# Mono on F&S Boards

Version 0.2
(2023-07-26)

# About This Document

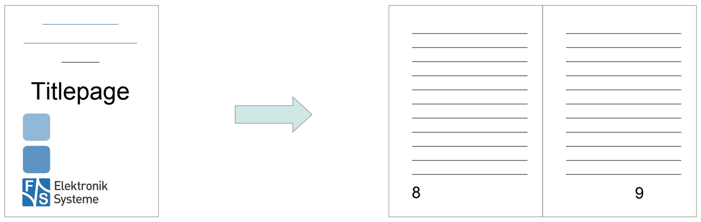
This document describes how to run .NET framework applications via Mono on F&S Boards.

### Remark

The version number on the title page of this document is the version of the document. It is not related to the version number of any software release! The latest version of this document can always be found at http://www.fs-net.de.

### How To Print This Document

This document is designed to be printed double-sided (front and back) on A4 paper. If you want to read it with a PDF reader program, you should use a two-page layout where the title page is an extra single page. The settings are correct if the page numbers are at the outside of the pages, even pages on the left and odd pages on the right side. If it is reversed, then the title page is handled wrongly and is part of the first double-page instead of a single page.



### Typographical Conventions

We use different fonts and highlighting to emphasize the context of special terms:

`File names`

*Menu entries*

```
Board input/output
```

`Program code`

```
PC input/output
```

```
Listings
```

```
Generic input/output
```

`Variables`

# History

| Date | V | Platform | A,M,R | Chapter | Description | Au |
|---|---|---|---|---|---|---|
| 2020-05-18 | 0.1 | ALL | A | ALL | Initial version | PG |
| 2020-06-09 | 0.2 | ALL | M | ALL | Correct typos and footer | PG |
| 2020-06-09 | 0.2 | ALL | A | 2 | Add System requirements chapter | PG |
| 2020-06-09 | 0.2 | ALL | M | 8 | Add  GTK# tutorial directly into this document | PG |
|  |  |  |  |  |  |  |

| | |
|---|---|
| V | Version |
| A,M,R | Added, Modified, Removed |
| Au | Author |

# Table of Contents

# 1 Introduction

This document describes how to run .Net framework application to F&S-Linux boards, using the Mono framework.

Mono is an open source implementation of the .Net framework, sponsored by Microsoft. It currently supports most of the .NET libraries up to version 4.7 except WPF, WWF, and with limited WCF and limited ASP.NET async stack. For detailed compatibility information please see the official website of the mono project.

https://www.mono-project.com/docs/about-mono/compatibility/

https://www.mono-project.com/docs/getting-started/application-portability/

Please notice, that there are no native libraries to access Linux hardware peripheries like I2C or SPI in Mono. If you need to access these in your application, please contact F&S for support to create these libraries.

This document assumes basic knowledge of using Linux on F&S boards. For a detailed introduction please see the *Linux on F&S Boards.pdf* from the document section of your F&S board at

https://www.fs-net.de/

# 2 System requirements

The mono rootfs images need a lot of disk space, so make sure your flash memory is big enough:

**Buildroot**

| Image type | Image size |
|---|---:|
| Ubifs Image (Nand Flash) | 210 MB |
| Ext4 Image (eMMC) | 480 MB |

**Yocto**

| Image type | Image size |
|---|---:|
| Ubifs Image (Nand Flash) | 210 MB |
| Ext4 Image (eMMC) | 520 MB |

# 3 Tested Software versions

The Software used in this document has been tested with the following versions.

| Software | Version |
|---|---|
| **Development Machine** | F_S_Development_Machine-Fedora_27_V1.2 |
| **Buildroot** | |
| fsimx6 | fsimx6-B2020.04 |
| **Yocto** | |
| fsimx6 | fsimx6-Y2020.03 |
| fsimx8mm | fsimx8mm-Y2020.05 |
| **Microsoft Visual Studio** | Community 2019 Version 16.5.4 |
| **MonoRemoteDebugger** | 1.5.2 |

# 4 Compiling the Mono images

F&S supports the build environments Buildroot and Yocto to build the system software.

This chapter describes how to build a root filesystem with preinstalled Mono binaries, using Buildroot or Yocto.

For a detailed description how to setup and use the build environments, please see the document *Linux on F&S Boards* chapter *Compiling the System Software.*

Please note that Buildroot currently only supports Mono for fsimx6/7-Boards. For fsimx8m/mm-Boards use Yocto.

## 4.1 Compiling the Mono images with Buildroot

1. Get the latest F&S-Buildroot release and execute the `setup-buildroot` script to install Buildroot to your development machine. Follow the instructions.
2. Build the standard defconfig of your machine. Run
   ```
   make fs<YOUR_MACHINE>_std_defconfig
   ```
   in your Buildroot main directory.
3. Open the configuration menu in your buildroot directory
   ```
   make menuconfig
   ```
4. Activate the mono package at
   `Target packages -> Interpreter languages and scripting-> mono`
   if you are using a GUI also activate libgdiplus
   `Target packages -> Libraries -> Graphics -> libgdiplus`
5. Mono needs about 170MB of disk space, so the rootfile system may get too large for .ext image. You can increase the .ext image size by setting
   `Filesystem images -> ext2/3/4 root filesystem -> exact size`
   in the configuration menu to at least 400000
6. Build buildroot
   ```
   make -j4
   ```
7. The build output can be found at
   `output/images/`

## 4.2 Compiling the Mono images with Yocto

1. Get the latest F&S- Yocto release and execute the `setup-yocto` script to install Yocto to your development machine. Follow the instructions.
2. Setup the build environment for your machine.
   Fsimx8 boards need a different DISTRO than fsimx6/7 boards, so please run the respective command in the main directory of your Yocto installation.
   **Fsimx6/7**
   ```
   DISTRO=fus-imx-x11 MACHINE=<YOUR_MACHINE> . fus-setup-release.sh
   ```
   **Fsimx8**
   ```
   DISTRO=fus-imx-xwayland MACHINE=<YOUR_MACHINE> . fus-setup-release.sh
   ```
3. Now run
   ```
   bitbake fus-image-mono
   ```
   to compile the images.
4. The build output can be found at
   `tmp/deploy/images/<YOUR_MACHINE>/`

# 5 Executing .NET applications on F&S boards using Mono

This chapter describes how to execute .NET applications on F&S boards and how to handle possible failures.

