

CheckBox

Namespace: `System.Windows.Forms`

Eigenschaften / Property's

- **Property - Standardwert**

Beschreibung oder Erläuterung der Eigenschaft

- **Appearance** - *Normal*

Darstellung der CheckBox

`Normal` = klassische Checkbox

`Button` = verhält sich wie ein Toggle-Button

- **AutoCheck** - *\$true*

Ob die CheckBox ihren Zustand automatisch selbst ändert

- **Checked** - *\$false*

Ob die CheckBox aktiviert ist

- **CheckState** - *Unchecked*

Zustand der CheckBox

(`Unchecked`, `Checked`, `Indeterminate`)

- **ThreeState** - *\$false*

Erlaubt dritten Zustand (`Indeterminate`)

- **Text** - *""*

Angezeigter Text neben der Checkbox

- **TextAlign** - *MiddleLeft*

Ausrichtung des Textes

- **CheckAlign** - *MiddleLeft*

Position des Häkchens

- **FlatStyle** - *Standard*

Darstellung der Checkbox

(`Standard`, `Flat`, `Popup`, `System`)

- **AutoSize** - *\$false*

Passt Größe automatisch an Inhalt an

- **Enabled** - *\$true*

Aktiviert oder deaktiviert die CheckBox

- **Visible** - *\$true*

Sichtbarkeit der CheckBox

- **Font** - *Standard-Systemfont*

Schriftart des Textes

- **ForeColor** - *Control/Text*
Textfarbe
- **BackColor** - *Transparent*
Hintergrundfarbe
- **Dock** - *None*
Docking innerhalb des Containers
- **Anchor** - *(Top, Left)*
Verhalten bei Größenänderung des Containers
- **Location** - *(0,0)*
Position innerhalb des Containers
- **Size** - *(104,24 ungefähr)*
Größe der CheckBox
- **TabIndex** - *0*
Reihenfolge beim Durchtabben
- **TabStop** - *\$true*
Ob die CheckBox per TAB erreichbar ist

Die **CheckBox** gehört zu diesen Controls, die harmlos aussehen... bis man plötzlich merkt, dass daran halbe UI-Logik hängt.

Denn technisch gesehen ist sie nicht einfach nur „an oder aus“.
Sie ist oft ein kleiner Schalter für:

- Einstellungen
- Features
- Berechtigungen
- Optionen
- Dynamisches UI-Verhalten

Und plötzlich hängt daran alles. Willkommen im Club menschlicher Selbstüberschätzung.

```
# CheckBox erstellen
$checkBox = New-Object System.Windows.Forms.CheckBox
$checkBoxNew = [System.Windows.Forms.CheckBox]::new()

# Text
$checkBox.Text = "Dark Mode aktivieren"

# Position & Größe
$checkBox.Location = New-Object System.Drawing.Point(10,10)
$checkBox.AutoSize = $true
```

```
# Standardmäßig aktiv
$checkbox.Checked = $true
```

Werte auslesen

```
# Aktiviert?
$state = $checkbox.Checked

# Exakter Zustand
$checkState = $checkbox.CheckState
```

- **Event - Hinweistext**

Auslöser / Trigger dieses Events

- **CheckedChanged**

Wird ausgelöst, sobald sich `Checked` ändert

- **CheckStateChanged**

Feuert bei Änderung von `CheckState`

- **Click**

Wird bei jedem Klick ausgelöst

- **DoubleClick**

Feuert beim Doppelklick

- **MouseDown**

Feuert vor `Click`

- **MouseUp**

Feuert nach `Click`

- **KeyDown**

Taste wird gedrückt

- **KeyUp**

Taste wird losgelassen

Events - *CheckBox*

☐ CheckedChanged

Das wichtigste Event der ganzen CheckBox.

