Command Line to Windows Application Conversion Using C# and cmd.exe

The desire to adapt the bcrypt module "http:/bcrypt.sourceforge.net/" to a Windows Application format motivated this work, which is explained here. bcrypt encrypts files using the Blowfish scheme; details about the encryption and decryption may be

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;
using System.IO;
using System.Collections;
using System.Threading;
using System.Text.RegularExpressions;
using PasswordDialog;/* see earlier manuscript (password dialog and namespace discussion:[1]


namespace bcrypt {
 public partial class Form1 : Form {
 public Form1() {
//this form was created in the standard manner with one button only ("DOIT").
InitializeComponent();
 }
```

(The creation of this form by drop-and-drag in Visual Studio is not discussed here.)

I simplified the user interface by having one button only, which when pressed, caused the following process to be carried out. The color coding of instances is intended to make the dispersed information slightly more understandable (I hope it works).

The first task is to realize that I am going to need a cmd.exe shell, which in C# is called a Process. We call my version of this process myBcrypt, and intend to use it further on down in the code.

But first, I need to find out what file we want encrypted or decrypted. For this I employ a FileDialog. The next few lines of code are settings required for this FileDialog. The only one that should be changed when in production, is the InitialDirectory value, which is set to the place where the bcrypt.exe files and the appropriate dll files are stored:

1"C# Namespace Usage for Including Foreign Projects" and other related papers can be found inside this site:
http://web.uconn.edu/~cdavid/Csharp/*.pdf

In the example being discussed, the directory structure is based on where Visual Studio's projects (including this one) were stored. I stored test case files there (in our project's directory) for en- and de-cryption during debugging. This saves some repetitive clicking about, and should/can be changed by the reader!

```
private void button1_Click(object sender, EventArgs e) {
//this is the DOIT buttons action
 Process myBcrypt= new Process();// see the code below marked ($%$).
 OpenFileDialog fdlg= new OpenFileDialog();
   fdlg.Title = "BCRYPTOpen File Dialog";
   fdlg.InitialDirectory = @"C:/Documents and Settings/david/My Documents/Visual Studio
2005/Projects/bcrypt";
fdlg.Filter = "All files (*.*)*.*All files (*.*)*.*";
   fdlg.FilterIndex = 2;
fdlg.RestoreDirectory = true;
```

Ah, not so fast! Before I do the file directory thing, let's do the password thing, so that we can apply the password to the file when the time comes. So here is the password thing. This has been discussed (password dialog and namespace discussion) already in these pages. The password program was downloaded into a separate project, compiled and tested there, and part of it is to be used here.

```
 passwordDlg passForm= new passwordDlg();
 if (passForm.ShowDialog(this) == DialogResult.OK) {
/* OK, do something with the password. This is just a demo.. so lets just splash it up for the world to
see.
MessageBox.Show(this,"The password entered was: " + passForm.getPass(), "Password Result",
MessageBoxButtons.OK, MessageBoxIcon.Information);
used earlier to confirm that entered password was the one desired
*/
 }
passForm.Dispose();
```

If we get this far, the password has passed muster, i.e., is 14 characters long (at least) has lower case, uppercase, numbers, strange symbols, etc., and we're ready to do. The following text checks the password, and executes the including code if all is well.

```
 if (Regex.IsMatch(passForm.getPass(), regex_psswd)) {
 if (fdlg.ShowDialog() == DialogResult.OK) {

    myFile = fdlg.FileName;
    FileInfo fleMembers = new FileInfo(myFile);
    myFile = fleMembers.Name;
```

First, we need to obtain the file's name from the dialog. Then, it turns out that the dialog returns the entire one, i.e., "C:\Doc..." and when that was passed to brcypt, it failed! Therefore, we need to remove the directory pre-pended information to obtain just the filename itself. fleMembers does this for us as shown. "myFile" contains the name actually chosen by the FileDialog.

```
// use command prompt
 myBcrypt.StartInfo.FileName = "cmd.exe";//($%$).
myBcrypt.StartInfo.WorkingDirectory = @"C:/Documents and Settings/david/My Documents/Visual
Studio 2005/Projects/bcrypt";
myBcrypt.StartInfo.ErrorDialog = true;

// /c switch sends a command

myBcrypt.StartInfo.Arguments = "/c bcrypt.exe " + myFile;//.bfe";
```

myBcrypt.StartInfo.WorkingDirectory has to be set with care. bcrypt.exe requires that "zlib.dll" exist in the same directory as the executable, and I happened to have placed them both in the directory you see above. You, of course, can place them anywhere you like so long as you change this entry!

We've got to stop, for a moment, and smell the roses. The "cmd.exe" executable usually opens a window which is the old Command Prompt (or DOS Window). One can type in it, and programs executing usually write to it. We need to

1. not open the window if possible,
2. not type in it, but have the windows application act as if its typing into it, and
3. not have any output written to it, i.e., if necessary, have the output redirected in some way to the windows application.

I started, as an ignorant beginner, with the assumption that I could "stack" the things to be typed into the cmd.exe window, two exactly equal versions of the password, onto the command stack passed to bcrypt.exe. I was wrong and the following failed:

```
/* myBcrypt.StartInfo.Arguments = myBcrypt.StartInfo.Arguments + "\r" + "somepassword" +
"\r" + "somepassword";
\r\n attempt at loading the command line stack
\r failed. window flashed by died, no encryption, it failed.*/
```

It turns out to be more complicated than I thought. Instead, one has to redirect I/O intelligently, and you will see in the next few lines how that was done.

```
// don't exec with shellexecute api
myBcrypt.StartInfo.UseShellExecute = false;

// redirect stdout to this program
myBcrypt.StartInfo.RedirectStandardOutput = false;//failed true &amp; false
```

```
// don't open a cmd prompt window
myBcrypt.StartInfo.CreateNoWindow = false;
myBcrypt.StartInfo.RedirectStandardInput = true;
myBcrypt.StartInfo.RedirectStandardOutput = true;
```

So, what we've done is to redirect input and output (ignoring errors now).

```
myBcrypt.Start();
```

Having started the cms.exe file, we create a StreamWriter from the redirected input and write the result of the PasswordDialog (above) twice. This is the password. We did it.

```
    StreamWriter sw = myBcrypt.StandardInput;
    StreamReader sr = myBcrypt.StandardOutput;
    sw.WriteLine(passForm .getPass());
    sw.WriteLine(passForm.getPass());
myBcrypt.CloseMainWindow();//this may not be necessary
myBcrypt.Close();
    }
 }
 else {
    MessageBox.Show("Your password does not pass the sniff test!");
    }
 }
string myFile;
String regex_psswd = @"(?x)^(?=.* ( \d \p{P} \p{S} )).{8,}";
 }
}
```

The regex_passwd above was stolen from a good web page: Regular Expressions which should be referenced for further information.

What have we done? We've 1) used the password dialog from another program, 2) caused a command line program to function under Windows as a Windows application, and 3) made it possible to input to this (in essence DOS program) data obtained from the Windows application!

One note in ending. The bcrypt module actually uses passphrases, not passwords. The idea is to use a phrase, presumably one which is simple, instead of a single expression. Therefore one should play with the regex_psswd string to check that there is at least one blank, and to lower the requirements about special characters, upper case characters, etc..

Carl David
Department of Chemistry
University of Connecticut
Storrs, Connecticut 06269-3060
Carl.David@uconn.edu