## 5.1 Running an application

1. Install the Mono images to your board. You will need to install kernel, device tree and root filesystem. The different ways of how to install the images are described in the document *Linux on F&S Boards* chapters *Image Download* and *Image Storage.*
2. Boot Linux and transfer your .NET application to the board. You can transfer it via network using the tftp command or use an USB stick. See *Linux on F&S Boards* chapter *Using the Standard System and Devices.*
3. Execute the .NET application using
   `mono </path/to/your/application.exe>`

## 5.2 Handling possible failures

Depending on your application you may get error messages for missing libraries, e.g.:

```
System.DllNotFoundException: libcairo-2.dll
```

This might have several reasons.

### 5.2.1 Missing entry in the DllMap

Mono uses DllMaps to map Windows library names (.dll) to Linux library names (.so).
The DllMaps are specified either in the global mono configuration file at

```
/etc/mono/config
```

Or at the assembly configuration file of your application at

```
<App_Name>.exe.config
```

A DllMaps entry looks like this:

```
<configuration>
…
   <dllmap dll="<DLL_NAME>" target="<SO_NAME>" os="<OS>"/>
…
</configuration>
```

with

- DLL_NAME = Name of the .dll needed by the application
- SO_NAME = Name of the .so library, the dll is mapped to. Mono checks the paths /lib/ and /usr/lib/ for <SO_NAME>.
- OS = Name of the operating system for which the mapping should be applied (e.g. linux, windows, osx...)

To check for a missing entry in the DllMap open the config file and look for the DLL_NAME

```
vi /etc/mono/config
```

If there is no exact entry for the missing .dll you can try to add one. First make sure your rootfs is mounted read-writeable

```
mount -o remount,rw /
```

Then open the config again and add a new entry, for example:

```
<dllmap dll="libcairo-2.dll " target="libcairo.so.2" os="linux"/>
```

To get the name of the respective Linux library you can search the paths /lib/ and /usr/lib.

```
find /lib /usr/lib -name "libcairo*"
```

### 5.2.2 Missing .so library

Sometimes the DllMap entry is just slightly different from the name of the installed library (for example libcairo.so.2.0 instead of libcairo.so.2). Just add a new entry with the correct names to the config file.

If you cannot find a fitting .so-library for your needed .dll, you may have to install it first to your root filesystem. How to do this depends on your build environment.

**Buildroot**

1. Open the configuration menu
   ```
   make menuconfig
   ```
2. Type “/” to search for a package. If you find it, activate it, save and exit.
3. Rebuild your root filesystem. Some packages may have dependencies that cause the build to fail, so you may have to build from scratch.
   ```
   make clean
   ```
   ```
   make -j4
   ```

**Yocto**

1. Go to the `sources/` directory in the main directory of your Yocto installation.
2. Search for your package name using the find command
   ```
   find -name “your_package_name*”
   ```
3. If you find it, add the following line to
   `conf/local.conf`

of your Yocto build directory:

```
CORE_IMAGE_EXTRA_INSTALL += "<your_package_name>"
```

4. Rebuild your root filesystem.

```
bitbake fus-image-mono
```

If you cannot find any packages for your missing library, please contact F&S.

### 5.2.3 Missing .dll library

Some .dll libraries are just shipped with the .NET application. Make sure you copied all necessary files to the board.

# 6 Using Visual Studios to remote debug Mono apps on F&S Boards

You can use Visual Studios to program and compile your .NET applications as usual, but if you want to debug your application while running on the F&S board, some additional preparations are needed.

The most comfortable way is to use the Visual Studios community extension MonoRemoteDebugger. It consists of a client that has to be installed as Visual Studios extension, and a server, which has to be run on the F&S board. The MonoRemoteDebugger enables the transfer of the application to the board via SSH and basic debugging, like stepping through the program code and read out some variables.

## 6.1 Installing MonoRemoteDebugger

**Visual Studios**

1. Click Extensions -> Manage Extensions and search for “MonoRemoteDebugger”.
2. Install it.

**F&S Board**

1. Download the latest Release of MonoRemoteDebugger.Server.zip from https://github.com/techl/MonoRemoteDebugger/releases
2. Transfer it to the F&S board via tftp or UBS stick
3. On the board, unpack it to the `/tmp/` directory using
   ```
   unzip MonoRemoteDebugger.Server.zip -d /tmp/
   ```

*Remark: In future releases the MonoRemoteDebugger.Server will be installed by Buildroot and Yocto.*

## 6.2 Using MonoRemoteDebugger

**F&S Board**

1. Activate the network. If your board is in a network with a DHCP server you can get an IP address by running
   ```
   udhcpc
   ```
   If not, you can a activate the network by running
   ```
   ifconfig eth0 up <IPADDR>
   ```
   Note the boards IP address
2. Run the MonoRemoteDebugger.Server
   ```
   mono /tmp/MonoRemoteDebugger.Server.exe
   ```

**Visual Studios**

1. Click Extensions -> MonoRemoteDebugger -> Debug with Mono (remote)
2. Enter the boards IP address into the field `Remote-IP`
3. Depending on your Application size you may have to increase the `Timeout` value
4. Click the `Connect` button. The application will be transferred to your board and the Visual Studios debugger will stop at the first breakpoint

## 6.3 Known issues

**Missing DllMaps**

Sometimes the RemoteDebugger does not find DllMaps even though they are listed in /etc/mono/config.

You can add these DllMaps to the `app.config` file of your Visual Studios Project.

**Debug output is too large**

The MonoRemoeDebugger transfers all output generated by Visual Studio the Board. Sometimes .dll libraries get build that are already installed by mono as shared library. These libraries can be pretty large and slow down the debugging process.

