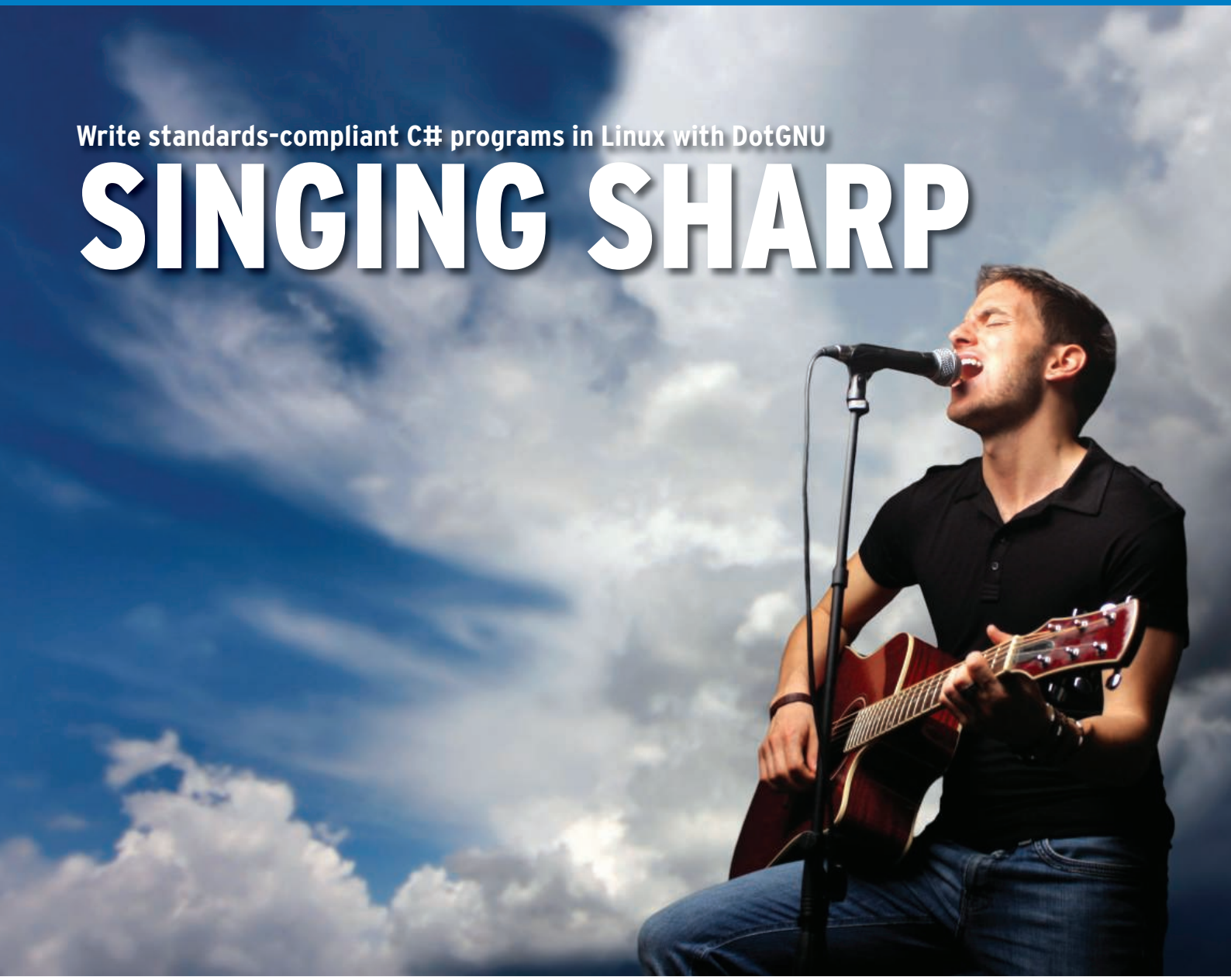

Write standards-compliant C# programs in Linux with DotGNU

SINGING SHARP

Write C# programs in Linux with the free and vendor-neutral DotGNU.

BY MAYANK SHARMA

Like many, I grew up programming in C and C++ before moving into LAMP-land and Python, Perl, and PHP. But like that first car, first date, and first paycheck, everyone has a special memory of the first time they compiled gibberish into executable machine code.

Thanks to GCC, open source developers have never had any issues writing C or C++ code on a Linux platform. However, when Microsoft announced their .NET initiative and the intention to focus their efforts around C#, few people expected Microsoft to release a Linux client for their "platform-independent" development tool.

