Using .NET with MySQL

Why .NET?

.NET is a big, continuously evolving bundle of old and new Microsoft development technologies. It's had more than 1.8 billion installations. More then six million Windows developers use it. Version 4.5 requires at least a 1GHz processor; 0.5GB RAM; 1GB (32-bit) or 2GB (64-bit) disk space; if a client then Windows Vista SP2, 7 SP1, 8 or 8.1; if a server then Windows Server 2008 SP2/R2 SP1or 2012 R2/64-bit. It includes:

- a kernel, the *.NET Framework*, that implements HTTP (Hypertext Transfer Protocol), XML (Extensible Markup Language), SOAP (Simple Object Access Protocol), DDEX (Data Designer Extensibility) and UDDI (Universal Description Discovery and Integration). There are multiple versions that advance frequently;
- data handling classes bundled as *ADO.NET* (ActiveX Data Objects);
- Internet Information Services (IIS);
- a *Common Language Runtime* (CLR), a virtual machine for executing code compiled from various languages (analogous, and not accidentally, to the Java Virtual Machine);
- *ODBC.NET*, a wrapper for ODBC drivers;
- Microsoft *SQL Server*;
- *ASP.NET* (Active Server Pages .NET) for developing web applications using VBScript, JScript, Perlscript, Python, Visual Basic, C#, C, Cobol, and even Lisp or Smalltalk;
- Visual Basic .NET;
- *Visual Studio*: an elegant application development environment incorporating all the above, with a new free Community edition and commercial versions costing up to $12,000 US;
- *WebMatrix*, a descendant from the discontinued *Web Matrix*: a free .NET-based web development environment for Windows;
- *Visual Basic, C#, C++, J++, WebDeveloper* and *Visual Studio Express Editions*, free or nearly-free follow-ons from Web Matrix for later .NET Frameworks.

.NET supports true O-O programming with inheritance, polymorphism, and encapsulation. It offers tremendous language flexibility, debugging support, and multi-level authentication—a must for multi-tiered web applications. If Visual Studio .NET is too rich for your budget or too massive for your environment, you can use *WebMatrix* 3 with a recent version of .NET Framework or a Borland tool like Delphi or C++ Builder. Since .NET is available for virtually any device that can reach the internet, it is an appealing platform for developing leasable on-

demand software. To browse options see *www.microsoft.com/net*. All that power, convenience, flexibility and robustness are strong reasons to consider .NET.

But …

That flexibility, robustness and marketing cost you computing power, efficiency, freedom, modularity and security, and in some releases cost a lot of money.

About the money: Some .NET elements have always been free, permitting development of real applications without a big purchase. Middleware that connects .NET to MySQL databases (*Connector/NET* from MySQL) is free too. Until late 2014, if you wanted the full-fledged Visual Studio .NET development environment, you found that Visual Studio .NET would cost you $500 to $12,000 depending on the package, and the total cost of ownership, including add-ins and necessary updates, would run beyond the affordable range of many a small organisation.

Microsoft's November 2014 release of Visual Studio Community Edition changed all that. The Community edition is now available completely free for …

- any individual developer working on a commercial or non-commercial project,
- any developer contributing to an open source project,
- anyone (student, teacher, classroom, online course) in an academic research or course setting,
- any non-enterprise organisation with 5 or fewer developers working together on a commercial or non-commercial project.

Microsoft also announced open sourcing of core .NET Framework, so it's on its way to Linux and OSX too. Here's an indicator of how far Microsoft has come in a short period of time: the Visual Studio code editor is built on Chromium, the open source version of the Google Chrome web browser, using the open source Electron framework, hosted on GitHub!

About the computing power: You now need a fast processor and several GB of RAM. The more computing power you throw at Visual Studio, the happier you will be.

About efficiency: The magic of .NET database interoperability is an XML layer between code and data source. That mapping costs you CPU cycles, memory and time. So does the huge .NET class hierarchy. ASP and ODBC impose additional length on code paths, are memory-hungry, and execute slowly—serious problems for web applications and their servers.

And if, to keep costs down, you are considering ODBC.NET, there is the double-edged question of database independence. If application code is free of specifics tying it to a particular RDBMS, the database backend can be swapped out like a battery in a car. Isn't that a terrific advantage? Yes and no. Yes to the extent that the application's SQL scripts are limited to entirely portable, vanilla code. No to a similar extent, since the degree to which the SQL code is generic is the degree to which the code fails to take advantage of optimisations available in a specific RDBMS. Database independence is good for portability. Database *dependence* is good for performance.

