A Testing Mail Server For Unit Testing Email Functionality

 So you are coding along riding that **TDD** high when you reach the point at which your code needs to send an email. What do you do now?

You might consider writing something that looks like:

```
EmailMessage email = new EmailMessage();
email.FromAddress = new EmailAddress(from);
email.AddToAddress(new EmailAddress(to));
email.Subject = subject;
email.BodyText = message;

SmtpServer smtpServer = new SmtpServer(SmtpServer, Port);
email.Send(smtpServer);
```

But you, being a TDD god(des) know better and quickly refactor that into some sleek code that uses an EmailProvider. This ensures that your code is not tied to any specific email implementation and will make unit testing your code easier. Just swap out your concrete email provider for a unit test specific email provider. Now your code looks like:

```
EmailProvider.Instance().Send(to, from, subject, message);
```

But a nagging thought still pulls at the edge of your consciousness. â€œShouldn’t I unit test my concrete email provider and actually make sure the email gets sent correctly?

I certainly think so. As for the semantic arguments around whether this really constitutes an *Integration Test* as opposed to a *Unit Test*, please don’t bore me with your hang-ups. Either way, it deserves a test and what better way to test it than using something like MbUnit or NUnit.

Wouldn’t it be nice to test your email code like so?

```
DotNetOpenMailProvider provider = new DotNetOpenMailProvider();
NameValueCollection configValue = new NameValueCollection();
configValue["smtpServer"] = "127.0.0.1";
configValue["port"] = "8081";
provider.Initialize("providerTest", configValue);

TestSmtpServer receivingServer = new TestSmtpServer();
try
{
receivingServer.Start("127.0.0.1", 8081);
provider.Send("phil@example.com",
"nobody@example.com",
"Subject to nothing",
"Mr. Watson. Come here. I need you.");
}
```

```
finally
{
receivingServer.Stop();
}

// So Did It Work?
Assert.AreEqual(1, receivingServer.Inbox.Count);
ReceivedEmailMessage received = receivingServer.Inbox[0];
Assert.AreEqual("phil@example.com", received.ToAddress.Email);
```

That there code starts up a mail server, sends an email to it, and then checks that the mail server received the email. It also quickly checks the to address.

This is a snippet of an actual unit test within the Subtext codebase.

A long while ago I discovered a wonderful .NET based freeware mail server written by Ivar Lumi. I decided to write a wrapper specifically for unit testing scenarios. I added the TestSmtpServer to a new project named Subtext.UnitTesting.Servers in the Subtext VS.NET solution.

The wrapper parses incoming **SMTP** messages and adds an ReceivedEmailMessage instance to the Inbox custom collection. This makes it easy to quickly examine the email messages sent via SMTP in your unit test.

As this is a very early draft, there are some key limitations. I have yet to implement multi-part messages and attachments in the object model. I also punted on dealing with multiple *to* addresses. However, the ReceivedEmailMessage class does have a RawSmtpMessage property you can examine. For now, it works very well for simple text based emails.

Over time, I hope to implement these more complicated testing features as the need arises. However, if you find this useful and would like to contribute, please do!

If you want to view the latest code, check out these instructions for downloading the latest Subtext code using Subversion.

Or you can simply download this one project here, though keep in mind that I will be updating this project, but not necessarily this link to the project.

Since the project is specifically for unit testing purpsose, I went ahead and embedded the unit tests for this server within the project itself using MbUnit references. However you can simply swap out the assemblies and references to use NUnit if that is your preference.