Fortunately, .NET's Common Language Infrastructure (CLI) and the C# programming language are codified as ECMA (a non-profit standards body) and ISO international standards, which creates the possibility for independent implementations. The DotGNU project was started with huge fanfare to bring standards-compliant C# to Linux. Over the years, DotGNU has received less attention than the other .NET for Linux: the Novell-sponsored Mono project. Nevertheless, the project is still chugging along, and it is even finding some use in commercial applications.

Although DotGNU has lost some of its steam in recent years, it is still in active development. Certainly one cannot compare the pace of development with that of Mono – or even with its own initial activity – but that's mainly because, at DotGNU, the emphasis is on standards compliance and staying free of license restrictions, rather than on being compatible with the latest from the Microsoft camp. As DotGNU developer Klaus Treichel points out, it is unfair to compare DotGNU with Mono because Mono has full-time developers paid by a large company. DotGNU has always been a voluntary project supported with donations. One such donation was from Trumpf Group, which has a pulsed laser cutting tool that uses DotGNU. The touch screen user interface of the laser tool was built through the use of DotGNU.

Although some parts of the Mono environment (including the C# compiler) are released under the GPL, other parts are subject to license and patent concerns that reflect the complex business relationships between Novell and Microsoft. Fortunately for us, then, the DotGNU project offers a vendor-indepen-

dent alternative for open source programmers who want to try their luck with C# and .NET.

Breaking Down DotGNU

DotGNU is more than just a C# compiler, but I'll start off with that. Portable.Net is the free implementation of .NET, and it contains a run-time engine, a C# compiler, and a host of other tools that make Portable.Net easy to port to other platforms. All these components are written in C. The aim of the project is to make the development of .NET apps easy on non-Microsoft platforms.

In the early days of development, the C# system library was split from the main Portable.Net distribution. The libraries are now available as part of the *pnetlib* package. Another important component is *treecc*, an aspect-oriented programming tool that assists in development with the DotGNU C# compiler.

For writing web services, DotGNU relies on the DotGNU Execution Environment, or DGEE, and phpGroupWare. DGEE is a web-service server that can accept and process XML-RPC requests from web services, and it can generate browsable documentation for these services in HTML or XML. phpGroupWare is a groupware suite that also provides a host of web-service components.

The DotGNU C# compiler also compiles programs written in C, thanks to the *libc* implementation of the C compiler, *pnetC*. Because DotGNU's objective is to follow the standards, rather than follow Microsoft, the implementation lacks a few assemblies. To help developers use the missing bits, the DotGNU folks distribute some of the libraries from the Mono project. This distribution of the Mono libraries is achieved via the build scripts in the *ml-pnet* package.

Why Program with DotGNU?

One of the main reasons for writing C# code in DotGNU is that it is compatible with the EMCA standards for C# and the CLI. Furthermore, DotGNU also is compatible with Microsoft's own CLI implementation of the .NET framework.

Thanks to the modular design of Portable.Net, the DotGNU C# compiler can run on multiple platforms. Portable.Net's run-time engine and the C# class library have extensive support for embedded system profiles and can be built with different ECMA profiles. Each profile enables or disables features in the system. As a testament to its portability, one of the founding developers, Gopal Vijayaraghavan, was able to get Portable.Net running on the Indian hand-held computer, the Encore Simputer, during the three days of FOSS.IN in Bangalore.

Programmers also will appreciate the self-contained nature of DotGNU and that it doesn't depend on external libraries. A much-discussed feature of DotGNU is its implementation of the *System.Windows.Forms* library, which is used to build GUIs. *System.Windows.Forms* simplifies development by reducing dependence on other toolkits.

Getting the Tools

Some distro repositories carry DotGNU binaries, but it's a good idea to compile them from source. The tarballs are available on the DotGNU website [1], and you can also fetch them via CVS.

To get the compiler and libraries, install *treecc*, *pnet*, and the *pnetlib* packages. Optionally, the *ml-pnet* package lets you work with Mono's libraries, and the *pnetC* package enables the DotGNU compiler to compile C programs.

The command:

```
cvs -z3 -d:pserver:anonymous@cvs.sv.gnu.org:/sources/dotgnu-pnet co .
```

downloads the latest source of all DotGNU packages inside the directory it is issued from, so make sure it is under something like */opt/dotgnu*. Now change to each directory and run *./auto_gen.sh* for all the packages to generate the configuration and make files. Once that's completed, or if you just grabbed the tarball instead of checking out via CVS, the usual *./configure*, *make*, *make install* (the last one, as usual, as root) will install the DotGNU compiler and libraries.