You can prevent Visual Studios from building unneeded libraries by expanding the `References` entry of your Project in your Solution Explorer and right click on n reference. Select `Properties` and set the entry `Copy Locale` to `False.`

Now the respective .dll will not be built with the project anymore.

# 7 Executing Xamarin applications on F&S boards using Mono

Xamarin extents the .NET platform with tools and libraries to create cross-platform applications. With the community driven *Xamarin.Forms GTK Backend* project, it is now possible to run graphical Xamarin.Forms applications on Linux systems, using the Mono GTK# backend.

A good tutorial on how to add the GTK# backend to Xamarin apps can be found at

https://docs.microsoft.com/de-de/xamarin/xamarin-forms/platform/other/gtk?tabs=windows

A PDF version of this website has also been attached to this document.

To run Xamarin.Forms on F&S- boards you need to add the *gtk-sharp* package to your rootfile system.

This is currently only supported for the Yocto build environment. If you need to build with Buildroot, please contact F&S.

**Yocto**

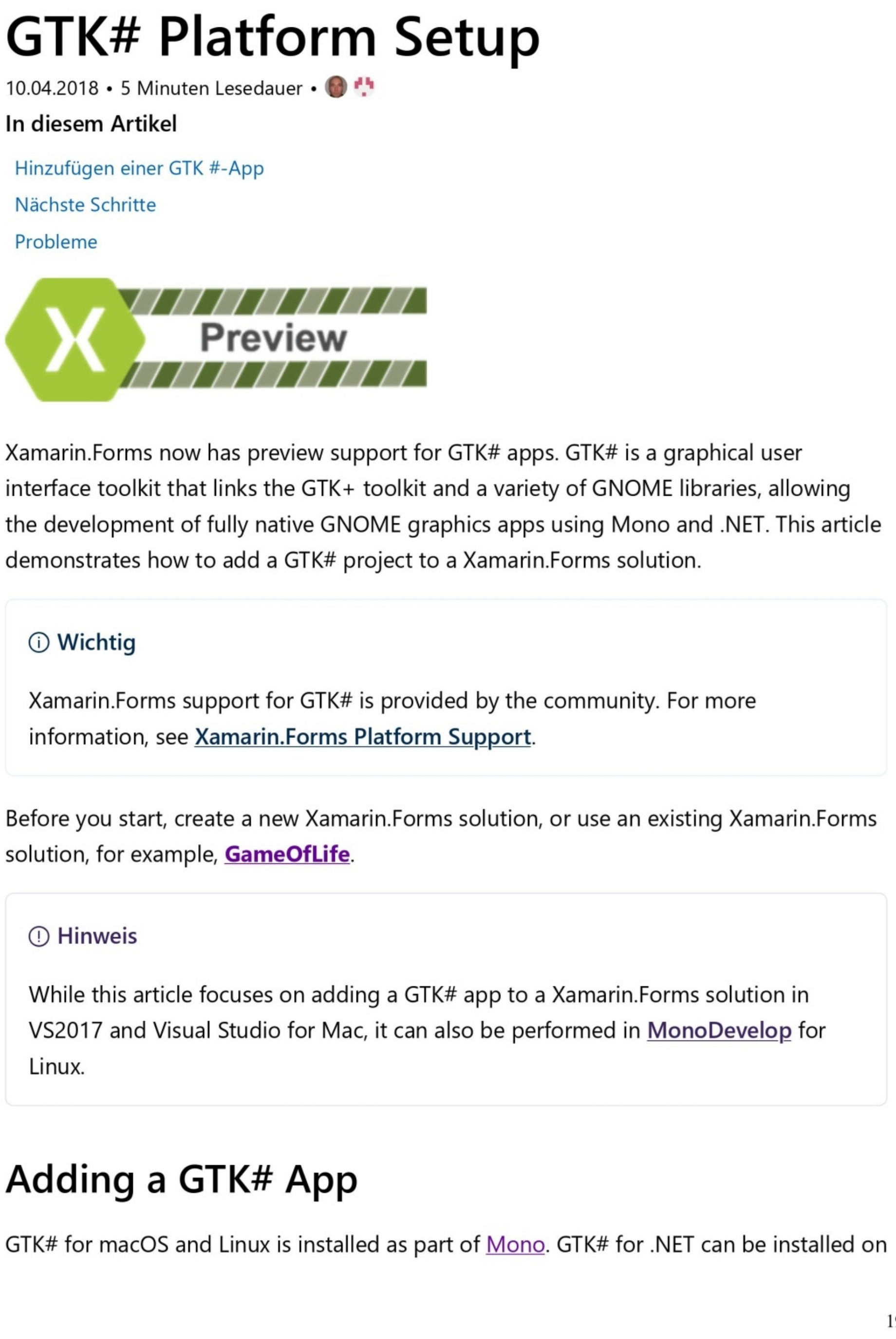
To add the gtk-sharp package to your rootfile system add
`CORE_IMAGE_EXTRA_INSTALL += " gtk-sharp"`
to `conf/local.conf` in your Yocto build directory.

# 8 Appendix

## 8.1 Add GTK# to Xamarin projects

Source :

*https://docs.microsoft.com/de-de/xamarin/xamarin-forms/platform/other/gtk?tabs=windows*



## GTK# Platform Setup

10.04.2018 • 5 Minuten Lesedauer •

**In diesem Artikel**

Preview

Xamarin.Forms now has preview support for GTK# apps. GTK# is a graphical user interface toolkit that links the GTK+ toolkit and a variety of GNOME libraries, allowing the development of fully native GNOME graphics apps using Mono and .NET. This article demonstrates how to add a GTK# project to a Xamarin.Forms solution.

> ⓘ **Wichtig**
>
> Xamarin.Forms support for GTK# is provided by the community. For more information, see Xamarin.Forms Platform Support.

Before you start, create a new Xamarin.Forms solution, or use an existing Xamarin.Forms solution, for example, **GameOfLife**.

