Exceptions and Threading

Mark Heath
http://markheath.net
@mark_heath

Exceptions

Anticipated

Unanticipated

Where to Catch?

UI Event Handlers

Non UI Threads

Global Exception Handler

Threads

How do I start a task on a background thread?

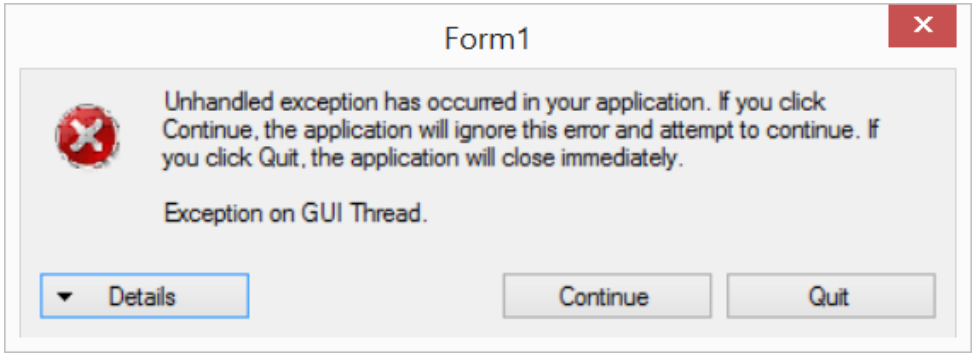
- ThreadPool?
- BackgroundWorker?
- Task Parallel Library?

How do I…

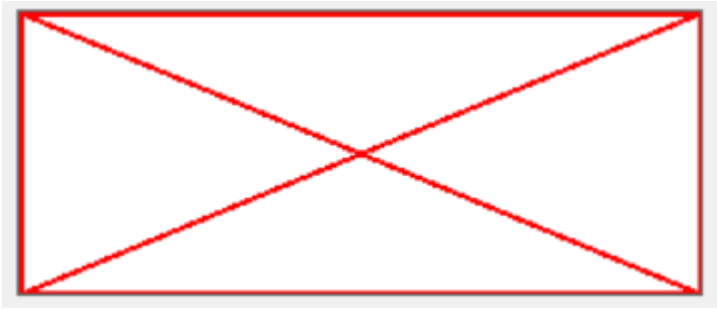
- Update the UI?
- Cancel the task?
- Avoid re-entrancy?
- Report progress
- Report completion?

Unhandled Exceptions

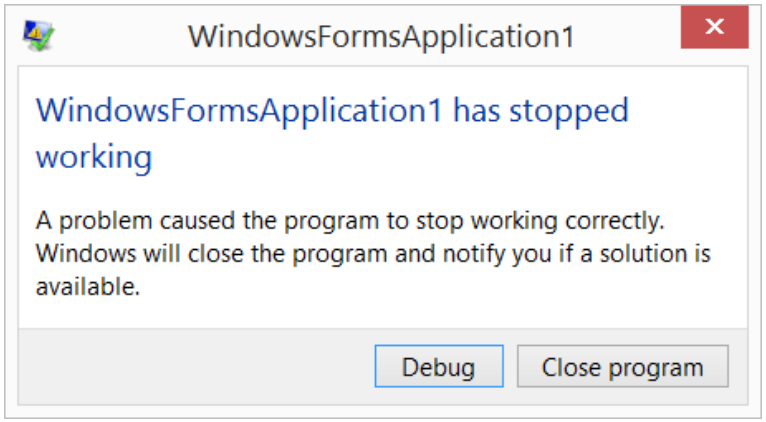
UI Thread
(e.g. Button Click)



Paint Event
Handler



Non UI Thread



Global Exception Handlers

```
Application.SetUnhandledExceptionMode(
        UnhandledExceptionMode.CatchException);


// catch exceptions on UI thread
Application.ThreadException +=
        ApplicationOnThreadException;
// option to continue or call Application.Exit


// catch exceptions on other threads
AppDomain.CurrentDomain.UnhandledException +=
        CurrentDomainOnUnhandledException;
// application will terminate
```

