

# **MuntsOS Embedded Linux**

## ***Application Note #18: .Net on MuntsOS***

**Revision 1  
18 January 2024**

**by Philip Munts  
dba Munts Technologies  
<http://tech.munts.com>**

## Introduction

By installing the `dotnet` extension package, you can enable **MuntsOS Embedded Linux** (hereafter just **MuntsOS**) to run .Net programs.

[Application Note #8](#) presents a worked example using `dotnet` and other command line tools to create a .Net Core C# console application (hereafter just .Net application) project, build the executables, transfer the executables to a **MuntsOS** target computer, and run the main program assembly on the target computer.

This application note provides a more detailed explanation of how **MuntsOS** supports .Net.

## .Net Project Templates

## .Net Application Deliverables

There are several ways to package the deliverables for a .Net application. The following are applicable to **MuntsOS**.

### **dotnet publish**

This is the canonical command to build a .Net application from the command line.

Running `dotnet publish` in the project directory creates a release subdirectory named `bin/Release/net9.0/publish` which contains the application executables.

Copy the contents of `bin/Release/net9.0/publish` to the target computer and run the application with `dotnet`, as the following commands excerpted from **Application Note #8** illustrates:

```
scp bin/Release/net9.0/publish/* root@snoopy:.  
ssh root@snoopy  
dotnet blinky.dll
```

The minimal set of executables in `bin/Release/net9.0/publish` contains all of the `.dll` files (one of which is the main program assembly) plus the one `.runtimeconfig.json` file. You do not need to transfer either of the `.pdb` and `.deps.json` files to the target computer, but it does no harm to include them either, and usually simplifies the copy operation.

The executables written to `bin/Release/net9.0/publish` are **architecture independent**, meaning they can (in principle) be run *as-is* on Windows, Linux or macOS computers of various and sundry instruct set architectures.

In reality, few if any applications intended for **MuntsOS** will be able to run on Windows or macOS computers, as they will necessarily lack the Linux shared library `libsimpleio.so`, but they often *will* be able to run on Linux computers with different instruction sets, such as 64-bit Intel x86-64 and 64-bit ARMv8 or even 32-bit ARMv7.

If you open a .Net program project with **Microsoft Visual Studio** on Windows, you can do the equivalent of `dotnet publish` by first setting the project configuration to **Release** and then doing **Build** → **Publish Selection** from the menu bar and working through the prompts that follow.

*Tip: You can also just build the solution with `dotnet build` at the command line or F6 aka **Build** → **Build Solution** in Visual Studio. Both of these operations place the executables in `bin/Release/net9.0` instead of `bin/Release/net9.0/publish`. F6 is much easier than wading through the dialogs of **Build** → **Publish Selection**.*

## dotnet pack

Running `dotnet pack` in the project directory results in writing a `.nupkg` file to the subdirectory `bin/Release`. The `.nupkg` file is just a renamed `.zip` file containing the minimal set (*i.e.* excluding the `.pdb` and `.deps.json` files) of architecture independent executables plus some rather opaque metadata files. Packing an application into a `.nupkg` file can reduce its total amount of storage space and make it more convenient to move around than a group of files, especially if you are using a lot of library assemblies.

The equivalent of `dotnet pack` in **Visual Studio** is just **Build** → **Pack <appname>**.

The **MuntsOS** root file system contains a program named `nupkg` that will unpack and install the .Net application executables contained inside a `.nupkg` file, using a command similar to the following:

```
nupkg blinky.1.0.0.nupkg
```

This command creates the directory `/usr/local/lib/blinky` and installs the architecture independent application executable files there. It also creates a one line shell script for running the application `/usr/local/bin/blinky` with contents similar to the following:

```
exec dotnet /usr/local/lib/blinky/blinky.dll "$@"
```

If you move the `.nupkg` file to `/boot/packages` on the target computer, it will be saved in permanent storage and installed automatically at boot time:

```
dotnet pack
scp bin/Release/blinky.1.0.0.nupkg root@snoopy:.
ssh root@snoopy
mount -orw /boot
mv blinky.1.0.0.nupkg /boot/packages
umount /boot
```

If you create your .Net application project using the template from **libsimpleio-templates**, the project file will contain some logic to automatically pick up a start script to be installed into `/etc/rc.d` on the target computer at boot time and executed automatically whenever the target computer reboots. The start script must be placed in the project directory, and named `S00<appname>`. Continuing with the `blinky` example, run the following command in the project directory to create a start script: `echo "/usr/local/bin/blinky" >S00blinky`

If you need to execute multiple programs at boot time, you can edit the project configuration file (e.g. `blinky.csproj`) for each program and change `S00` to `S01`, `S02`, `S03`, etc. to control the execution order, since start scripts in `/etc/rc.d` are executed in alphabetical order. Each program started by a script in `/etc/rc.d` must run to completion and exit or detach itself from foreground execution using `LINUX_detach()` or its equivalent to avoid blocking its successor(s).

**dotnet publish -r linux-arm64 -p:PublishSingleFile=true --self-contained true**

A single file application *must* be built for a particular architecture, `linux-arm64` (current 64-bit targets) or `linux-arm` (obsolete 32-bit targets) for **MuntsOS**, because the deliverable is just a binary program file for the target computer.

A **self-contained** .Net single file application contains the main program assembly, library assemblies, *and* the entire .Net runtime within the program file.

**Pros:**

- You never have to worry whether the .Net runtime extension installed on the **MuntsOS** target computer is too old or installed at all.

**Cons:**

- The application program file will be very large--over 78 MB at time of writing.
- You must install the `libcicu` extension (another 36 MB) to the target computer *or* define the following environment variable in `/etc/environment`:

```
DOTNET_SYSTEM_GLOBALIZATION_INVARIANT=true
```

**dotnet publish -r linux-arm64 -p:PublishSingleFile=true --self-contained false**

A **framework-dependent** .Net single file application contains only the main program assembly and library assemblies.

**Pros:**

- The deliverable program file will be much smaller.
- `DOTNET_SYSTEM_GLOBALIZATION_INVARIANT` is set by the .Net runtime extension package.

**Cons:**

- The target computer must have a recent enough .Net runtime extension package installed before you can run your program.

The recommended types of deliverables for running a .Net program on **MuntsOS** are a `.nupkg` file or a framework dependent single file.

The `Makefile` included in the project template from `libsimpleio-templates` produces a framework dependent single file application by default. You can edit `Makefile` to change `default: coreapp_mk_single` to `default: coreapp_mk_nupkg` to produce a `.nupkg` file instead.