ADO.NET 4.5

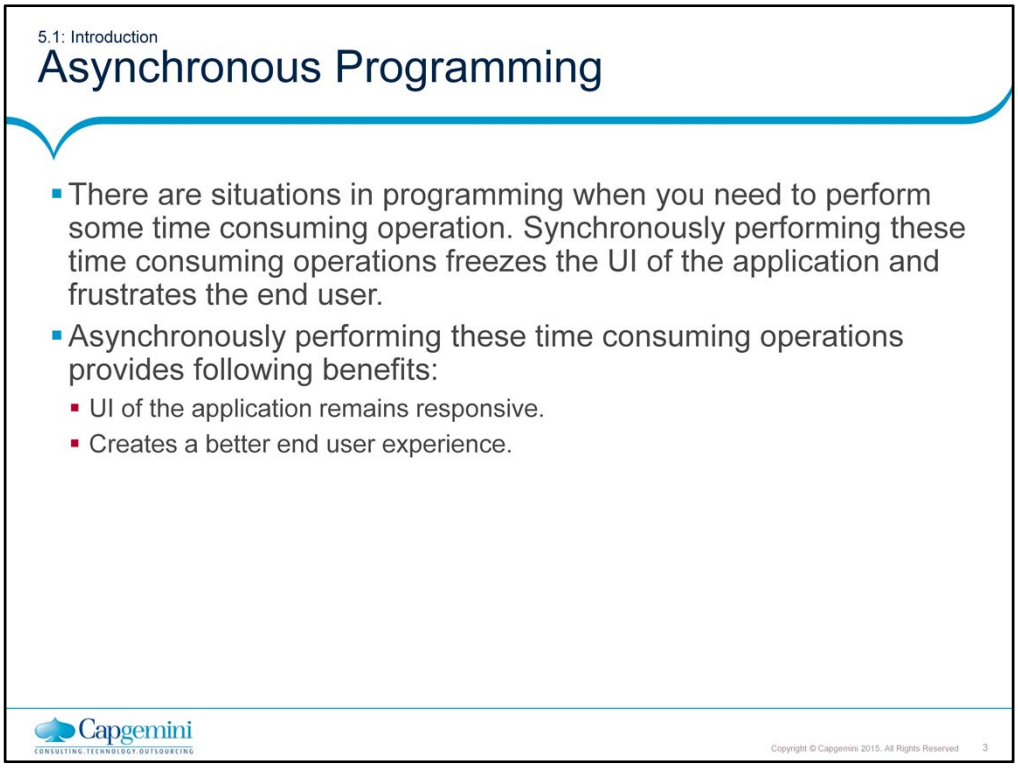
Lesson 5: Asynchronous Data Access

Lesson Objectives

- In this lesson, you will learn about:
  - Asynchronous Data Access in ADO.NET 4.5
  - Sql Credential

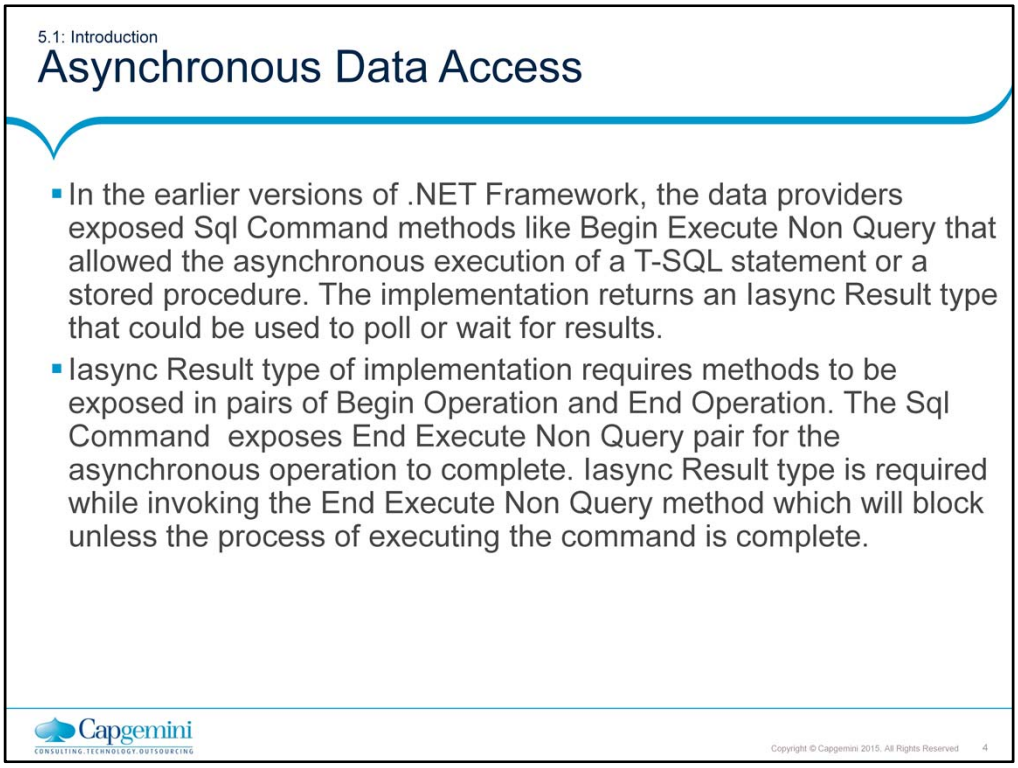
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2



