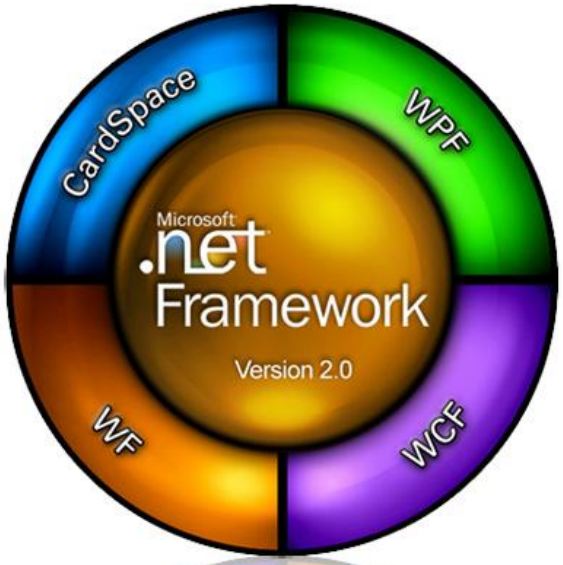


C# Programmierung

Eine Einführung in das .NET Framework

Zeitplan diese Woche

Mo	Di	Mi	Do	Fr
Zeichnen GDI+ Details Threas (CTE) lock, mutex ThreadPools TimerThreads BGW	Process-Class Callbacks DirectX Webservices ASP.NET C# 4 – Tasks WP 7	SDL.NET Lokalisierung Bibliotheken Registry Setup-Projekt Projektideen Projektteams	Arbeiten am Projekt	Arbeiten am Projekt: Release Cand. Präsentation des Projektes

Mo

Zeichnen
GDI+ Details
Threas (CTE)
lock, mutex
ThreadPools
TimerThreads
BGW

Tag 6 ...

Fortgeschrittene Programmierung

Die Macht des Systems ausnutzen – nicht mehr auf Subprozesse warten müssen!

Wiederholung

- Windows Forms – **ToolBox** und **Designer**
- Ohne Designer: Hinzufügen von Elementen
- Eigene Elemente – einfache **OOP**
- Kurze Wiederholung zu OOP und dem .NET Framework (*Arrays*, *Collections*, *Strings*, *Streams*, …)
- Für async Programmierung essentiell: ***Delegates***!

Eigene Steuerelemente 2

- Häufig wird *OnPaint(PaintEventArgs)* überschrieben
- Zeichnen des Steuerelements sehr wichtig. GDI+ wird hier verwendet – bequemer Zugriff in .NET
- Zentrales Objekt: **Graphics**
- Können damit sehr schnell Steuerelemente, Bilder, Druckausgaben, uvm. zeichnen

GDI+ im Detail

- Verwenden das Steuerelement *PictureBox*
- *Koordinatensystem* von .NET beim Zeichnen
- Welche Bildformate werden unterstützt? (quasi alle!)
- Einige Klassen im Detail: **Pen**, **Brush**, **Color**, **Font**
- Nutzen der Grafik-Funktionen wie z.B. *DrawEllipse();*

Noch mehr GDI

- Mehr Pinsel, z.B. Farbverlauf mit *LinearGradientBrush*
- Gerätekoordinatensystem mit *PageUnit*
- Kanten- und Textglättung aktivieren
- Double-Buffering Eigenschaft des Formulars um Flackern zu unterbinden (wie geht das?)
- Koordinatentransformationen mit *Rotate* und *Scale*