> ⓘ **Hinweis**
>
> While this article focuses on adding a GTK# app to a Xamarin.Forms solution in VS2017 and Visual Studio for Mac, it can also be performed in MonoDevelop for Linux.

### Adding a GTK# App

GTK# for macOS and Linux is installed as part of Mono. GTK# for .NET can be installed on

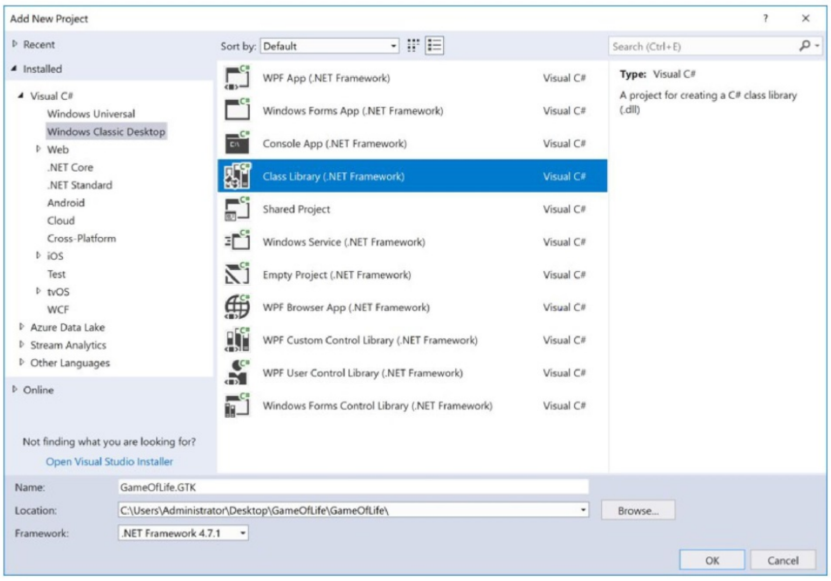
1 von 8 19.05.2020, 15:39

Windows with the GTK# Installer.

Visual Studio | Visual Studio for Mac

Follow these instructions to add a GTK# app that will run on the Windows desktop:

1. In Visual Studio 2019, right-click on the solution name in **Solution Explorer** and choose **Add > New Project...**.
2. In the **New Project** window, at the left select **Visual C#** and **Windows Classic Desktop**. In the list of project types, choose **Class Library (.NET Framework)**, and ensure that the **Framework** drop-down is set to a minimum of .NET Framework 4.7.
3. Type a name for the project with a **GTK** extension, for example **GameOfLife.GTK**. Click the **Browse** button, select the folder containing the other platform projects, and press **Select Folder**. This will put the GTK project in the same directory as the other projects in the solution.


Add New Project
Recent
Installed
Visual C#
Windows Universal
Windows Classic Desktop
Web
.NET Core
.NET Standard
Android
Cloud
Cross-Platform
iOS
Test
tvOS
WCF
Azure Data Lake
Stream Analytics
Other Languages
Online
Not finding what you are looking for?
Open Visual Studio Installer
Sort by: Default
Search (Ctrl+E)
WPF App (.NET Framework) Visual C#
Windows Forms App (.NET Framework) Visual C#
Console App (.NET Framework) Visual C#
Class Library (.NET Framework) Visual C#
Shared Project Visual C#
Windows Service (.NET Framework) Visual C#
Empty Project (.NET Framework) Visual C#
WPF Browser App (.NET Framework) Visual C#
WPF Custom Control Library (.NET Framework) Visual C#
WPF User Control Library (.NET Framework) Visual C#
Windows Forms Control Library (.NET Framework) Visual C#
Browse...
OK
Cancel

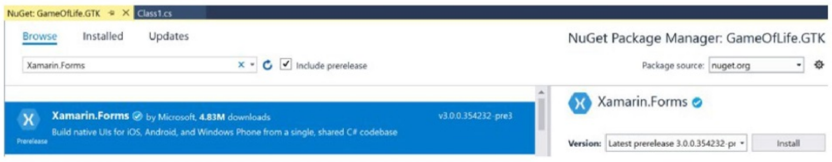
Type: Visual C#
A project for creating a C# class library (.dll)

Name: GameOfLife.GTK
Location: C:\Users\Administrator\Desktop\GameOfLife\GameOfLife\
Framework: .NET Framework 4.7.1

Press the **OK** button to create the project.

4. In the **Solution Explorer**, right click the new GTK project and select **Manage NuGet Packages**. Select the **Browse** tab, and search for **Xamarin.Forms** 3.0 or

greater.



NuGet Package Manager: GameOfLife.GTK

Xamarin.Forms ✓ Include prerelease Package source: nuget.org

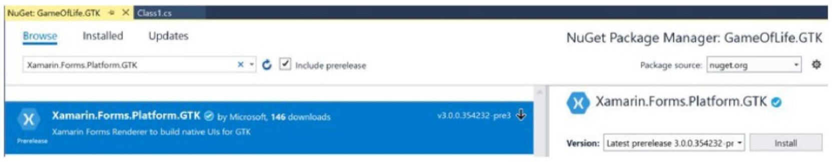
Xamarin.Forms by Microsoft, **4.83M** downloads v3.0.0.354232-pre3
Build native UIs for iOS, Android, and Windows Phone from a single, shared C# codebase
Prerelease

Xamarin.Forms

**Version:** Latest prerelease 3.0.0.354232-pr

Select the package and click the **Install** button.