With asynchronous programming new operation can be started and without waiting for this new operation to complete user can continue his work.
E.g.- When you are filling up a registration form, you enter your email address and that email address is verified for its validity. If you do this synchronously, then user won't be able to key in remaining information in the registration form unless email address is validated.

Asynchronous Programming can be used in following situations:
For performing some time consuming database operations
For performing some time consuming IO operation (Read/Write operation on a file).




```
using (var connection = new SqlConnection("..."))
{
try
{
var command = new SqlCommand(commandText, connection);
connection.Open();
var callBack = new AsyncCallback(CallBack);
var result = command.BeginExecuteNonQuery(callBack, command);
while (!result.IsCompleted)
{
//TODO: Continue to perform your other operations here
}
}
catch (SqlException)
{
//Log Exception
}

catch (Exception)
{
//Log Exception
}
}
```

You will notice that the main thread is not blocked, and you can continue to see the results inside the while loop
that checks for the IAsyncResult IsCompleted property. A simple callback handler will look like the following:

```
private static void CallBack(IAsyncResult result)
{
try
{
var command = (SqlCommand)result.AsyncState;
command.EndExecuteNonQuery(result);
}
catch (Exception)
{
//Log Exception
}
}
```

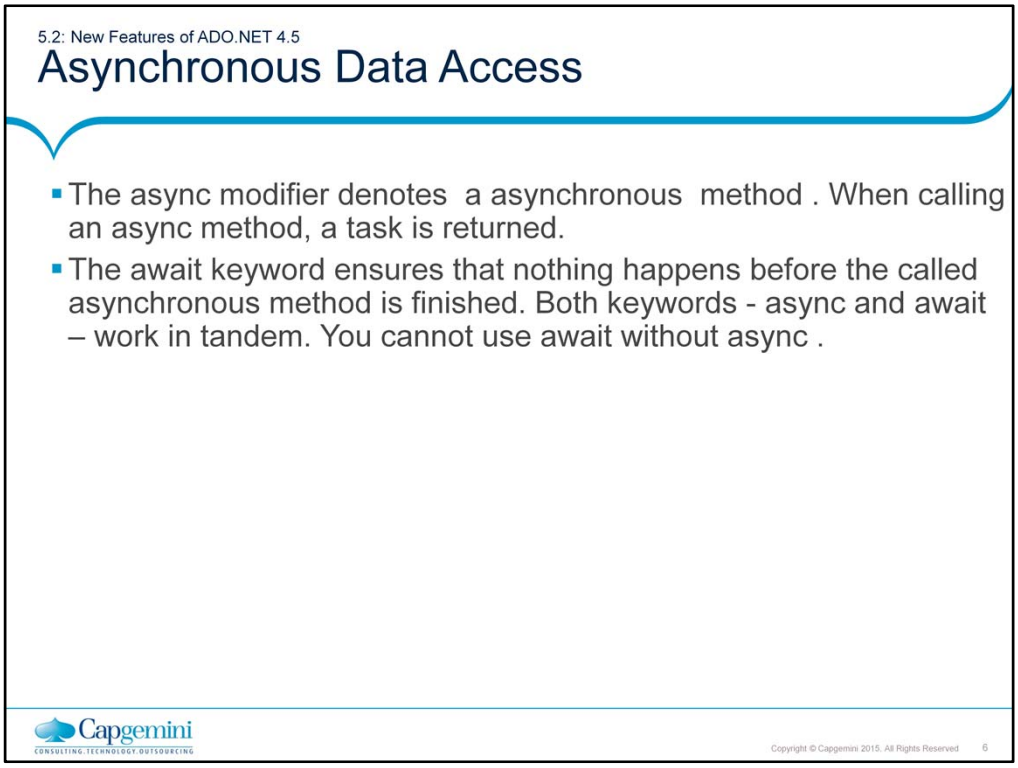
5.2: New Features of ADO.NET 4.5

Asynchronous Data Access

- Till .NET Framework 3.5 you need to specify a callback method that will be called to do some processing after asynchronous operation finishes.
- ADO.NET 4.5 introduces an overly simplified async programming model where you can perform asynchronous calls without using callbacks. The async and await modifiers in .NET Framework 4.5 are used for this purpose.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 5



For a simple asynchronous execution of the command, this is lot of code. You will find out how this is simplified in .NET Framework 4.5.