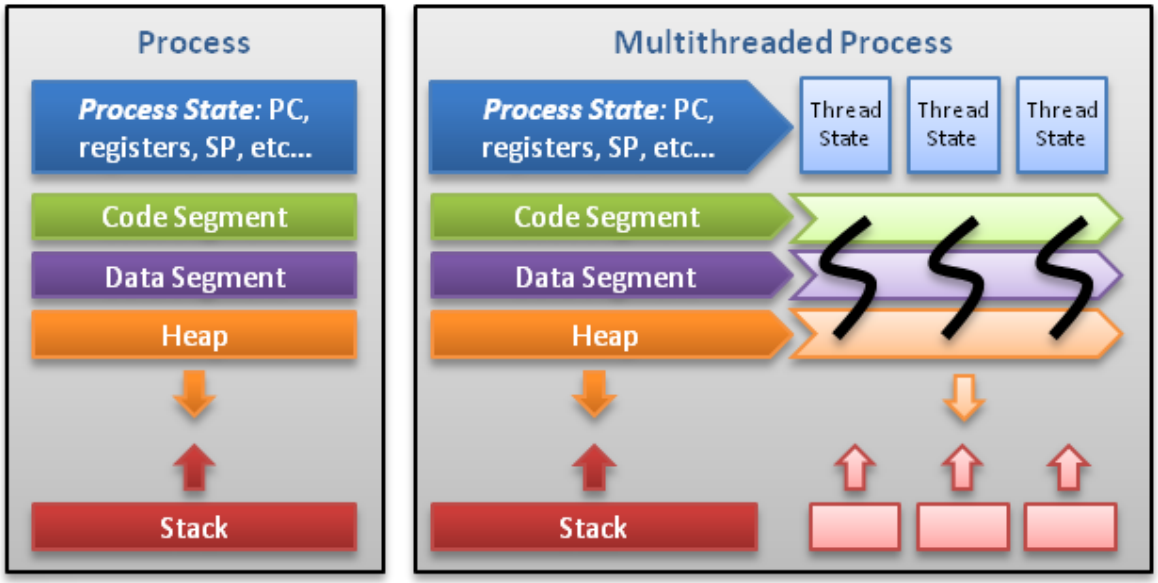
Beispiel

01 – Zeichnen mit GDI+

Das Windows System

- Eingeteilt in **Threads** – täuschen *Multitasking* vor
- **Echtes** Multitasking nur auf Mehrprozessor Systemen
- Threads können verwendet werden um ein Programm in *Aufgaben* zu unterteilen
- Windows gibt den Threads *Rechenzeit* (je nach Prio)
- Im .NET Framework ex. *Thread*-Klasse!

Veranschaulichung



Threads contain only necessary information, such as a stack (for local variables, function arguments, return values), a copy of the registers, program counter and any thread-specific data to allow them to be scheduled individually. Other data is shared within the process between all threads.

© Alfred Park, http://randu.org/tutorials/threads

Erstellen eines Threads

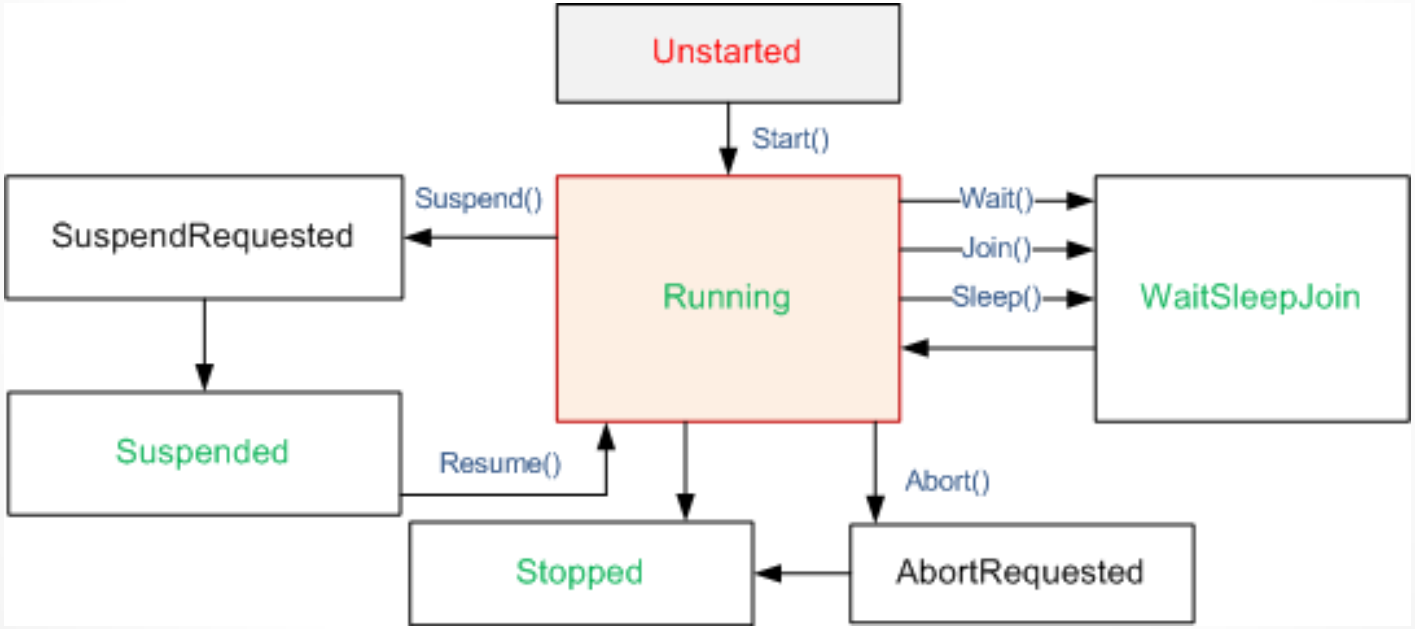
- Ein Thread läuft immer: in dem ist die **Anwendung**
- Beim Erstellen von einem neuen Thread: er läuft und wird anschließend beendet (wenn die Thread Methode abgeschlossen ist)
- Was passiert beim *Zugriff auf Programmoberfläche*?
- Wie kann man nun auf Steuerelemente zugreifen? – Wie werden die **delegates** sinnvoll verwendet?