5. Now search for the **Xamarin.Forms.Platform.GTK** 3.0 package or greater.



NuGet Package Manager: GameOfLife.GTK

Xamarin.Forms.Platform.GTK ✓ Include prerelease Package source: nuget.org

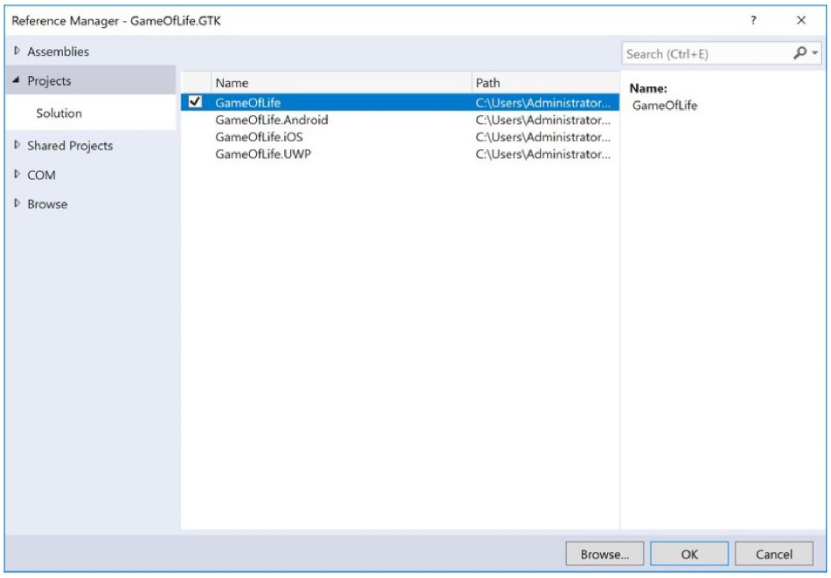
Xamarin.Forms.Platform.GTK by Microsoft, **146** downloads v3.0.0.354232-pre3
Xamarin Forms Renderer to build native UIs for GTK
Prerelease

Xamarin.Forms.Platform.GTK

**Version:** Latest prerelease 3.0.0.354232-pr

Select the package and click the **Install** button.

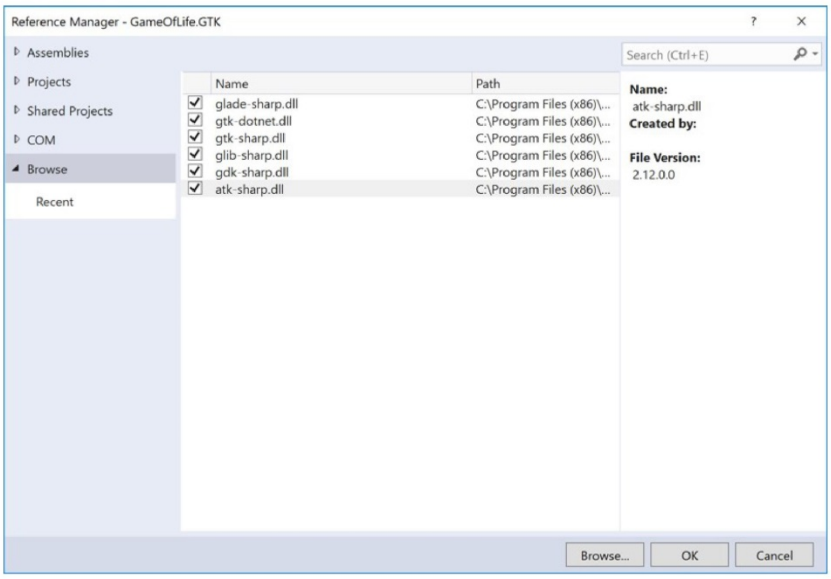
6. In the **Solution Explorer**, right-click the solution name and select **Manage NuGet Packages for Solution**. Select the **Update** tab and the **Xamarin.Forms** package. Select all the projects and update them to the same Xamarin.Forms version as used by the GTK project.
7. In the **Solution Explorer**, right-click on **References** in the GTK project. In the **Reference Manager** dialog, select **Projects** at the left, and check the checkbox adjacent to the .NET Standard or Shared project:



| ✓ | Name | Path |
|---|---|---|
| ✓ | GameOfLife | C:\Users\Administrator... |
| | GameOfLife.Android | C:\Users\Administrator... |
| | GameOfLife.iOS | C:\Users\Administrator... |
| | GameOfLife.UWP | C:\Users\Administrator... |

**Name:**
GameOfLife

8. In the **Reference Manager** dialog, press the **Browse** button and browse to the **C:\Program Files (x86)\GtkSharp\2.12\lib** folder and select the **atk-sharp.dll**, **gdk-sharp.dll**, **glade-sharp.dll**, **glib-sharp.dll**, **gtk-dotnet.dll**, **gtk-sharp.dll** files.