The ADO.NET Async programming model in .NET Framework 4.5 exposes an equivalent asynchronous method for every synchronous method exposed by the data providers. Continuing with the previous example, the ExecuteNonQuery method of SqlCommand class has an equivalent asynchronous method named ExecuteNonQueryAsync. It accepts a CancellationToken as a parameter allowing the operation to be aborted before the command timeout elapses.

Now let's take a look at how the programming model significantly reduces the lines of code you have to write for the same operation.

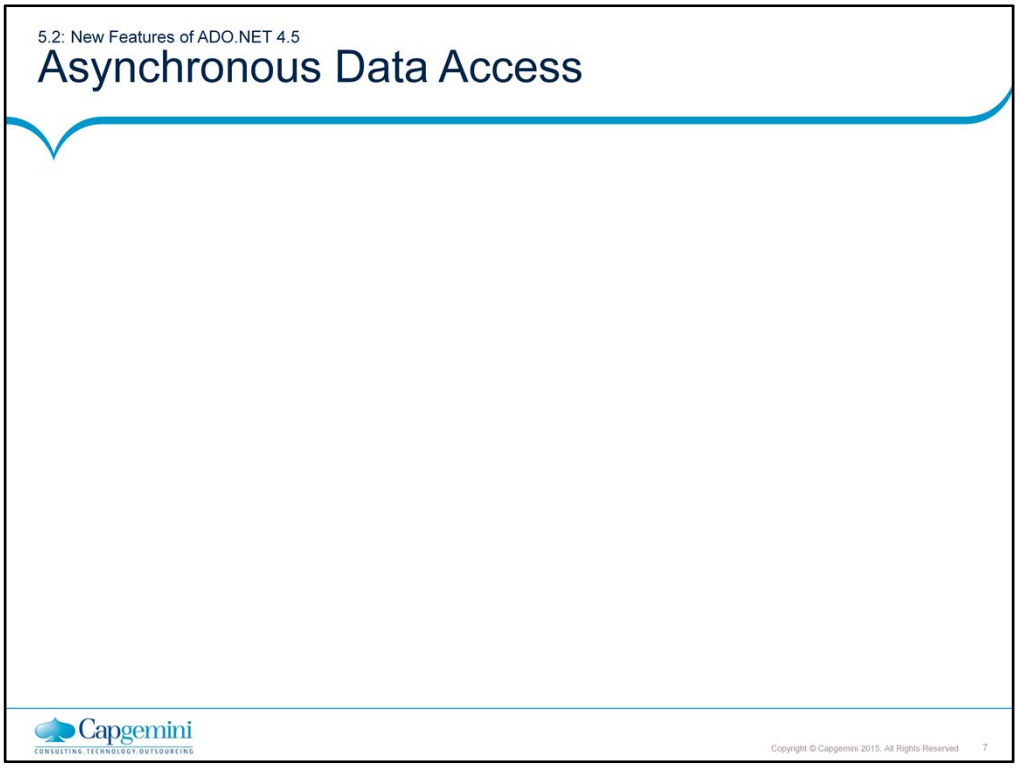
```
private static async Task<int> ExecuteCommandAsync(SqlConnection connection, SqlCommand command)
{
await connection.OpenAsync();
await command.ExecuteNonQueryAsync();
return 1;
}
```

A caller method can pass the connection and command instances to call this method and execute the T-SQL statement or procedure.

```
using (var connection = new SqlConnection("..."))
{
try
{
var command = new SqlCommand(commandText, connection);
int result = ExecuteCommandAsync(connection, command).Result;
}
catch (SqlException)
{
//Log Exception
}
catch (Exception)
{
//Log Exception
}
}
```