Exception Handling Guidelines

- **Don't use the global exception handler for everything**
  - Only use it
- **Handle excep**
  - Handle as close to source of exception as possible
- **Catch specific exceptions**
  - Only catch if you can do something
- **Log everything**
  - Message
  - Stack Trace
  - Inner Exception

"Defensive Coding in C#"
Deborah Kurata

Where to Handle

- **User initiated actions (e.g. Button clicks)**
  - Report the error to user
  - Perform any cleanup
- **Other UI events**
  - e.g. Timer, Paint, MouseMove
  - Avoid displaying multiple error dialogs
- **Background Threads**
  - Handle for each task

Ways to Start a Background Task

Ways to Start a Background Task

myDelegate.
BeginInvoke()
(APM)

Thread.Start()

ThreadPool.
QueueUser
WorkItem()

background
Worker1.Run
WorkerAsync()

(TPL)

myAsyncFunc

Call EndInvoke to get result and catch exceptions

Can poll for completion with IAsyncResult.IsCompleted

Ways to Start a Background Task

myDelegate.
BeginInvoke()
(APM)

Thread.Start()

ThreadPool.
QueueUser
WorkItem()

- Can set thread name and priority
- Allows us to wait or poll for completion
- May be appropriate for long-running tasks

Worker1.Run
WorkerAsync()

Task.Run()

await
myAsyncFunc

Ways to Start a Background Task

myDelegate. BeginInvoke() (APM)	Thread.Start()	ThreadPool. QueueUser WorkItem()
BackgroundWorker1.Run WorkerAsync()	Task.Run() (TPL)	await myAsyncFunc

