# Getting a Cannot await void, on a method that I have want to await on

4

1

I'm on a team writing a WPF app. We have to make it so that when a user hides/shows different columns that it will reflect that in a ReportViewer control on one of the views. In testing we've found that it takes a long time to add the data to the ReportViewer's data sources; sometimes on the order of several seconds to possibly a minute. Too long I think for the users. So I'm trying to use C#'s async and await. However, when I applied await to the line that is the process hog and then compile it, I get an error from the C# compiler, "Cannot await 'void'". In this case, I cannot change what the .NET framework returns, its void. So how do I handle this situation? Here's the code:

```
private async Task GenerateReportAsync()
{
    DataSet ds = new DataSet();
    DataTable dt = new DataTable();
    dt.Clear();
    int iCols = 0;
    //Get the column names
    if (Columns.Count == 0)     //Make sure it isn't populated twice
    {
        foreach (DataGridColumn col in dataGrid.Columns)
        {
            if (col.Visibility == Visibility.Visible)
            {
                Columns.Add(col.Header.ToString());     //Get the column heading
                iCols++;
            }
        }
    }
    //Create a DataTable from the rows
    var itemsSource = dataGrid.ItemsSource as IEnumerable<Grievance>;
    if (this.rptViewer != null)
    {
        rptViewer.Reset();
    }
    rptViewer.LocalReport.DataSources.Clear();
    if (m_rdl != null)
    {
        m_rdl.Dispose();
```

```
CoreUtils.ToDataTable(itemsSource)));
    rptViewer.RefreshReport();
}
```

c# wpf asynchronous async-await

asked Feb 14 '17 at 18:37

Rod
**1,525** 7 33 60

6 You must have some false belief about what `await` does if you think that awaiting something that takes a long time is a solution to your problem. Can you describe what your beliefs are about `await` ? It only makes sense to await something that is *already* asynchronous; do you believe that await *makes* something that is not asynchronous into an asynchronous operation? It does not. – Eric Lippert Feb 14 '17 at 18:44

2 It is my understanding that using async/await makes it possible to return control to the UI thread, rather than hang it up, waiting for some long working process to finish. But thank you for asking. – Rod Feb 14 '17 at 20:40

My point is that if the function is void returning *and* already asynchronous then *it should not be taking much time*. If it is void returning, synchronous, and high latency then there is nothing to `await` . Await manages an existing asynchronous operation. If you have a high latency operation that you'd like to be asynchronous, `await` isn't going to help you. You're going to have to figure out how to make it asynchronous via some other mechanism. – Eric Lippert Feb 14 '17 at 21:12

## 2 Answers

8

For methods that are inherently synchronous, you need to wrap them in your own `Task` so you can await it. In your case, I would just use `Task.Run` :

```
await Task.Run(() =>
{
   rptViewer.LocalReport.DataSources.Add(new ReportDataSource("MyData",
CoreUtils.ToDataTable(itemsSource)));
});
```

edited Feb 14 '17 at 19:45

answered Feb 14 '17 at 18:43

BradleyDotNET
**53k** 8 74 95

3 It is possible that this will throw a cross thread exception because you are accessing UI components like the report viewer from a background thread. – Scott Chamberlain Feb 14 '17 at 18:46

@ScottChamberlain Absolutely, I have noted this. – BradleyDotNET Feb 14 '17 at 19:46

@BradleyDotNET I've taken your suggestion and came up with the following: DataTable dtTmp = null; await Task.Run(() => dtTmp = CoreUtils.ToDataTable(itemsSource)); rptViewer.LocalReport.DataSources.Add(new ReportDataSource("MyData", dtTmp)); I didn't know that using await on a UI component would cause an exception. Thanks for the head's up! The above gives control back to the UI thread quickly. Thanks, again.. – Rod Feb 14 '17 at 20:44

2 @Rod it is not the await that causes the exception it is the Tas.Run that does. – Scott Chamberlain Feb 14 '17 at 21:02

4

I can almost be certain that the entire line is not the slow line, it is much more likely that `CoreUtils.ToDataTable(itemsSource)` is the slow piece and that is the part that needs to be improved.

You did not include the source of `ToDataTable` so I can't say for sure what the best way would be, the first and best option would be write a new version of the function that leverages async calls internally to create a function that allows you to await it.

```
var data = await CoreUtils.ToDataTableAsync(itemsSource)
rptViewer.LocalReport.DataSources.Add(new ReportDataSource("MyData", data));
```

The second, less performant option is do the slow part on a background thread using `Task.Run` then back on the UI thread set the data source as needed.

```
var data = await Task.Run(() => CoreUtils.ToDataTable(itemsSource));
rptViewer.LocalReport.DataSources.Add(new ReportDataSource("MyData", data));
```

answered Feb 14 '17 at 18:45

Scott Chamberlain