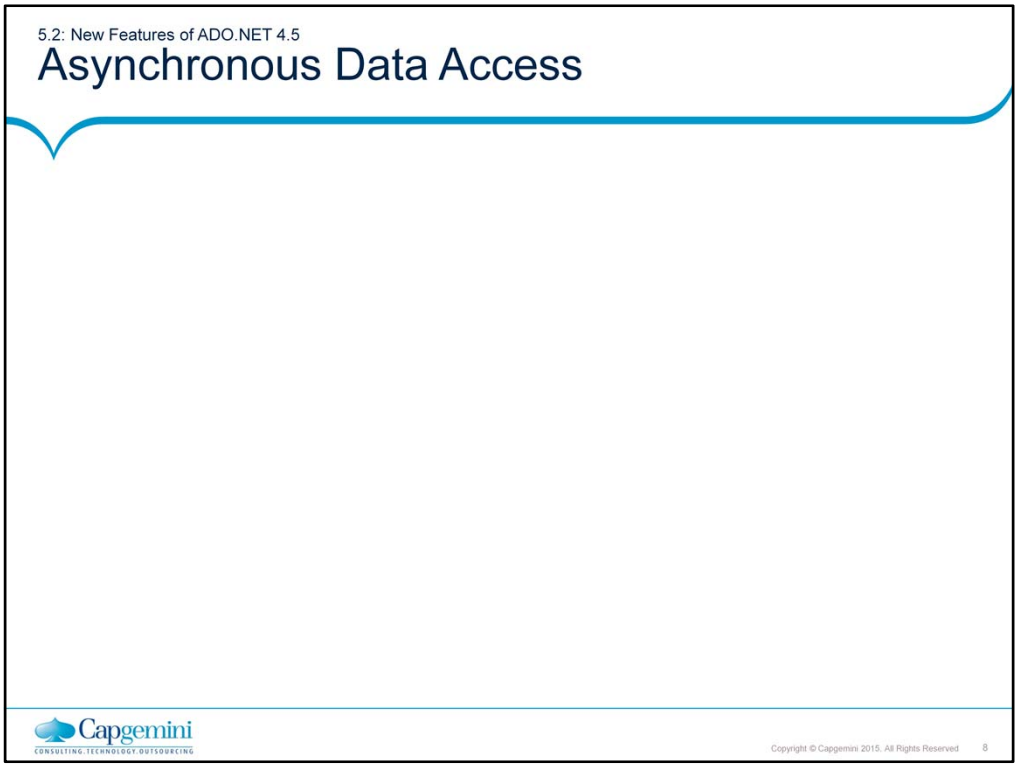
The async programming model is extremely robust, and it allows you to execute a task-based chain of commands. You could potentially use the new async pattern along with your existing asynchronous implementation using the older model. Here is an example using the code bits used previously:

```
using (var connection = new SqlConnection("..."))
{
try
{
AsyncCallback callBack = new AsyncCallback(CallBack);
connection.OpenAsync().ContinueWith((task) =>
{
SqlCommand cmd = new SqlCommand("...", connection);
IAsyncResult ia = cmd.BeginExecuteNonQuery(callBack, cmd);
}, TaskContinuationOptions.OnlyOnRanToCompletion);
}
catch (SqlException)
{
//Log Exception
}
catch (Exception)
{
//Log Exception
}
}
```



Previously, asynchronous programming with the SqlClient data provider was achieved by adding Asynchronous Processing = true to the connection string and using SqlCommand.BeginExecuteNonQuery() or SqlCommand.BeginExecuteReader(). These methods still exist in .NET 4.5 (though you no longer need the asynchronous command in the connection string), but like elsewhere in the framework, the TPL has been implemented in SqlConnection, SqlCommand, SqlDataReader, and SqlBulkCopier (as well as their underlying abstract classes) to provide asynchronous versions of existing synchronous methods. The new methods follow the standard pattern of appending Async to the method name so, for example, SqlConnection.Open() now has as a corresponding asynchronous method SqlConnection.OpenAsync().
The following example demonstrates some of the async methods that are available.

```
class Program
{
public static void Main()
{
string connString = "Data Source=(local); Initial Catalog=DummyDb; Integrated Security=SSPI";
var cts = new CancellationTokenSource();
using (var conn = new SqlConnection(connString))
{
```




```
var command = new SqlCommand("WAITFOR DELAY '00:00:05';select FirstName from
FakePeople", conn);
ExecuteNonQueryAsync(conn, command)
.ContinueWith(t =>
{
Console.WriteLine("Start reader");
command.ExecuteReaderAsync(cts.Token)
.ContinueWith(async t2 =>
{
if (t2.Status == TaskStatus.Canceled)
Console.WriteLine("Read was cancelled");
while (await t2.Result.ReadAsync())
Console.WriteLine(t2.Result[0]);
});
Console.WriteLine("Do something else while getting the results");
});
Console.WriteLine("Waiting...");
Console.ReadKey();
}
}
public static async Task<int> ExecuteNonQueryAsync(SqlConnection conn, SqlCommand
cmd)
{
await conn.OpenAsync();
Console.WriteLine("Connection open");
await cmd.ExecuteNonQueryAsync();
Console.WriteLine("Query completed");
return 1;
}
}
```

