myCOAL ePermitting System Design

# Background

In one of my previous jobs I was put on a team of 8 developers to maintain a huge legacy system. It had been worked on by many different developers for nearly a decade. Simply finding your bearings in this system was challenging, and making updates was intimidating. The policy there was “you break it you own it”, but it was really more of a “you look at it you own it” kind of place. I was new enough to look, and I found a maze of code spread between a database and an application. Several of the functions contained thousands of lines of code that updated global variables that affected the entire system. Much of the logic had spilled over into the database and I found stored procedures with thousands of lines of code.

The short version is that nobody understood the system, and even small changes took weeks to test and sort out. It was incredibly expensive to maintain that system, but replacing it was even more. I was interested in how something like that happens and wanted to learn how to avoid getting into that situation myself one day.

The biggest problem I found was the difficulty in find what you were looking for, and not knowing what side effects your changes might have. Everything ran together. A typical process would start with a call to a stored procedure. You’d open the stored procedure and wade through a complex decision tree that could have several outcomes based on the parameters sent in and values in the database. The application code was worse. You’d start at the top of a very long function which had nested loops that updated global variables that affected things outside the function. I sometimes spent days trying to find the problem lines of code and chasing down tangents. A simple fix would create new problems. That cycle had been going for who knows how many years. It was terrible.

I came up with a few rules to avoid that situation in my projects:

1. Don’t ever use global variables
2. Keep functions short enough to print on a single page of paper
3. Keep business logic out of the database (no complex stored procedures)
4. Keep data access, business logic, and presentation code isolated from each other

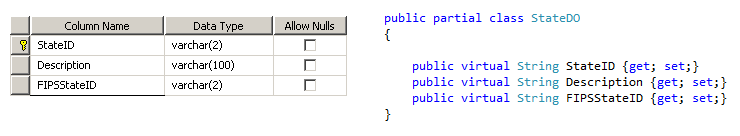
# Code Generation (DALAPI)

Following these rules turned out to be more challenging that I though. The problem was that when I was developing code I wrote what I needed when I needed it, which is human nature. I’d find myself writing a query to get data followed by a few if statements in the code behind to my form. These functions would start off simple but quickly grow quite large. And then would come the global variables, a telltale sign that things are getting out of hand.

I needed to do something different. My natural way of coding didn’t follow my rules. In order to help myself follow my own rules I created a program called Dalapi. It’s a code generator that produces stored procedures and data access code from a database schema. It guarantees that your data access is separate from anything else, and it follows a standard format – it creates a *Data Access Layer API* for your database. It also saves a ton of time. You don’t have to write any SQL or code to execute SQL. One line of code is all you need to get data from the database or update the database.

# Data Objects (DO)

A data object is simply a .net version of a row in a database table. It has a property for each table column with the same name and the .net equivalent of the data type. In the example below a varchar sql cell is a string in the data object. Data Objects are kind of like DataRow objects in a DataTable object, except they’re strongly typed and don’t have any extra stuff.



# Data Access Layer (DAL)

The DAL is the generated API to the database. It provides the CRUD operations that you typically need to perform on a table. Each table always has a GetAll, GetByPK, Create, Update, and Delete method. Additionally, a GetBy\_IndexName method is also included for every index on the table. These methods take Data Objects for parameters and return arrays of Data Objects. They do all the grunt work for getting data into and out of a database. Each method on a DAL class executes a corresponding stored procedure in the database.

The DAL resides in its own project within the solution. Dalapi generates the C# code in the CSharp folder. The code includes the files for the DO and DAL classes. It also generates stored procedure files in the SQL folder. These SPs are automatically run on the database, so the files are only for reference. It also generates a Tables folder which contains definitions for the tables, again for reference.

# Business Objects (BO)

The generated Data Access Layer is great because it saves you from spending a ton of time doing very repetitive work, but the functionality that it provides is not quite ready for application development. For one thing, a typical screen in an application uses data from several tables in the database. This is where Business Objects come to play.

Business Objects combine data objects into a form that the front-end application data needs, like a parent data object and a list of child data objects.

# Business Logic Layer (BLL)

The Business Logic Layer is where the brains of the application reside. It manages the database using the DAL. It does the work of combining Data Objects into Business Objects that are needed by the front-end application. It also takes Business Objects from the front-end and parses them back into Data Objects to update the database. It is the middle man between the database and the front-end application.

The BLL also enforces rules required by the business. For example, your business may require you create a record in table B & C whenever you update table A.

Like the DAL, the BLL is also its own project in the solution, and it contains the Business Object and the BLL classes. The BLL project references the DAL project.

