this, right-click on the "References" folder to the right and select "Add Reference".



Next select the *Tinkr* references as seen below



Once you have your references added, connect a display module to your Gadgeteer Mainboard in the designer, like below.



display_TE35

Finally, you'll need to open your Program.cs file and modify it with the following code.

```
using System;
using System.Threading;

using Skewworks.NETMF;
using Skewworks.NETMF.Controls;
using Skewworks.Gadgeteer.CP7Helper;

namespace HelloWorldInForms
{
    public partial class Program
    {
        // This method is run when the mainboard is powered up or reset.
        void ProgramStarted()
        {
            // Comment line below if you're using CP7 or similar
            Core.Initialize(TouchCollection.NativeSingleTouch);

            // Uncomment lines below if you're using CP7
            //Core.Initialize(TouchCollection.ManualOrMultiTouch);
            //CP7TouchHelper CP7 = new CP7TouchHelper();
            //display_CP7.ScreenPressed += (Display_CP7 sender, Display_CP7.TouchStatus
TT) => CP7.CP7Pressed(new Point(TT.touchPos[0].xPos, TT.touchPos[0].yPos));
            //display_CP7.screenReleased += (Display_CP7 sender) => CP7.CP7Released();
            //display_CP7.backPressed += (Display_CP7 sender) => CP7.CP7BackPressed();
            //display_CP7.homePressed += (Display_CP7 sender) => CP7.CP7HomePressed();
            //display_CP7.menuPressed += (Display_CP7 sender) => CP7.CP7MenuPressed();
```

```
            // Gadgeteer doesn't like infinite sleeps on the main thread, so start a new
one
            new Thread(HelloWorld).Start();
        }

        void HelloWorld()
        {
            // Create the form
            Form frmMain = new Form("frmMain");

            // Create a label
            Label lbl1 = new Label("lbl1", "Hello World!", Fonts.Droid12, 4, 4);

            // Add label to the form
            frmMain.AddChild(lbl1);

            // Activate the form
            Core.ActiveContainer = frmMain;

            // Make sure application doesn't exit
            Thread.Sleep(Timeout.Infinite);
        }

    }
}
```

Once your Program.cs has been modified you can press F5 or “Debug -> Start Debugging” to launch your application. Be sure you have a device connect, Gadgeteer applications *do not* work well in the Emulator.

3.5 Working with Multiple Forms

As mentioned previously, *Tinkr* allows for multiple forms in applications. You can work with multiple forms in a couple of ways; the most memory efficient of which is to create forms upon request.

In order to show how to do this, please open your HelloWorldInForms project.

If you're working with pure NETMF please update your Main method to look like the one below:

```
        public static void Main()
        {
            // Initialize Tinkr in normal mode
            // This allows for native touch collection
            Core.Initialize(TouchCollection.NativeSingleTouch);

            // Display the first form
            ShowForm1();

            // Make sure application doesn't exit
            Thread.Sleep(Timeout.Infinite);
        }
```

If you're using Gadgeteer, please update your HelloWorld method with the following:

```
        void HelloWorld()
        {
            // Initialize Tinkr in normal mode
            // This allows for native touch collection
            Core.Initialize(TouchCollection.NativeSingleTouch);

            // Display the first form
            ShowForm1();

            // Make sure application doesn't exit
            Thread.Sleep(Timeout.Infinite);
        }
```