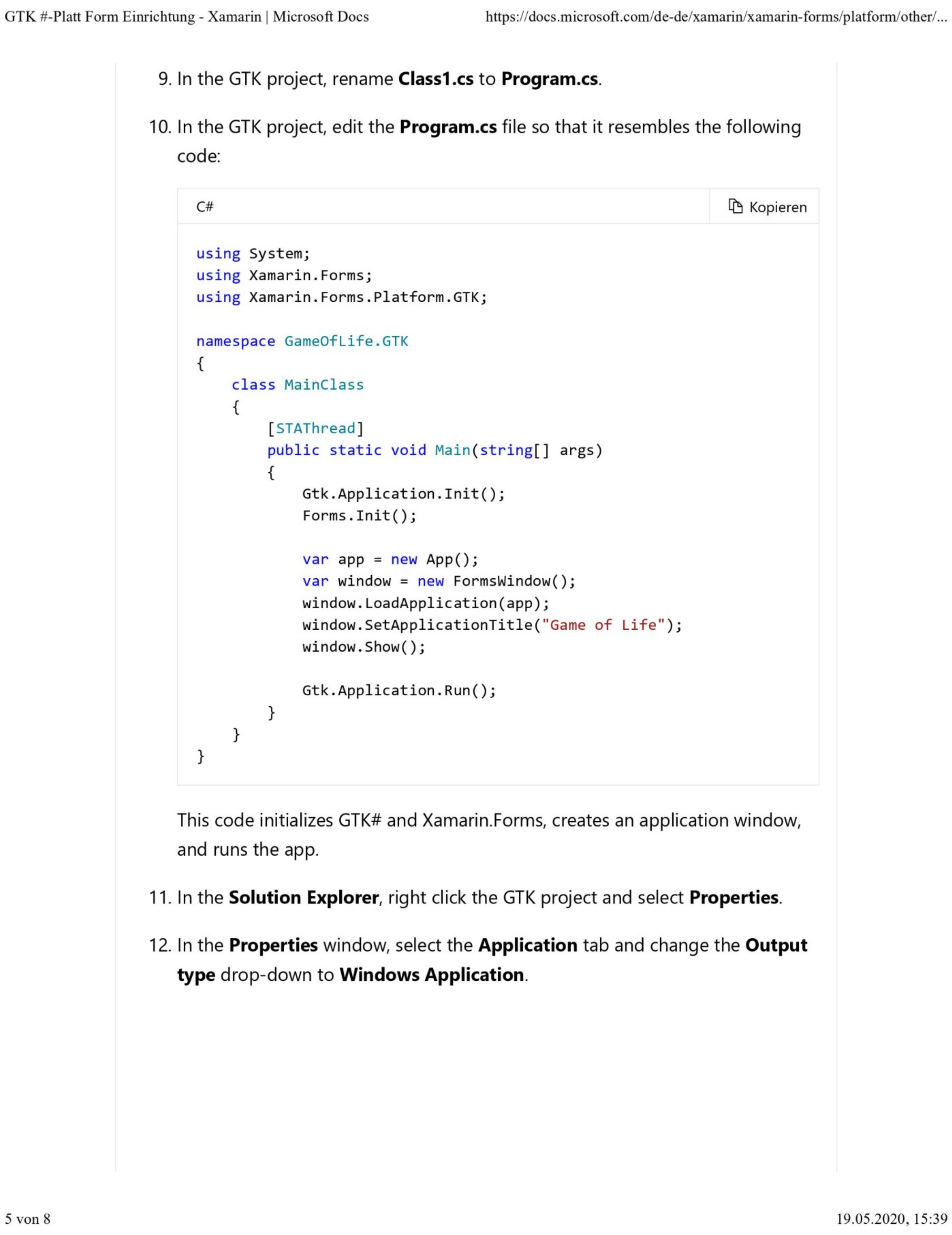


| ✓ | Name | Path |
|---|---|---|
| ✓ | glade-sharp.dll | C:\Program Files (x86)\... |
| ✓ | gtk-dotnet.dll | C:\Program Files (x86)\... |
| ✓ | gtk-sharp.dll | C:\Program Files (x86)\... |
| ✓ | glib-sharp.dll | C:\Program Files (x86)\... |
| ✓ | gdk-sharp.dll | C:\Program Files (x86)\... |
| ✓ | atk-sharp.dll | C:\Program Files (x86)\... |

**Name:**
atk-sharp.dll
**Created by:**

**File Version:**
2.12.0.0

Press the **OK** button to add the references.



GTK #-Platt Form Einrichtung - Xamarin | Microsoft Docs https://docs.microsoft.com/de-de/xamarin/xamarin-forms/platform/other/...

9. In the GTK project, rename **Class1.cs** to **Program.cs**.
10. In the GTK project, edit the **Program.cs** file so that it resembles the following code:

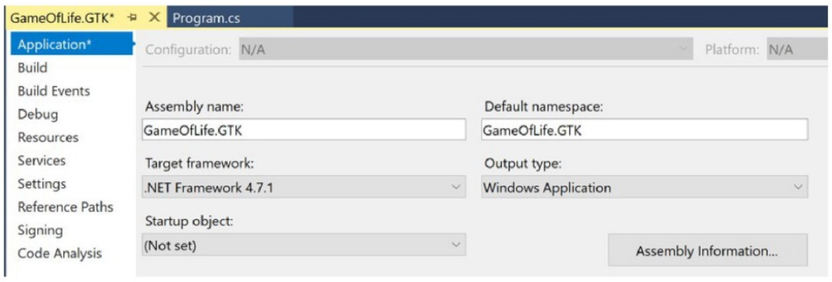
C# Kopieren

```
using System;
using Xamarin.Forms;
using Xamarin.Forms.Platform.GTK;

namespace GameOfLife.GTK
{
    class MainClass
    {
        [STAThread]
        public static void Main(string[] args)
        {
            Gtk.Application.Init();
            Forms.Init();

            var app = new App();
            var window = new FormsWindow();
            window.LoadApplication(app);
            window.SetApplicationTitle("Game of Life");
            window.Show();

            Gtk.Application.Run();
        }
    }
}
```

This code initializes GTK# and Xamarin.Forms, creates an application window, and runs the app.

11. In the **Solution Explorer**, right click the GTK project and select **Properties**.
12. In the **Properties** window, select the **Application** tab and change the **Output type** drop-down to **Windows Application**.

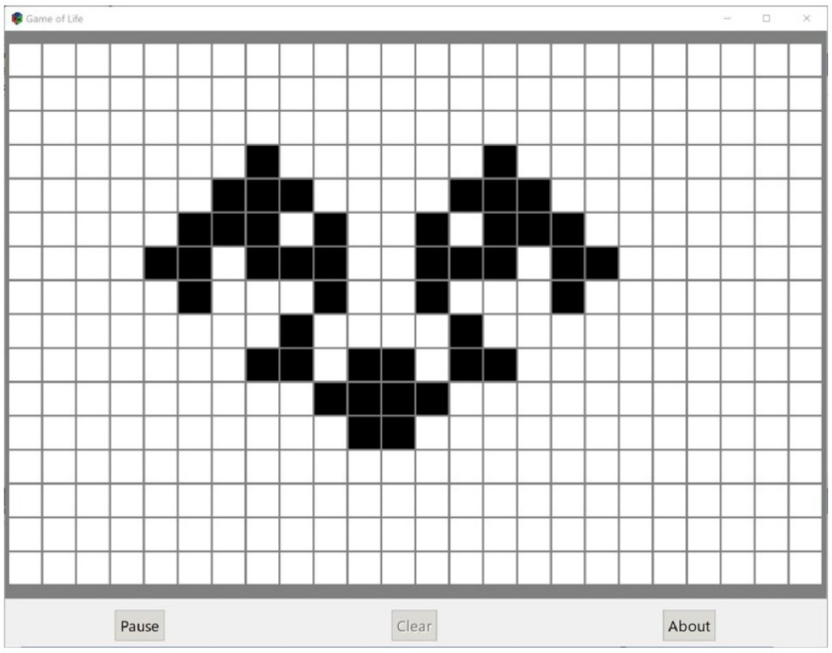
5 von 8 19.05.2020, 15:39



Configuration: N/A    Platform: N/A

Assembly name: GameOfLife.GTK
Default namespace: GameOfLife.GTK
Target framework: .NET Framework 4.7.1
Output type: Windows Application
Startup object: (Not set)

