C# and PostgreSQL installation cheat sheet v0.5

INTRODUCTION

This document contains brief instructions for installing a standard development environment for C# and using a PostgreSQL database backend.

If you are already comfortable working with another programming language and an associated backend RDMBS, we urge you to use that language. The purpose of this document is to furnish students who have no clear preferences with a straightforward solution that is known to work and is in line with the course material.

INSTALLATION OF VISUAL STUDIO/MONODEVELOP AND C#

C# currently requires the MS .NET environment, or some emulation of it, to function properly.

For computers running Windows, the absolutely easiest way to ensure that the correct environment is set up is to download and install **MS Visual Studio**. For MacOS and Linux, we recommend **MonoDevelop**; note that on all other operating systems than Windows-based ones, some fiddling with local settings and packages may be necessary.

- Windows: https://www.visualstudio.com/ (choose, e.g., "community edition")
- MacOS: http://www.monodevelop.com/download/
- Linux: http://www.monodevelop.com/download/

  All of the above have self-extracting or automatic installers; no adjustments to your local computer's environment (beyond the ones required in the installation and the required packages) should be necessary.

NOTES FOR INSTALLATION ON MS WINDOWS:

- During installation of Visual Studio, you will be prompted to select one or more "Workloads" – large bundles of development functionalities tied by a common theme. Choose *at least* **".NET Desktop development"**. Feel free to choose any other workloads or individual components you may desire.
- Cross-platform coding in recent distributions of Visual Studio is done via "UWP" (Universal Windows Platform). Several open-source libraries that allow for interfacing with database distributions not commercially endorsed by Microsoft are currently incompatible with UWP;

please be aware of this when settling on your high-level software design. Usually, non-UWP development (for e.g., traditional "Desktop" applications) is unaffected, and 3rd-party tools will run smoothly.

- After installation of Visual Studio, please use the following procedure:
    1. from the "**File**" menu, choose "**New**"->"**Project**". You will be prompted to select a *template* (and a name and storage location) for your project. A standard, no-frills choice of template is "**Windows classic desktop**"-> "**Empty project**" (or "console application").
    2. In the "**Tools**" menu, choose "**NuGet Package Manager**"->"**Package Manager Console**". A console will appear. In the console, at the "**PM>**" prompt, write "**Install-Package Npgsql".** The package will be installed automatically; the package can now be used in the scope of the solution it was imported in (include it by writing "**using npgsql"** as with other C# packages).
    3. Test your installation by replacing the code in the automatically created program "program.cs" file with the sample code at the end of this document. Run the code.
    4. Your installation may require references to be resolved when running the code; the most common problem is for references to "**system.data**" to be unresolved. Resolve any such problems by right-clicking the "**references"** item in the "**solution"** menu. Add a reference to "**system.data**" by searching for it in the pop-up window, putting a check mark next to it, and clicking "ok". Do the same, mutatis mutandis, for any other missing references (the error messages will tell you *exactly* which references need to be resolved).

NOTES FOR INSTALLATION ON MACOS:

Please follow the below procedure:

1) follow URL above (http://www.monodevelop.com/download/).
2) Install Mono + GTK.
3) Install Xamarin Studio Community edition.
4) Add support for .Net. Download .net core SDK (https://www.microsoft.com/net/core#macos)
5) Start Xamarin studio and create a solution: **Xamarin-> File -> New solution -> .NET -> Console Project (C#) -> Next-> Project name: <test_uis> -> Create**
6) In Xamarin studio, with the focus on the test_uis project, go to the Projects menu and add the npgsql library: **Xamarin -> Projects -> Add NuGet package -> search for npgsql. Check / add package.**
7) Your installation may require references to be resolved when running code; the most common problem is for references to "system.data" to be unresolved Resolve any such problems by right-clicking the "**references"** item in the "**solution"** menu. Add a reference to "**system.data**" by searching for it in the pop-up window, putting a check mark next to it, and clicking "ok". Do the same, mutatis mutandis, for any other missing references (the error messages will tell you *exactly* which references need to be resolved). In Xamarin studio, similarly to the above, add a

reference to System.Data: **Xamarin -> Projects -> Edit References -> search for system.data -> select System.Data version 4.0.0.0 -> OK.**

8) Test your installation by replacing the code in the automatically created program "Program.cs" file with the sample code at the end of this document. Run the code.

NOTES FOR INSTALLATION ON LINUX:

<Should be the same as for MacOS, mutatis mutandis. Must be tested>

INTERFACING WITH POSTGRESQL

SETTING UP A DATABASE CONNECTION

The following shows how to setup a Database connection to the DIKU-based server used elsewhere in the course. The procedure for setting up remote access to other servers is similar (and the code example we give later can be reused, with a few obvious changes). Setting up access to a local postgres server running on your own machine is even simpler.

For access to the DIKU-based server set up an ssh-tunnel to postgres:

```
ssh -L 5433:localhost:5431 your_kuid@ssh-diku-
uis.science.ku.dk ssh -L 5431:localhost:5432 -N a00648
```

The connect-string following C#-program must also be modified to access the DIKU-server and access to the DIKU server confirmed.

Example of using PostgreSQL in C#

The following is an example of how to use invocation of methods from the classes in Npgsql allows to query a PostgreSQL database. Please refer to the online documentation (or look directly in the Npgsql package files) for more examples of executing queries or stored procedures.

```
using System;
using Npgsql;

class Sample
{
   static void Main(string[] args)
```

```
    {
        // Connect to PostgreSQL  database
        NpgsqlConnection dbconn = new NpgsqlConnection("Server=127.0.0.1; User Id = postgres_user; "
+ " Password = your_postgres_password; Database = postgres; Port = 5432");
        dbconn.Open();

        // Define a query returning a single row result set
        NpgsqlCommand query = new NpgsqlCommand("SELECT * FROM information_schema.tables", d
bconn);

        // Execute the query and obtain a result set
        NpgsqlDataReader dr = query.ExecuteReader();


        while (dr.Read())
            Console.Write("{0}\t{1}\t{2} \n", dr[0], dr[1], dr[2]);

        Console.WriteLine("Press any key to exit.");
        Console.ReadKey(); //keep console window open in debug mode

        dbconn.Close();
    }
}
```

To change connection to a local instance of postgres, alter the connect string.

Database	Connect string
Diku	"Server=127.0.0.1; User Id = your_kuid; " + " Password = your_postgres_password; Database = uis; Port = 5433"
Local postgres	"Server=127.0.0.1; User Id = postgres; " + " Password = local_password; Database = postgres;"

Anders Lassen & Jakob Grue Simonsen, March 2017