For both projects you will need to add the following methods, please note that if you're using Gadgeteer you should **remove** the Static keyword.

```
        static void ShowForm1()
        {
            // Cleanup between forms
            CleanSwitch();

            // Create the form
            Form frmMain = new Form("frmMain");

            // Create a label
            Label lbl1 = new Label("lbl1", "Hello World!", Fonts.Droid12Bold, 4, 4);

            // Create a button
            Button btn1 = new Button("btn1", "Next >", Fonts.Droid11, 0, 0);

            // The button was automatically sized for us on creation
            // Now lets put it in the bottom-right corner
            btn1.X = frmMain.Width - btn1.Width - 4;
            btn1.Y = frmMain.Height - btn1.Height - 4;

            // Tell the button to take us to the next form
            btn1.Tap += (object sender, point e) => ShowForm2();

            // Add both items to the form
            frmMain.AddChild(lbl1);
            frmMain.AddChild(btn1);

            // Activate the form
            Core.ActiveContainer = frmMain;
        }

        static void ShowForm2()
        {
            // Cleanup between forms
            CleanSwitch();

            // Create the Form
            Form frmTwo = new Form("frmTwo", Colors.LightBlue);

            // Let's have some fun on this form
            // Create a checkbox to control the button
            Checkbox chk1 = new Checkbox("chk1", string.Empty, Fonts.Droid11, 4, 4);

            // Controls can be added in any order
            // So this time we'll add them as they're created
            frmTwo.AddChild(chk1);

            // You can also instantly add a control, like so
            frmTwo.AddChild(new Label("lbl1", "Enable Back Button", Fonts.Droid11,
chk1.Width + 8, chk1.Height / 2 - Fonts.Droid11.Height / 2 + 4));
```

```
            // Create a back button
            Button btn1 = new Button("btn1", "< Back", Fonts.Droid11, 4, 0);

            // Controls are rendered as soon as they are added
            // But the form isn't active so renders won't occur
            frmTwo.AddChild(btn1);

            // This time let's add it to the lower left
            btn1.Y = frmTwo.Height - btn1.Height - 4;

            // Disable the button
            btn1.Enabled = false;

            // Subscribe to the Tap event
            btn1.Tap += (object sender, point e) => ShowForm1();

            // Have the checkbox control the btn
            chk1.CheckChanged += (object sender, bool value) => btn1.Enabled =
chk1.Value;

            // Activate the Form
            Core.ActiveContainer = frmTwo;
        }

        /// <summary>
        /// Calling this method helps ensure Garbage Collection kicks in
        /// If we didn't do this we'd run out of memory if we switch forms rapidly
        /// You could also use static forms so they weren't created and destroyed
everytime.
        /// Doing so would eliminated the need for this method
        /// </summary>
        static void CleanSwitch()
        {
            // See if we already have a form active
            if (Core.ActiveContainer != null)
            {
                // Remove the active form
                Core.ActiveContainer = null;

                // Now force GC to get rid of the form
                Debug.GC(true);
            }
        }
```

Once your Program.cs has been modified you can press F5 or “Debug -> Start Debugging” to launch your application.

Again, if you’re using Gadgeteer you will need to have a device attached.