Runs immediately or queues for available ThreadPool thread

No completion notification or exception handling

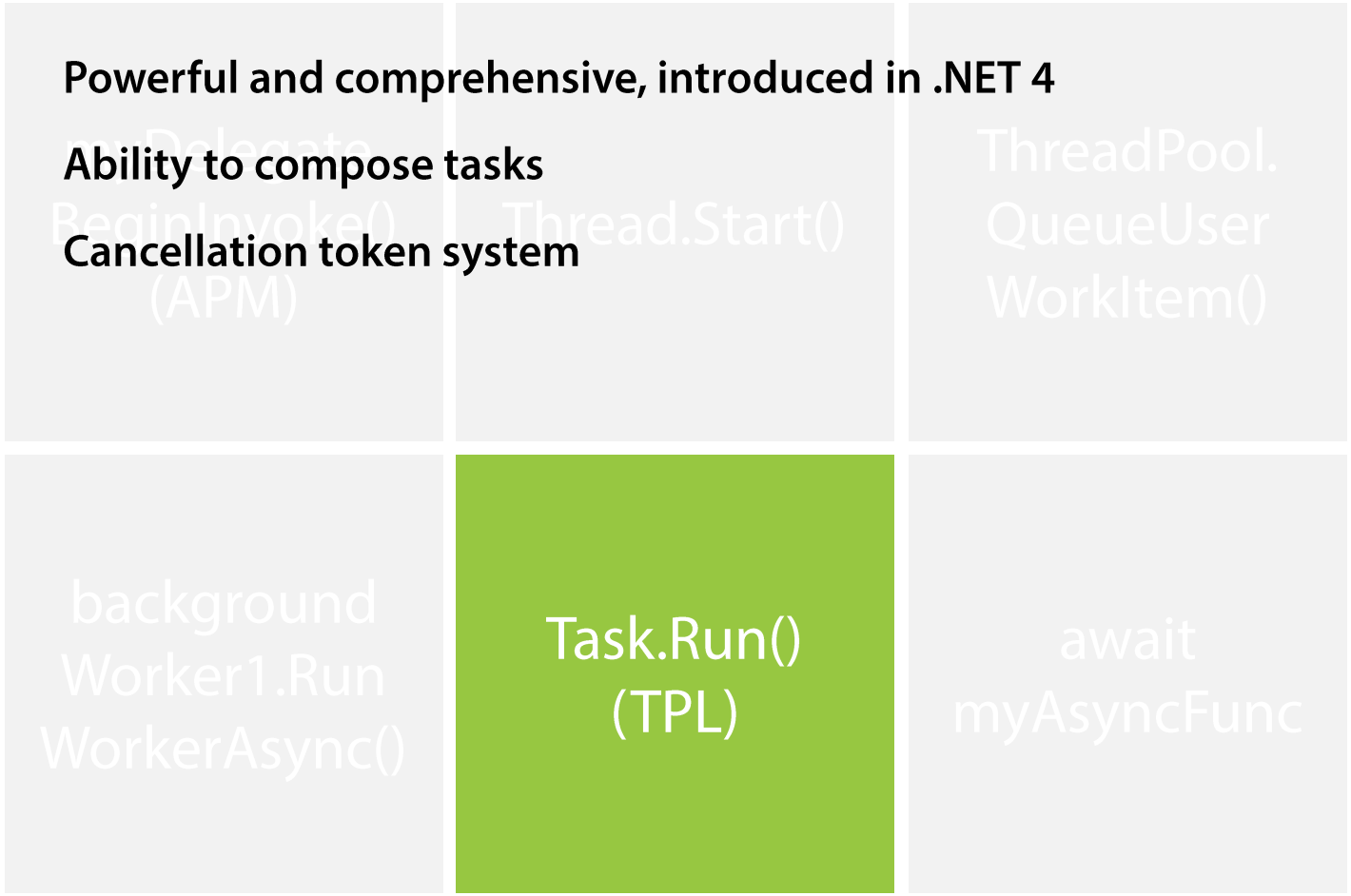
Ways to Start a Background Task

myDelegate. BeginInvoke() (APM)	Thread.Start()	ThreadPool. QueueUser WorkItem()
background Worker1.Run WorkerAsync()	Task.Run() (TPL)	await myAsyncFunc

Notifies us of progress, completion and errors on UI thread

Has a built-in cancellation mechanism

Ways to Start a Background Task



Ways to Start a Background Task

myDelegate.
BeginInvoke()
(APM)

Thread.Start()

ThreadPool.
QueueUser
WorkItem()

backgroundWorker1.Run
WorkerAsync()

Task.Run()
(TPL)

await
myAsyncFunc

async and await keywords part of C# 5, introduced with .NET 4.5

Can be used with .NET 4

Synchronous-looking code flow

Catch exceptions over multiple threads

Continues on UI thread

Ways to Start a Background Task

http://markheath.net/post/starting-threads-in-dotnet

myDelegate. BeginInvoke() (APM)	Thread.Start()	ThreadPool. QueueUser WorkItem()
background Worker1.Run WorkerAsync()	Task.Run() (TPL)	await myAsyncFunc

Updating the User Interface

- **Background threads cannot update UI**
  - InvalidOperationException
- **Solutions**
  - Control.BeginInvoke

```
label1.BeginInvoke((Action)(() => label1.Text = "Hello"));
```

  - SynchronizationContext.Post

```
syncContext.Post(s => label1.Text = "Hello", null);
```

  - Poll with a Timer
  - BackgroundWorker
  - async / await

While a Task is Running

- **Disable any controls to prevent re-entrancy**
  - Disabling a container disables child controls
- **Show progress**
  - `Cursor = Cursors.WaitCursor`
  - ProgressBar
- **Offer a cancelation option**
  - Or automatically timeout

Module Summary

Exceptions

Robustness

Threads

Responsiveness

Best Practices

Handle anticipated exceptions "locally"

Report to the user

Clean up

Provide a global exception handler

Log everything

Exit the application

Perform long-running tasks in background threads

Task Parallel Library (TPL)

async & await

Always update the UI from the UI thread

await long-running tasks