Dealing with Passwords

The passwords returned are a Base64 string with a length of 64. The first 32 characters are a randomly generated salt for each password and the remaining 32 characters are a PBKDF2 (Password-Based Key Derivation Function 2) hash with 50,000 iterations used.

The implementation for this algorithm is found in the **Rfc2898DeriveBytes** class provided by **.NET**, for more information see here:

Rfc2898DeriveBytes

The original raw passwords for these logins are as follows:

LoginID (Customer Name)	Password (Raw)
12345678 (Matthew Bolger)	`abc123`
38074569 (Rodney Cocker)	`ilovermit2020`
17963428 (Shekhar Kalra)	`youWill_n0tGuess-This!`

NOTE: The raw passwords are only shown here so you can use them when logging in, you are **not** to store this information anywhere in your source code or database.

When verifying passwords with user input against values stored in the database it is recommended you use the **SimpleHashing** package, alternatively you can use the **Rfc2898DeriveBytes** class directly. The **SimpleHashing** package is an easy-to-use wrapper for the **Rfc2898DeriveBytes** class in **.NET** with no modifications to the hashing algorithm and is the recommended approach.

Use the boolean return value of the **SimpleHashing.PBKDF2.Veryify** method to determine if an entered password matches the salted & hashed value stored in the database.

Consider the following hard-coded example demonstrating its use:

```
// Include the following namespace:
using SimpleHashing;
...

// Example of what is stored in the PasswordHash field within the Login table.
string passwordHash =
    "YBNbEL4Lk8yMEWxiKkGBeoILHTU7WZ9n8jJSy8TNx0DAzNEFVsIVNRktiQV+I8d2";

// These strings represent examples of user input:
string passwordCorrect = "abc123";
string passwordIncorrect = "idontknowthepassword!";

// Output is: True
Console.WriteLine(PBKDF2.Verify(passwordHash, passwordCorrect));

// Output is: False
Console.WriteLine(PBKDF2.Verify(passwordHash, passwordIncorrect));
```