About modularity: Windows products are designed to work together. and sell one another, and since early in the history of Windows cycle have been built more for corporate environments because that's where the big software purchasing money is. Two big priorities in Windows development have been security and integration, both of which impose deep interdependencies, so most Windows components you acquire, even those that are freely downloadable, have very many installation contingencies and dependencies. The more you work in Windows, the more likely it is that some component you're trying to install or use is out of sync with some other functionality you previously installed and still need. With Visual Studio.NET this problem may

be worse than with other Microsoft software. If you work independently or in a small shop, the costs in time and money imposed by these interdependencies can be big enough to make open source alternatives to Visual Studio, and to .NET itself, look very attractive.

About freedom and security: Those modularity issues have a lot to do with the susceptibility of Windows to viruses, worms, hacks and trojans. ASP.NET officially requires that you deploy on Microsoft's IIS, which is infamous enough for security vulnerabilities to inspire a steady march away from it, especially to PHP and Apache. Whether IIS vulnerabilities are due more to Windows architecture, bad code or to hacker preferences, the problems are serious, they continue, and they cost.

You may be able to use .NET and sidestep IIS. Covalent Technologies discontinued its Apache web server that supported ASP.NET, but you can run a website on Apache and relay requests to IIS running internally on another machine. Or you can switch to Linux/UNIX and develop ASP.NET applications using Novell's *Mono*.

Table 15-1: Apache/PHP or .NET?

	Apache/PHP	***.NET***
Price of core elements	zero	zero
Price of development tools	zero	Zero for Community ed.
OOP	weak	strong
Open-source	yes	no
Development GUI	good (3rd party)	excellent
Rich UI data controls	none	many
Support, bug fixes	free, quick	Well, it's Microsoft
Speed	good	fair
Efficiency	good	poor
Security	good	good
Vulnerability	small	large
Exception management	since PHP 5	yes

The choice between PHP/Apache and .NET is a matrix (Table 15-1). .NET is extremely powerful and versatile. It is rigorously O-O, designed with the enterpise in mind, supported by the biggest software company on the planet, and here to stay. The Visual Studio IDE is spectacularly good. Total cost of PHP/Apache ownership is much lower, though, because full-featured downloads are free, upgrades are free, many add-ins are free or nearly so, and there is a huge and helpfully responsive user community. No open-source development company will charge you money to report a bug, and the PHP and Apache open-source development groups are very good at correcting bugs quickly. PHP and Apache run on every platform you are likely to work on or port to. Which line items in the matrix weigh most heavily in your situation will determine whether .NET should be your choice.

.NET architecture

Windows and Java are the two most popular programming environments on the planet. Java enjoys three advantages: it is object-oriented bottom to top, it manages code to prevent dangling pointers and similar errors, and it runs on any machine that runs a *Java Virtual Machine* (JVM), regardless of operating system.

.NET is a Microsoft response to Java, meant to provide for Windows what Java provides for a Java-enabled operating system, plus language neutrality—you do not need to learn one particular language, and you should be able to port much Windows code to .NET without a lot of rewrites.

.NET Framework interfaces inherit multiply, its objects inherit singly. There is support for events and properties. A fundamental concept is the *namespace*, which is a collection of classes, interfaces, enumerations, and delegates. The root object is System.Object. The *Common Type System* (CTS) defines basic value types, type composition and safety, classes, interfaces and delegation. The *Common Language Specification* (CLS) is a subset of CTS with which .NET languages comply. Microsoft has promised that all future APIs will comply with CLS.

The *Common Intermediate Language* (CIL) is the intermediate code to which all .NET languages compile in executables called *assemblies*, which are DLLs or EXEs having a header that indicates the presence of CIL code and .NET metadata. The *Common Language Runtime* (CLR), written in C++, includes the CTS, the garbage collector, exception handlers, and a module which just-in-time-compiles assembly CIL byte codes to native object code.