Threads im Detail

- Methoden und Eigenschaften der Thread-Klasse
- Codeblöcke über *lock* sperren
- Weitere Möglichkeiten – ein mutex verwenden
- Mehrere Threads mit einem *ThreadPool* kontrollieren
- Das richtige Timing mit Timer-Threads.

Die Thread Zustände

- Viele Funktionen nur bei bestimmten Zuständen möglich – muss davor abgefragt werden



Beispiel

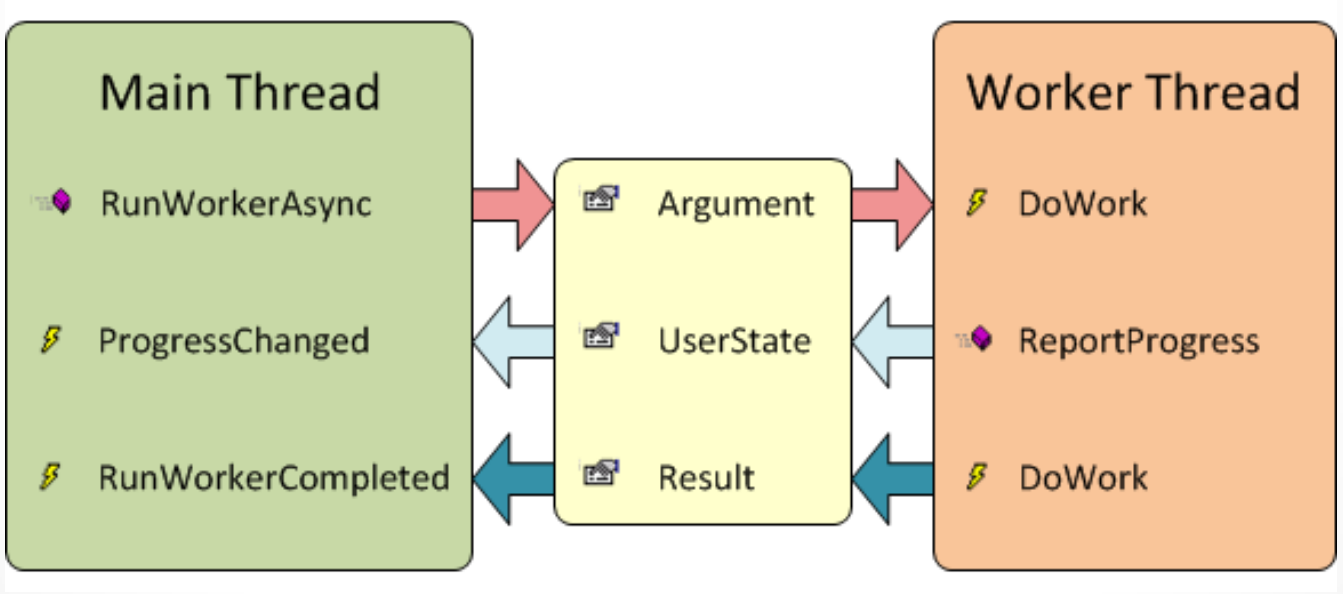
02 - Threads

Geht das einfacher?

- Die BackgroundWorker Komponente
- Ereignisse des Steuerelements
- **Fortschrittsanzeige** und **Abbruchmöglichkeit** einbauen
- Nachteile der BackgroundWorker Komponente

Background-Worker

- Die drei essentiellen Ereignisse verstehen und benutzen



Beispiel

03 – BackgroundWorker

Abschließendes Beispiel

- **Eigenes** Steuerelement, welches sich selbst zeichnet (analog Progressbar – nur schöner)
- Steuerelement soll gestartet werden können (beginnt großen Schreibprozess bis ***x*** MB)
- *Abbrechen* muss möglich sein
- *Logik* (d.h. IO Zugriff) von *Oberfläche* (GDI+) trennen über BackgroundWorker