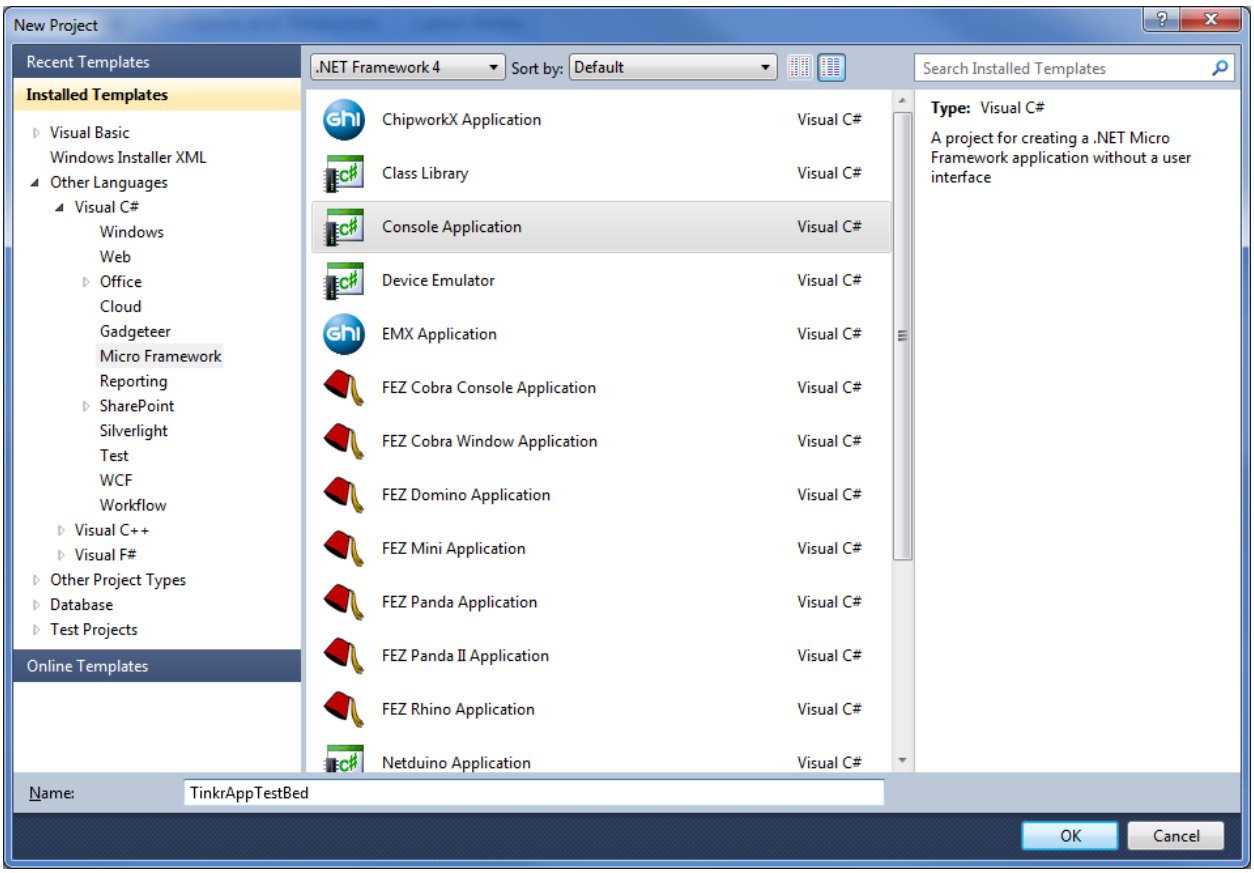
4 Getting Started with Applications

Tinkr has built-in application support; this allows you to run multiple NETMF or Gadgeteer applications simultaneously. In order for an application to be compatible with Tinkr it must be marked Serializable and inherit from Skewworks.NETMF.Applications.NETMFApplication.

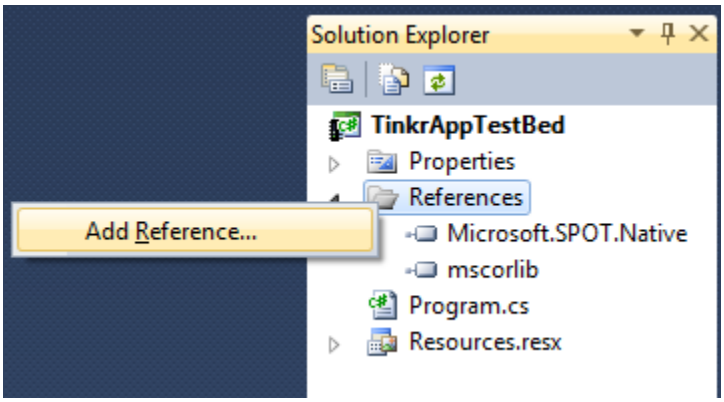
These applications can be launched from an embedded resource or from an external source such as SD card or USB drive. You can even debug applications in the Emulator by creating a testbed.