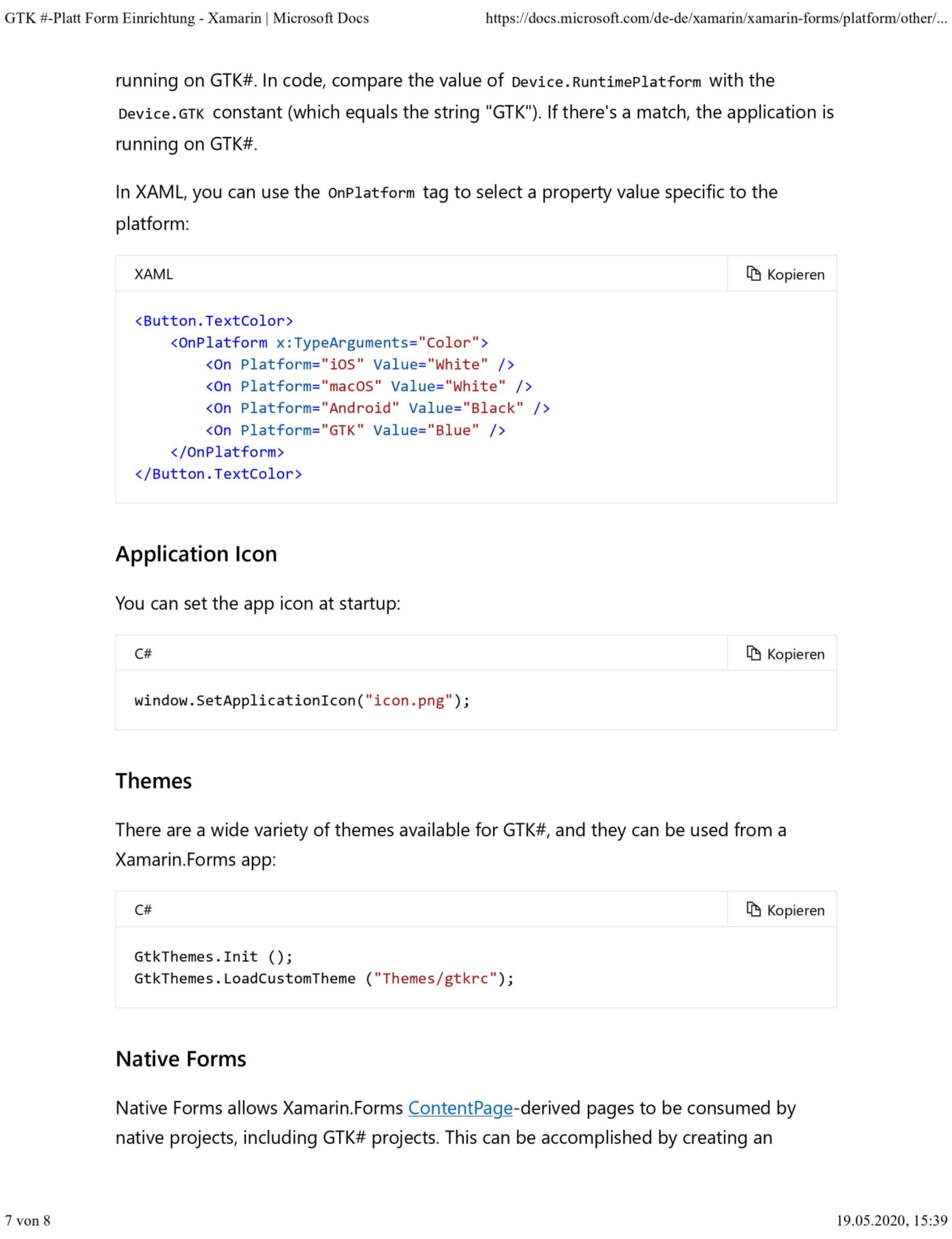
13. In the **Solution Explorer**, right-click the GTK project and select **Set as Startup Project**. Press F5 to run the program with the Visual Studio debugger on the Windows desktop:



## Next Steps

### Platform Specifics

You can determine what platform your Xamarin.Forms application is running on from either XAML or code. This allows you to change program characteristics when it's



GTK #-Platt Form Einrichtung - Xamarin | Microsoft Docs https://docs.microsoft.com/de-de/xamarin/xamarin-forms/platform/other/...

running on GTK#. In code, compare the value of `Device.RuntimePlatform` with the `Device.GTK` constant (which equals the string "GTK"). If there's a match, the application is running on GTK#.