# Presentation Layer (GUI)

This is the front-end application. It can be a windows form, a DOS console, a web application, an app… In our case it’s an MVC application. One of the big advantages of doing all the work mentioned up to now is that you don’t have to do all that work in the front-end. This is important because the front-end is what gets beaten up the most. You’ll never get a group to agree on what the front-end application should look like or how it should work, but you can get that same group to agree on how the business operates (I didn’t say it would be easy).

The point is that the front-end application is going to be retired eventually. Something better will come out and it will become obsolete. Give it 5 years and it will start showing its age. The nice thing for us is that the business processes don’t change as quickly as technology, meaning we can reuse the BLL for our next generation of the application.

The third project in our solution is the presentation layer. It references both the BLL and DAL.

# Sand Box

Sand Box is an example of this system architecture that you can use to experiment with all the technology and not worry about accidentally messing something up or being confused by all the implementation details of the ePermit system.

## Setting Up The Sand Box

Use GitHub to sync the Sand Box solution. <https://github.com/nathantownsend/SandBox>

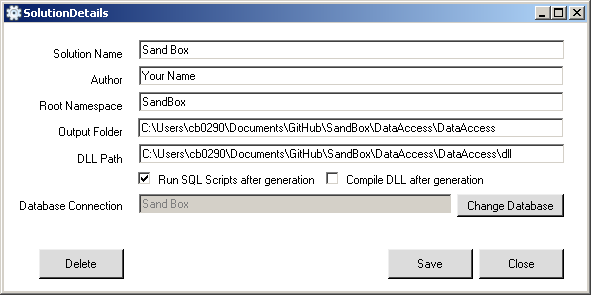
## Create the database

Create a new SQL Server database called SandBox. We’re using 2012, but 2008 or 2005 should be fine, including Express editions (which are free). Next, run the Create userTable.sql script that is included in the zip.

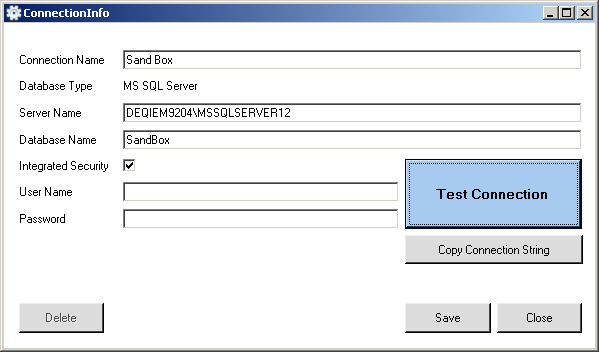
## Setup Dalapi

I included the compiled version of Dalapi in the Resources folder under the SandBox repository. Create a shortcut on your desktop to the DalapiWindowsClient.exe file located in the Dalapi folder under Resources. I haven’t take then time to create a deployment, so this is the old fashioned approach.

Run Dalapi from the shortcut. On the main screen click the New button and setup as follows. Leave Compile DLL unchecked.



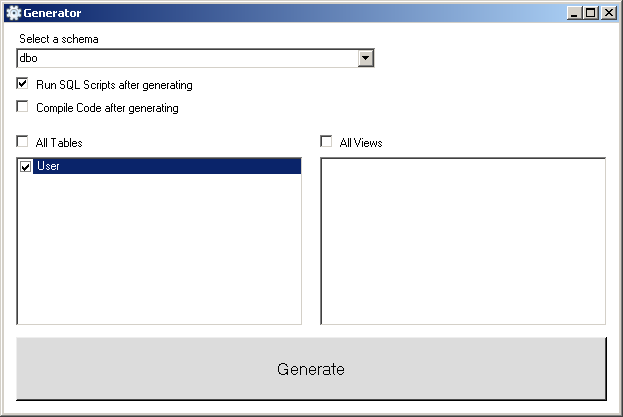
Then click Change Database and click New on the Database Connections window and enter your database information. Mine is shown for example. Use the Test Connection button to make sure it works. Once it is working click the Copy Connection String button and past it somewhere for later.



Once your connection is working click Save, then click Select from the Database Connection screen (which should show Sand Box as the selected database). Finally, click Save on the Solution Details screen to complete the setup.

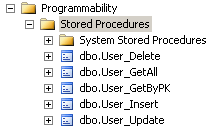
## Run Dalapi

After setup the main screen should have Sand Box as the selected solution. Click Load. Select dbo from the schema drop down at the top. Check the User table and click Generate at the bottom.



## Verifying Code Generation

Once the generator has run it will display a Message box saying your code has been generated. To verify open the database and verify the stored procedures were created:



## The SandBox Solution

Open the Sand Box solution in Visual Studio 2012. You’ll see the three projects named Business Logic, Data Access, and Presentation. In the Presentation project open the web.config file at the bottom of the list. Change the connection string to your database. You can get the connection string from the connection info screen in Dalapi.

Press F5 to ensure the solution runs. Once everything is setup and running, try to work through the assignments in the Sandbox.