4.1 Creating a Testbed

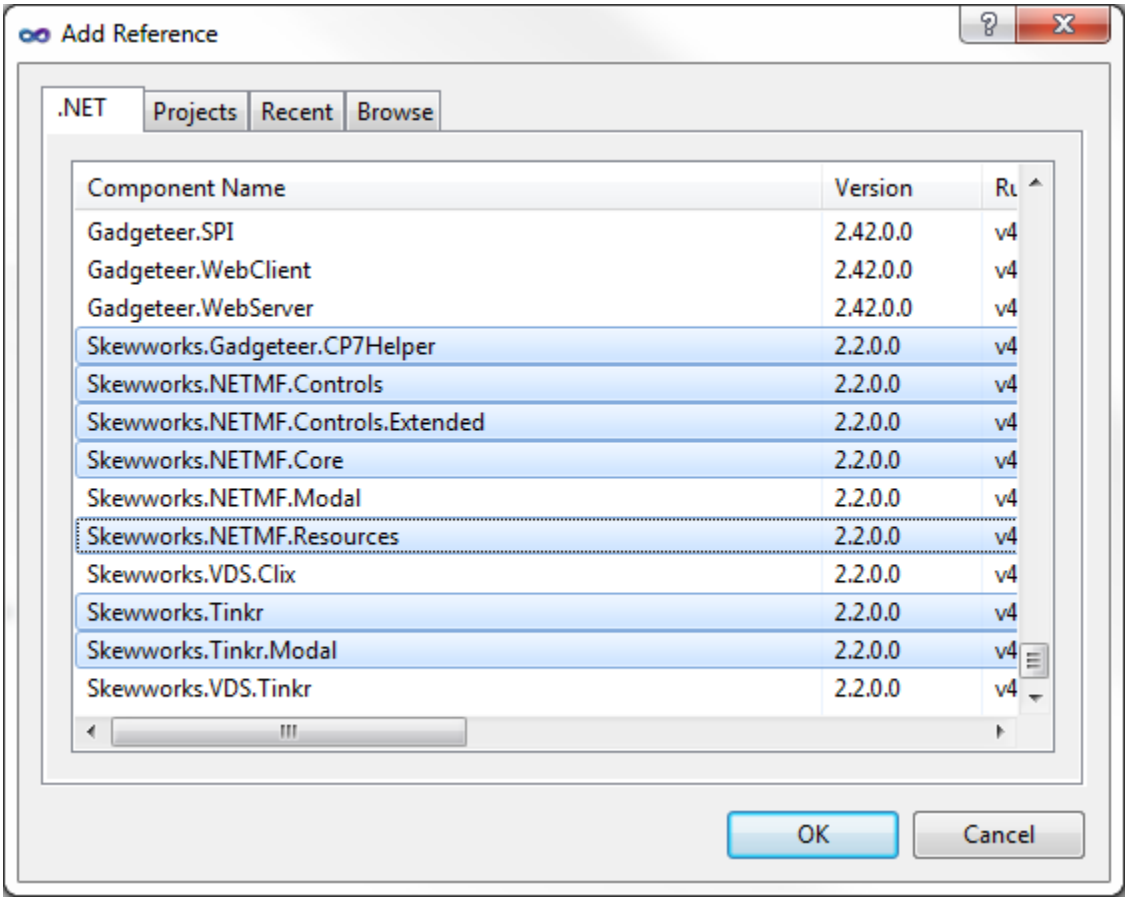
First let's start by creating a new pure NETMF application. Name your solution TinkrAppTestBed as seen below.



Next we'll add a reference to all of the *Tinkr* DLLs. To do this, right-click on the "References" folder to the right and select "Add Reference".



Next select the *Tinkr* references as seen below as well a **Microsoft.SPOT.Graphics**



Next we'll need to create a compatible application to run. Create a new class called SampleApp.cs and place the following code inside.

```
using System;

using Microsoft.SPOT;

using Skewworks.NETMF.Applications;
using Skewworks.NETMF.Graphics;

using Skewworks.NETMF;
using Skewworks.NETMF.Controls;

using Skewworks.Tinkr;
using Skewworks.Tinkr.Controls;
using Skewworks.Tinkr.Modal;
using Skewworks.Tinkr.Resources;


namespace TinkrAppTestBed
{
    [Serializable]
    public class SampleApp : NETMFApplication
    {

        #region Properties

        public override ApplicationDetails ApplicationDetails
        {
            get { return new ApplicationDetails("SampleApp", "Sample Tinkr Application",
"Skewworks", "2012 Skewworks.com", "1.0.0.0"); }
        }

        #endregion

        #region Public Methods

        public override void Main()
        {
            // Create form
            Form frmApp = new Form("frmApp");

            // Create label
            frmApp.AddChild(new Label("lbl1", "Hello World! I'm an application running in
the AppDomain named '" + AppDomain.CurrentDomain.FriendlyName + "'", Fonts.Droid12, 4, 4,
frmApp.Width - 8, frmApp.Height - 8));

            // Create a button to terminate
            Button btn1 = new Button("btn1", "Terminate Application", Fonts.Droid12, 4,
0);
            btn1.Y = frmApp.Height - btn1.Height - 4;

            // This will cause the application to close on button tap
```

```
            btn1.Tap += (object sender, point e) =>
Core.TerminateApplication(this.ThreadId);

            frmApp.AddChild(btn1);

            // Activate form
            Core.ActiveContainer = frmApp;
        }

        public override string SendMessage(object sender, string message, object args =
null)
        {
            Prompt.Show("Application.SendMessage", "Message Received:\n" + message,
Fonts.Droid16, Fonts.Droid12);
            return string.Empty;
        }

        public override void Terminating()
        {
            Debug.Print("Application.Terminating");
        }

        #endregion

    }
}
```