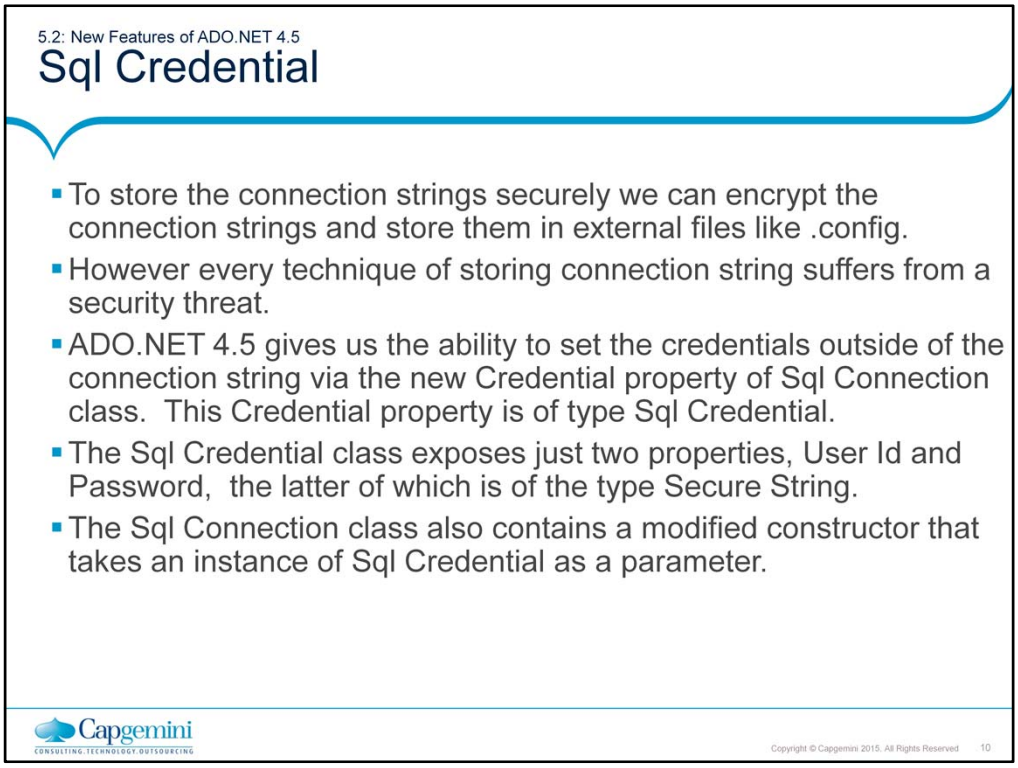
5.2: New Features of ADO.NET 4.5

Sql Credential

- Strings have always attracted interest of the hackers because
  - String data type is commonly used to store general purpose literals like connection string.
  - Large number of developers use string data type to store hard coded values like promotion codes, license keys etc.
- The information stored as string can be easily viewed by using tools like ILDASM or Windbg.
- Connection strings stored as strings are particularly vulnerable as they contain sensitive information like user name and password that will be used while connecting to database.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 9



Here is an example of how it can be used:

```
private void LoginButton_Click(object sender, RoutedEventArgs e)
{
var connString = "Data Source=(local);Initial Catalog=DummyDb";
var password = PasswordText.SecurePassword;
password.MakeReadOnly();
var sqlCredential = new SqlCredential(UsernameText.Text,password);
string message = "Successfully logged in";
try
{
using (var connection = new SqlConnection(connString,sqlCredential))
{
connection.Open();
}
}
catch (Exception ex)
{
message = "failed to log in";
}
MessageBox.Show(message);
}
```

The key thing to note is that the SqlCredential constructor will only allow you to pass an instance of SecureString that has been marked as read only.

Summary

- In this lesson, you have learnt:
  - New Features of ADO. NET 4.5, namely:
    - Asynchronous Data Access
    - SqlCredential

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 11

Review Question

- Question 1: The ___ and ___ modifiers in .NET Framework 4.5 are used for asynchronous data access.
- Question 2: Sql Credential class exposes two properties namely _____ and _____.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 12