Working with the Compiler

For those who speak binary, DotGNU works by transforming bytecode into a simple instruction set that is passed on

Listing 1: Hello.cs

```
01 using System;
02 public class HelloWorld
03 {
04         public static void Main (string [  ] args)
05         {
06                 if (args.Length != 1)
07                 {
08                         Console.Error.WriteLine("You must tell me your name.");
09                         Environment.Exit(-1);
10                 }
11                 string name = args[0];
12                 Console.WriteLine ("Hello, {0}!", name);
13         }
14 }
```

Anzeige
wird
separat
angeliefert

Anzeige
wird
separat
angeliefert

Listing 2: Hello-Advanced.cs

```
01 using System;
02 using System.Collections;
03
04 public class HelloWorld
05 {
06        public static void Main()
07               {
08               String value;
09                      value = Environment.
   GetEnvironmentVariable("USER");
10                      if(value != null)
11                      {
12                              Console.WriteLine("Hello,
   {0}!", value);
13                      }
14                      else
15                      {
16                           Console.WriteLine("Sorry, you
   apparently don't have a name!");
17                      }
18               }
19 }
```

to a virtual machine to be executed via an interpreter. This design makes DotGNU easily portable and explains the number of supported platforms.

The components you'll use are the *ilrun* run-time engine, which executes the binaries doled out via the *cscc* compiler, and *cscc some-program.cs*, which produces a file called *a.out*. This file can then be executed with:

```
ilrun a.out
```

With the *-o* switch, you can specify a file name when compiling your program with *cscc* – for instance:

```
cscc -o some-program.exe ➥
some-program.cs
```

To print debugging information, use the *-v* switch while compiling.

If you have to link against a particular library, say *System.Drawing*, you'll need to point this out to the compiler with the *-lLIBRARY* switch, which will search for the libraries DLL along the library search path. If you are compiling a GUI program that uses the *System.Windows.Forms* library, you can also use the *-winforms* switch, which automatically links all the libraries required to process a WinForms-dependent program. Sometimes you'll have to create your own DLL libraries. The *-shared* switch will produce these DLLs instead of a *.exe*.

In addition to C#, the DotGNU compiler can also compile to Java Virtual Machine bytecode with the *-mjvm* switch. Remember to use the *.jar* file extension instead of *.exe* or *.dll*.

Dive In

After that brief introduction to the compiler and common options, I'll honor the long tradition of coding tutorials by writing a "Hello, World" program in C# (Listing 1) and compiling it with *cscc -o hello.exe hello.cs*.

The simple program contains only one method: *Main()*. Command-line arguments are passed to this method as an array of string objects by way of the System library and its various classes and methods. *System.Environment.Exit()* exits the program and sends a return code to the shell. The *System.Console* class interfaces the command line to the program. The *Console.Writeline()* method writes the greeting to standard output, and *Console.Error.Writeline()* writes to standard error.

Listing 1 greets the user with a name slipped in as input. But why enter a name for the user when a name is probably already defined on the system? Using the *System.Collections* library, you can, among other things, display the contents of all the environment variables. If you modify the *hello.cs* program to use the *System.Collections library* (Listing 2), you can read the username of the user executing the program with *value = Environment.GetEnvironmentVariable("USER")*, wherein *value* is a String type variable.

Get Set GUI

One of the best bits about DotGNU is an implementation of the *System.Windows.Forms* library that doesn't require translation via other popular toolkits such as Gtk. Much like the Java Swing library, DotGNU's *System.Windows.Forms* draws its own controls.

To compile the code in Listing 3, use:

```
cscc -o form.exe form.cs ➥
-winforms.
```

The code displays a simple re-sizable window with the usual minimize, maximize, and close controls.