Now that we have our application we need to update our program to use it. Since the point of the Testbed is to allow us to debug the application we won't want to call Core.LaunchApplication. Therefore we're going to need to do a few things that *Tinkr* normally takes care of for us; namely ensuring the application is launched in a new thread, registering the application, and raising a fake ApplicationLaunched event (should you require the use of it in testing).

Open your Program.cs and replace the code with what's below.

```
using System;
using System.Threading;

using Microsoft.SPOT;

using Skewworks.NETMF.Applications;
using Skewworks.NETMF.Graphics;

using Skewworks.NETMF;
using Skewworks.NETMF.Controls;

using Skewworks.Tinkr;
using Skewworks.Tinkr.Controls;
using Skewworks.Tinkr.Modal;

namespace TinkrAppTestBed
{
    public class Program
    {
        public static void Main()
        {
            // Initialize Tinkr
            Core.Initialize(TouchCollection.NativeSingleTouch);

            // Subscribe to events
            Core.Instance.ApplicationClosed += (object sender, ApplicationDetails
appDetails) => new Thread(Testbed).Start();

            // Launch
            new Thread(Testbed).Start();
        }

        static void Testbed()
        {
            // Create the main form
            Form frmMain = new Form("frmMain");

            // Add a button
            Button btn1 = new Button("btn1", "Launch Sample App", Fonts.Droid16, 4, 4);
            btn1.Tap += (object sender, point e) => new Thread(LaunchApp).Start();
            frmMain.AddChild(btn1);

            // Activate the form
            Core.ActiveContainer = frmMain;
        }
```

```
        static void LaunchApp()
        {
            // Testbed launching is currently unavailable.
        }

    }
}
```

4.2 Deploying Real Applications

When you've thoroughly tested your application and are ready to deploy you can simply remove Program.cs and change your project type to "Class Library" and rebuild the application.

Once done, locate your project's .pe file in the bin directory of the project; it will be in both the le and be subdirectories of bin, please choose the one that is proper for your device.

You do **not** need to copy any of the *Tinkr* .pe files, as they will already be loaded. However, if you have any other dependencies you should copy those as well.

4.3 App.Config Files

If your dependencies (should there be any) require loading in a specific order, for example if one dependency should also have a dependency, you will need to create an app.config file. This is a simple text file that lists each of the files your application needs to be run, in the order they need to be loaded, separated by commas.

You can also use this with relative paths so your dependencies don't have to be in the same directory as your main .pe file. This can help avoid confusion on the user's part and also allows you to change the extension of your dependencies.

4.4 Launching Applications

Outside of the Testbed environment applications can be launched with a single line of code. For applications embedded in resources it looks as follows:

```
Core.LaunchApplication(Resources.GetBytes(Resources.BinaryResources.SampleApp), null);
```

And for applications being launched externally it looks like this:

```
Core.LaunchApplication("\\SD\\Apps\\SampleApp\\SampleApp.pe", null);
```

The second argument in each call is an optional array of startup arguments that you can send to the application.

5 API Reference

A complete API reference is available here: http://www.skewworks.com/api

6 Using Visual Design Studio Files

To use a compiled file from Visual Design Studio, simply place the file on your USB drive or [micro]SD card, then load the file.

Example:

```
Core.ActiveContainer = Design.LoadForm("\\ROOT\\test.bin");
```