The *Framework Class Library* (FCL) is the .NET runtime library. No more Win32 API, which no-one will miss. Of particular interest to MySQL developers are .NET Framework database and .NET web classes. Database classes inherit from System.Data and implement ADO.NET. There are three DataProvider groups: System.Data.Odbc, System.Data.OleDb, and direct data providers. Web classes inherit from System.Web, and implement ASP.NET.

For database applications, data modelling in the development tool is crucial. The core .NET data model is in the System.Data namespace, structuring classes for data source connection, query submission, data manipulation, and result processing. It serves as a hierarchical data cache, serviceable online and offline.

A ***DataSource*** may be a database, an XML file, or any other data store for which the .NET data interface has been implemented. Out of the box, .NET implements these interfaces as managed DataProviders for Microsoft SQL Server, OLE DB, Oracle, and ODBC. So out of the box, you can use .NET with MySQL via ODBC once you install MySQL's *Connector/ODBC* (*Chapter 11*). MySQL does not support Ole DB, but publishes the native managed MySQL DataProvider *Connector/NET* for all versions of .NET Framework.

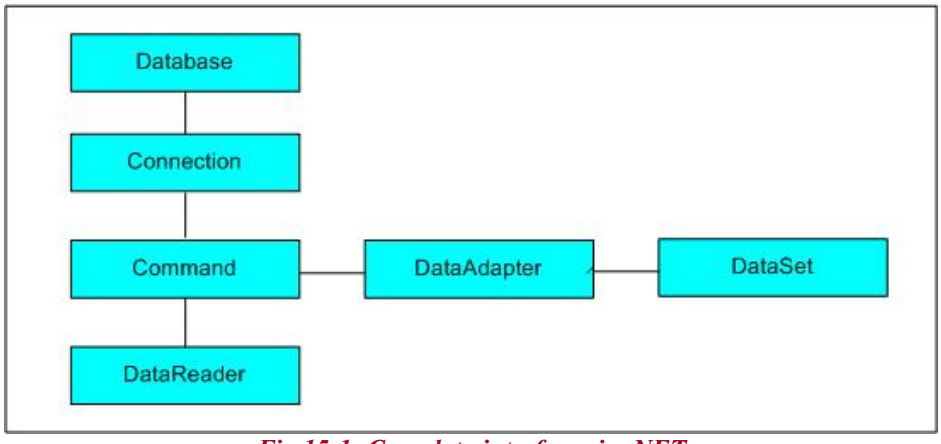

Fig 15-1: Core data interfaces in .NET

To connect to a database, you need a ***Connection*** object for that database. For SQL Server there is the built-in *System.Data.SQLClient.SQLConnection* namespace, for MySQL the namespace specified by the provider, for example *System.Data.MySQL. MySQLConnection*. In each case you tailor a ***Command*** object to specify connection parameters. A subclassed ***DataAdapter*** serves as a helper object linking Command and DataSet objects.

The ***DataSet*** object is a memory-resident store for data tables, their relationships, their constraints, and of course their data. It provides a consistent relational programming model that is independent of the data source. It represents its data as a set of ***DataTables***, which can be defined to exactly mirror underlying database tables, or to combine them in whatever way is desired. It can also be customised, by way of an XSD schema, into a typed DataSet.

The data stored in the DataSet object is asynchronous with its underlying database. Changes are cached. To decide whether there are changes, canvass *DataSet. HasChanges()*.To send changes back to the database, pass the entire object to the middle tier server, or call DataSet.GetChanges() to extract modified rows from a DataSet so only changed rows are sent back.

.NET and MySQL

To develop MySQL-enabled .NET applications, you need several bits and pieces of software, starting with a current version of the .NET Framework, downloadable *here.* For early .NET releases you also needed a current version of Microsoft Data Access (MDAC). For big projects or team development, you will want the Visual Studio IDE.

On the MySQL side, two methods for connecting .NET applications to MySQL databases are supported: ODBC.NET, and native .NET providers: So you need one or both of …

Connector/ODBC (*Chapter 3*, *Chapter 11*)if you wish to connect to MySQL from Visual Studio via ODBC.NET, or if you wish your application to use ODBC;

Connector/.NET: Versions and how they match up with various releases of MySQL, .NET and Visual Studio are described *here*.

Use the *MySQL all-in-one installer* to install MySQL Server, Connectors, Workbench, Notifier, and MySQL for Excel and Visual Studio all at once.

TOC ***Previous*** ***Next***