

<http://tech.pro/tutorial/800/working-with-the-wpf-dispatcher> 

MAR APR SEP

16 captures

11 Feb 2013 - 23 May 2017

2013 2015 2016

About this capture

[Profile](#) [Network](#) [Posts](#) [Q&A](#) [Jobs](#) [More](#) [Create a Profile](#) or [Login](#)

[How Tech.pro Works](#)



The nerd table is now the cool table.

[learn more](#)Brandon Cannaday  
posted 6 years ago

WPF

C#

.NET 3.0

INTERMEDIATE

## Working With The WPF Dispatcher



Proper use of threads can greatly increase the responsiveness of your WPF applications. Unfortunately, you can't update any UI controls from a thread that doesn't own the control. In .NET 2.0, you used `Control.Invoke()`. Now, we've got something similar but more powerful - the Dispatcher. This tutorial will explain what the Dispatcher is and how to use it.

First of all, you need to know where the Dispatcher lives. Every Visual (Button, Textbox, Combobox, etc.) inherits from [DispatcherObject](#). This object is what allows you to get a hold of the UI thread's Dispatcher.

The Dispatcher is why you can't directly update controls from outside threads. Anytime you update a Visual, the work is sent through the Dispatcher to the UI thread. The control itself can only be touched by it's owning thread. If you try to do anything with a control from another thread, you'll get a runtime exception. Here's an example that demonstrates this:

```
public partial class Window1 : Window
{
    public Window1()
    {
        InitializeComponent();

        CheckBox myCheckBox = new CheckBox();
        myCheckBox.Content = "A Checkbox";

        System.Threading.Thread thread = new System.Threading.Thread(
            new System.Threading.ThreadStart(
                delegate()
                {
                    myCheckBox.IsChecked = true;
                }
            ));

        thread.Start();
    }
}
```

This code creates a Checkbox, then creates a thread which tries to change the checked state. This will fail because the Checkbox was created on a different thread than the one

Likes

3

Views

260k

### ABOUT THE AUTHOR

Brandon Cannaday  
[Follow](#)

Graduate of Rose-Hulman Institute of Technology with a degree in software engineering. Software Consultant and User Experience Expert.

### WANT TO WRITE WITH US?

Anyone can write at Tech.pro. Writing at Tech.pro can help you get recognized, spread knowledge, and inspire others.

[Write with us](#)

### INVITE A FRIEND

Tech.pro is a communications platform and will become more useful to you as your colleagues join the site. Go ahead and invite a few today and start reaping the benefits of membership.

[Invite a Friend](#)

Dear Tech Professional,

Tech.pro has been temporarily deactivated. We will bring Tech.pro back online after our team has grown in sufficient size to support the community.

Sincerely,

16 captures

11 Feb 2013 - 23 May 2017

Go

MAR APR SEP

24

2013 2015 2016



About this capture

So the question is, how do you update the Checkbox from this thread? Fortunately, the Dispatcher gives us the ability to Invoke onto its thread. Invoking probably looks really familiar if you've programmed in .NET 2.0. We actually have an in-depth tutorial on [invoking](#) that you might want to read. Below is code using the Dispatcher that will run and update the Checkbox without throwing an exception.

```
public Window1()
{
    InitializeComponent();

    CheckBox myCheckBox = new CheckBox();
    myCheckBox.Content = "A Checkbox";

    System.Threading.Thread thread = new System.Threading.Thread(
        new System.Threading.ThreadStart(
            delegate()
            {
                myCheckBox.Dispatcher.Invoke(
                    System.Windows.Threading.DispatcherPriority.Normal,
                    new Action(
                        delegate()
                        {
                            myCheckBox.IsChecked = true;
                        }
                    ));
            }
        ));

    thread.Start();
}
```

Now we've introduced the Dispatcher object. The call to [Invoke](#) needs to take a few pieces of information. First is the priority you'd like your work executed with. Next is the delegate that contains the work you actually want to do. If your delegate takes parameters, the Invoke call will also accept an Object or Object[] to pass into the delegate function. It will also accept a timespan that limits the amount of time the Invoke call will wait to execute your code.

