**ASYNCHRONOUS PROGRAMMING WITH ASYNC AND AWAIT**

***Ivan Naydonov, Leonid Rebenok***

National Technical University of Ukraine “Kyiv Polytechnic Institute”

You can avoid performance bottlenecks and enhance the overall responsiveness of your application by using asynchronous programming. However, traditional techniques for writing asynchronous applications can be complicated, making them difficult to write, debug, and maintain.

Visual Studio 2012 introduces a simplified approach, async programming, that leverages asynchronous support in the .NET Framework 4.5 and the Windows Runtime. The compiler does the difficult work that the developer used to do, and your application retains a logical structure that resembles synchronous code. As a result, you get all the advantages of asynchronous programming with a fraction of the effort.

The async/await keywords make asynchronous programming much easier for developers to digest and implement correctly. They also hide a little bit of complexity that can sneak up and (bug) bite you if you are not paying attention. It is worth reading and applying these best practices when doing .NET asynchronous programming.

Asynchrony is essential for activities that are potentially blocking, such as when your application accesses the web. Access to a web resource sometimes is slow or delayed. If such an activity is blocked within a synchronous process, the entire application must wait. In an asynchronous process, the application can continue with other work that does not depend on the web resource until the potentially blocking task finishes.

The Async and Await keywords in Visual Basic and the async and await keywords in C# are the heart of async programming. By using those two keywords, you can use resources in the .NET Framework or the Windows Runtime to create an asynchronous method almost as easily as you create a synchronous method. Asynchronous methods that you define by using async and await are referred to as async methods.

In async methods, you use the provided keywords and types to indicate what you want to do, and the compiler does the rest, including keeping track of what must happen when control returns to an await point in a suspended method. Some routine processes, such as loops and exception handling, can be difficult to handle in traditional asynchronous code. In an async method, you write these elements much as you would in a synchronous solution, and the problem is solved.

Introducing the new syntax, Microsoft offers not just syntactic sugar, but a pattern of asynchronous programming, which includes the handling of exceptions and method of termination of an asynchronous function, and information on the progress of the implementation

**References:**

1. Wagner, J. (2012). Best Practices for c# async/await. Retrieved Februar 26, 2013, from http://code.jonwagner.com/2012/09/06/best-practices-for-c-asyncawait/
2. Microsoft (2012) async (C# Reference). Retrieved Februar 26, 2013, from http://msdn.microsoft.com/en-us/library/vstudio/hh156513.aspx