In XAML, you can use the `OnPlatform` tag to select a property value specific to the platform:

XAML Kopieren

```xml
<Button.TextColor>
    <OnPlatform x:TypeArguments="Color">
        <On Platform="iOS" Value="White" />
        <On Platform="macOS" Value="White" />
        <On Platform="Android" Value="Black" />
        <On Platform="GTK" Value="Blue" />
    </OnPlatform>
</Button.TextColor>
```

## Application Icon

You can set the app icon at startup:

C# Kopieren

```csharp
window.SetApplicationIcon("icon.png");
```

## Themes

There are a wide variety of themes available for GTK#, and they can be used from a Xamarin.Forms app:

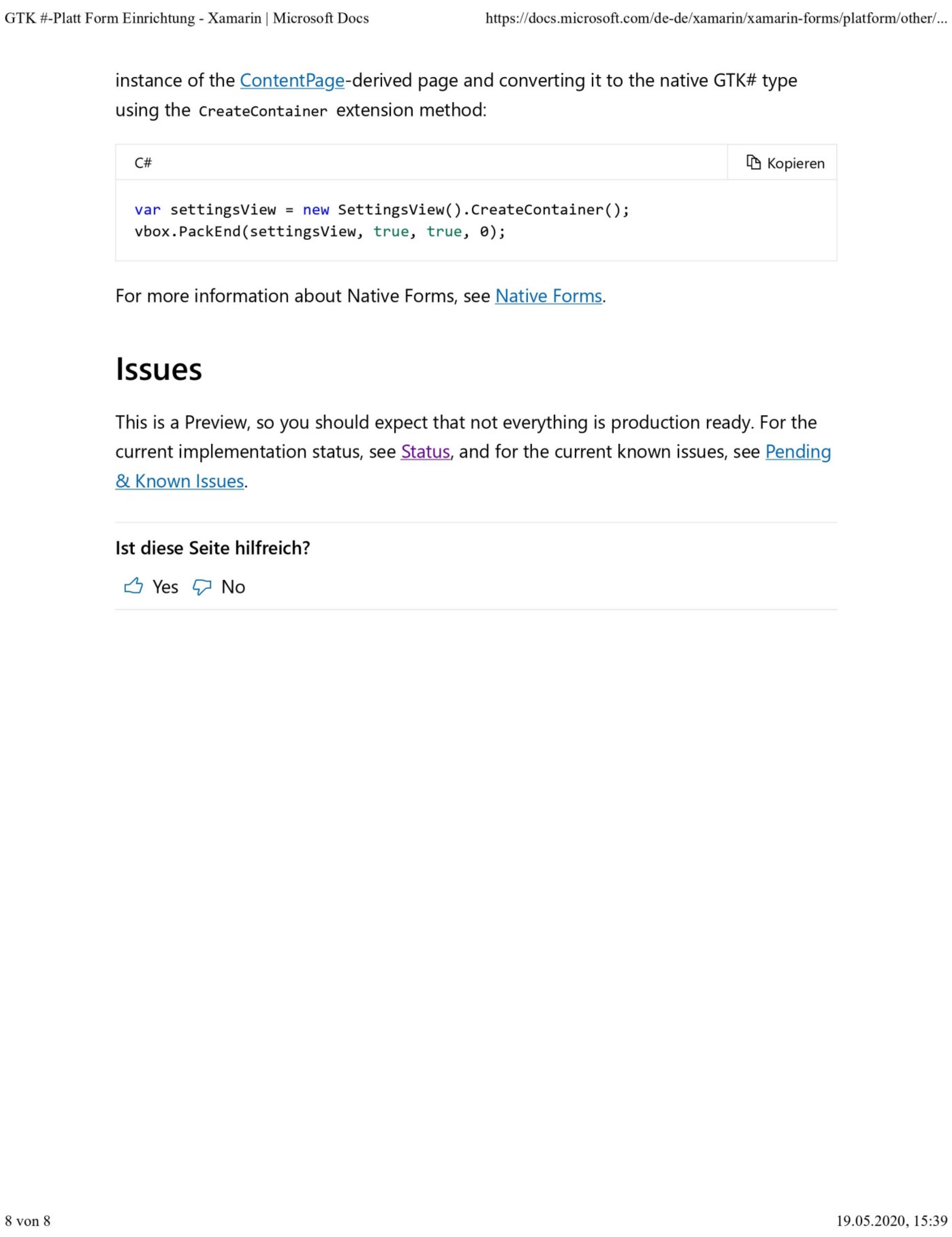
C# Kopieren

```csharp
GtkThemes.Init ();
GtkThemes.LoadCustomTheme ("Themes/gtkrc");
```

## Native Forms

Native Forms allows Xamarin.Forms ContentPage-derived pages to be consumed by native projects, including GTK# projects. This can be accomplished by creating an

7 von 8 19.05.2020, 15:39



GTK #-Platt Form Einrichtung - Xamarin | Microsoft Docs https://docs.microsoft.com/de-de/xamarin/xamarin-forms/platform/other/...

instance of the ContentPage-derived page and converting it to the native GTK# type using the `CreateContainer` extension method:

C# Kopieren

```csharp
var settingsView = new SettingsView().CreateContainer();
vbox.PackEnd(settingsView, true, true, 0);
```

For more information about Native Forms, see Native Forms.

## Issues

This is a Preview, so you should expect that not everything is production ready. For the current implementation status, see Status, and for the current known issues, see Pending & Known Issues.

**Ist diese Seite hilfreich?**

Yes No

8 von 8 19.05.2020, 15:39

# 9 Important Notice

The information in this publication has been carefully checked and is believed to be entirely accurate at the time of publication. F&S Elektronik Systeme assumes no responsibility, however, for possible errors or omissions, or for any consequences resulting from the use of the information contained in this documentation.

F&S Elektronik Systeme reserves the right to make changes in its products or product specifications or product documentation with the intent to improve function or design at any time and without notice and is not required to update this documentation to reflect such changes.

F&S Elektronik Systeme makes no warranty or guarantee regarding the suitability of its products for any particular purpose, nor does F&S Elektronik Systeme assume any liability arising out of the documentation or use of any product and specifically disclaims any and all liability, including without limitation any consequential or incidental damages.

Products are not designed, intended, or authorized for use as components in systems intended for applications intended to support or sustain life, or for any other application in which the failure of the product from F&S Elektronik Systeme could create a situation where personal injury or death may occur. Should the Buyer purchase or use a F&S Elektronik Systeme product for any such unintended or unauthorised application, the Buyer shall indemnify and hold F&S Elektronik Systeme and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, expenses, and reasonable attorney fees arising out of, either directly or indirectly, any claim of personal injury or death that may be associated with such unintended or unauthorized use, even if such claim alleges that F&S Elektronik Systeme was negligent regarding the design or manufacture of said product.