The call to Invoke will block until your function has been executed. Depending on the priority you've set, this might take a while. The Dispatcher also has the ability to invoke code asynchronously using [BeginInvoke](#). Let's look at the same example using BeginInvoke.

```
public Window1()
{
    InitializeComponent();

    CheckBox myCheckBox = new CheckBox();
    myCheckBox.Content = "A Checkbox";

    System.Threading.Thread thread = new System.Threading.Thread(
        new System.Threading.ThreadStart(
            delegate()
            {
                System.Windows.Threading.DispatcherOperation
                dispatcherOp = myCheckBox.Dispatcher.BeginInvoke(
                    System.Windows.Threading.DispatcherPriority.Normal,
                    new Action(
                        delegate()
                        {
                            myCheckBox.IsChecked = true;
                        }
                    ));
            }
        ));

    thread.Start();
}
```



Dear Tech Professional,

Tech.pro has been temporarily deactivated. We will bring Tech.pro back online after our team has grown in sufficient size to support the community.

Sincerely,  
Tech.pro Team

#### TECH.PRO DAILY EMAIL

Get the latest content from Tech.pro delivered to your Inbox each day. Simply create an account to subscribe.

[Subscribe](#)

16 captures

11 Feb 2013 - 23 May 2017

Go

MAR APR SEP

24

2013 2015 2016

About this capture

```
    }
    ));

    thread.Start();
}

void dispatcherOp_Completed(object sender, EventArgs e)
{
    Console.WriteLine("The checkbox has finished being updated!");
}
```

BeginInvoke takes many of the same arguments as Invoke, but now returns a [DispatcherOperation](#) that lets you keep track of the progress of your function. In this case, I simply hooked the Completed event that notifies me when the work has been completed. The DispatcherOperation object also lets you control the Dispatcher by changing its priority or aborting it all together.

As I mentioned above, we can limit the amount of time an Invoke is allowed to take by passing the Invoke call a [TimeSpan](#) structure. Here's an example that will wait 1 second for the queued function to complete:

```
public Window1()
{
    InitializeComponent();

    CheckBox myCheckBox = new CheckBox();
    myCheckBox.Content = "A Checkbox";

    System.Threading.Thread thread = new System.Threading.Thread(
        new System.Threading.ThreadStart(
            delegate()
            {
                myCheckBox.Dispatcher.Invoke(
                    System.Windows.Threading.DispatcherPriority.SystemIdle,
                    TimeSpan.FromSeconds(1),
                    new Action(
                        delegate()
                        {
                            myCheckBox.IsChecked = true;
                        }
                    ));
            }
        ));

    thread.Start();
}
```

Unfortunately, there's no way to determine if the invoke finished or timed out from the outside. You can always put code inside the invoked function to determine if it executed.

All right, so we've got our dispatchers working and code is being executed where it's supposed to be. Invokes, however, are kind of an expensive process. What happens when your controls are being updated from both external threads and the main UI thread. How do you know if you're supposed to use Invoke? The Dispatcher object provides a function that tells you whether or not you have to use Invoke. In WinForms you call `InvokeRequired` on the actual control. In WPF, you call `CheckAccess()` on the Dispatcher object. `CheckAccess()` returns a boolean indicating whether or not you can modify the control without Invoking.

16 captures

11 Feb 2013 - 23 May 2017

Go

MAR APR SEP

24


2013 2015 2016

About this capture

```
System.Windows.Threading.DispatcherPriority.Normal,
new Action(
    delegate()
    {
        myCheckBox.IsChecked = true;
    }
));
}
else
{
    myCheckBox.IsChecked = true;
}
```

So now, before I invoke, I check to see if I even need to. If I do, I invoke the function, if I don't, I simply update the control directly.

As you can see, the Dispatcher provides a great deal of flexibility over the WinForms Invoke. There's a lot about WPF's dispatching model that we didn't touch in this tutorial. The [System.Windows.Threading](#) namespace contains a lot more useful objects that extends the power of the dispatcher even further.



Dear Tech Professional,

Tech.pro has been temporarily deactivated. We will bring Tech.pro back online after our team has grown in sufficient size to support the community.

Sincerely,  
Tech.pro Team

### Login

Email Address

Password

Login

### Register

Email Address

First NameLast Name


Password

☐ I want to receive Tech.pro's email newsletter

By joining, you agree to our [terms & conditions](#) & [privacy policy](#)

Register

### Comments (0)

Join the discussion...

Go

MAR APR SEP

24

2013 2015 2016

About this capture

16 captures

11 Feb 2013 - 23 May 2017

Links

Projects

Contact Us

Q&A

Companies

Privacy

Invite Others

My Network

Site Map

Twitter

Linkedin

Facebook

Copyright © 2015 Tech.pro