Listing 3: Using System.Windows.Forms

```
01 using System;
02 using System.Windows.Forms;
03
04 public class MyForm : Form
05 {
06        public MyForm ()
07        {
08               this.Text = "The beginnings of a multi-tab
   text editor";
09               this.Height = 600;
10               this.Width = 800;
11        }
12 }
13
14
15 public class MyApp
16 {
17        public static void Main(string[] args)
18        {
19               App.Run(new MyForm());
20        }
21 }
```

Listing 4: Advanced Form Controls

```
01 public class MyMenu : System.Windows.Forms.MainMenu
02 {
03         public System.Windows.Forms.MenuItem mFile;
04         public System.Windows.Forms.MenuItem mFileNew;
05         public System.Windows.Forms.MenuItem mFileSave;
06         public System.Windows.Forms.MenuItem mFileExit;
07
08         public MyMenu()
09         {
10                 mFile = new MenuItem("&File");
11                 this.MenuItems.Add(mFile);
12
13                 mFileNew = new MenuItem("&New");
14                 mFileSave = new MenuItem("&Save");
15                 mFileExit = new MenuItem("E&xit");
16
17                 mFile.MenuItems.Add(mFileNew);
18                 mFile.MenuItems.Add(mFileSave);
19                 mFile.MenuItems.Add(mFileExit);
20         }
21 }
```

The next step is to add a menu to this window. The complete listing is available on the magazine website [2], but the most important bits are in Listing 4. A custom class called *MyMenu* inherits from the *System.Windows.Forms.MainMenu* class and is used to create the menu items. The menu item variables are of the *MenuItem* type, which is specifically used to create items within a menu or a context menu.

The *MyMenu()* constructor creates the *File* menu item with the new keyword. Similarly, Listing 4 creates an instance of the three menu items and specifies how they'll appear in the menu. Because I already have the File menu ready, I use the *Add()* method to add the three menu items to the main menu.

For the first-time GUI programmer, this procedure might be a little overwhelming, but really it couldn't be simpler. The various *System.Windows.Forms* methods take care of adding GUI functionality to the menu and the items, so that when you compile and run the code, the File menu will function as it does in any GUI application: First you click to display the items, and then you click to fold them back in.

Of course, you still have to associate your menu with your original form code and program event handlers for the various items to get them to work. Also, you'll have to tie these event handlers to the particular menu item's click event. Further on, you have to create dialog boxes and add functionality to them with the use of buttons to save the file and do other tasks.

Conclusion

Several books and tutorials on C# can be found on the Internet. The advantage of DotGNU is that it is completely standards compliant; therefore, you can use documentation from Microsoft's C# library. To get things done faster, you can use front ends, such as Microsoft's Visual Studio and Mono's MonoDevelop IDE [3]. But if you want to write truly portable code with DotGNU, you'll have to be extra careful with these tools and make sure you stay away from features unique to a particular platform. ■

Languages and Libraries

C# borrows a bit from several popular languages, including C, C++, and Java. If you've programmed in Java, you'll find that, in C#, you can have a class name with a name that's not the same as the source file. Similar to Java, and unlike the C languages, C# doesn't have header files. Plus, you have the flexibility of either importing a namespace or specifying it fully when using a function like *System.Console.Writeline*. Again, this has the Java footprint all over it.

Also note that the C++ *::* operator isn't required in C#. The task of referencing class members is accomplished with the dot (.) operator.

All the most important C# libraries are implemented in DotGNU. The library lineup includes the core *System* library that (among other things) contains classes for handling data types, the *System.Collections* library that helps define and traverse objects such as lists and arrays, *System.IO* for reading and writing streams and files, *System.Net* for network programming, *System.Security* for making your programs respect permissions, and, of course, *System.Windows.Forms* for creating GUIs.

However, some libraries are not directly available, such as *System.Data*, which provides classes to implement the ActiveX Data Object, or ADO for reading and writing data from various data stores such as databases. But you can use the ADO library via the *ml-pnet* package.

DotGNU-specific definitions for the packaged libraries are available online [4], although they are not complete, and it is possible that a function might not be defined there but available nonetheless. Some of the libraries do have missing constructs and methods, and you can also learn more about these elements online [5]. However, this information might not be up to date.

INFO

[1] DotGNU website: *http://www.gnu.org/software/dotgnu/*

[2] Listings for this article: *http://www.linux-magazine.com/resources/article_code*

[3] MonoDevelop: *http://monodevelop.com/*

[4] DotGNU library definitions: *http://www.gnu.org/software/dotgnu/pnetlib-doc/*

[5] Library status: *http://www.gnu.org/software/dotgnu/pnetlib-status/*

THE AUTHOR

Mayank Sharma has written for various Linux publications, including Linux.com, IBMdeveloperWorks, and Linux Format, and he has published two books through Packt on administering Elgg and Openfire. Occasionally he teaches FLOSS technologies. You can reach him at: http://*www.geekybodhi.net*.