Wird ausgelöst, sobald sich der Zustand ändert.

```
$checkBox.Add_CheckedChanged({  
    Write-Host "CheckBox Zustand:" $checkBox.Checked  
})
```

☐☐ Das ist normalerweise das Event, das du wirklich willst.

Nicht `Click`.

Nicht `MouseDown`.

Nicht irgendwelche kreativen Konstruktionen aus emotionalem Kontrollverlust.

☐☐ CheckStateChanged

Ähnlich wie `CheckedChanged`, aber für `CheckState`.

Relevant bei `ThreeState`.

```
$checkBox.ThreeState = $true  
  
$checkBox.Add_CheckStateChanged({  
    Write-Host "State:" $checkBox.CheckState  
})
```

☐☐ Ohne `ThreeState` bringt dir das meistens exakt gar nichts.

Click

Feuert bei jedem Klick auf die CheckBox.

```
$checkbox.Add_Click({  
    Write-Host "CheckBox wurde geklickt"  
})
```

Das Problem:

`Click` bedeutet nicht automatisch, dass sich der Zustand geändert hat.

Das vergessen Leute ständig und bauen dadurch doppelte Logik.

KeyDown

Für Tastatursteuerung.

```
$checkbox.Add_KeyDown({  
    Write-Host $_.KeyCode  
})
```

 Viele vergessen komplett, dass Benutzer auch Tastaturen besitzen. Faszinierende gesellschaftliche Entwicklung eigentlich.

ThreeState

Normalerweise kennt eine CheckBox nur:

```
Checked  
Unchecked
```

Mit `ThreeState` kommt hinzu:

```
Indeterminate
```

Beispiel:

```
$checkBox.ThreeState = $true  
$checkBox.CheckState = "Indeterminate"
```

Das nutzt man oft für:

- Teilweise ausgewählt
- Gemischte Zustände
- „Nicht eindeutig“

Klassisches Beispiel:

“ Ordner-Auswahl mit Unterelementen
Einige aktiviert → graues Kästchen

Appearance = Button

Das hier kennen überraschend viele nicht:

```
$checkBox.Appearance = "Button"
```

Dann wird aus der CheckBox ein Toggle-Button.

```
$checkBox.Text = "Musik aktivieren"  
$checkBox.Appearance = "Button"
```

 Sehr praktisch für moderne UI-Schalter.

Und ja, technisch bleibt es trotzdem einfach nur eine CheckBox im Kostüm. Menschen machen das übrigens auch ständig.

Typische Stolperfallen

- `CheckedChanged` feuert auch bei Änderungen per Code

```
$checkBox.Checked = $true
```

→ Event wird trotzdem ausgelöst

- `Click` und `CheckedChanged` gleichzeitig nutzen
→ doppelte Ausführung

- `ThreeState` aktiviert, aber nur `Checked` geprüft

```
if ($checkBox.Checked)
```

→ ignoriert `Indeterminate`

- `AutoCheck = $false`

Dann ändert die CheckBox ihren Zustand **nicht selbst**.

```
$checkBox.AutoCheck = $false
```

☐ Ab da bist *du* verantwortlich.

Glückwunsch. Du bist offiziell der Zustand-Manager deines kleinen Universums.

☐ Best Practice

- Für einfache Optionen → `CheckBox`
- Für Ein/Aus-Schalter → `Appearance = Button`
- Für mehrere zusammenhängende Optionen → `GroupBox`
- Für „eine von vielen“ → eher `RadioButton`

Ich greif einen Punkt raus, den viele komplett unterschätzen:

Eine CheckBox ist kein Datenspeicher. Sie ist nur UI.

Das hier:

```
if ($checkBox.Checked)
```

...ist kein „Systemzustand“.

Das ist nur die aktuelle Anzeige im Interface.

Wenn deine komplette Logik davon abhängt, ob irgendein Kästchen gerade angehakt ist, endet dein Projekt irgendwann wie ein Keller voller Verlängerungskabel. Funktioniert irgendwie. Bis jemand dagegen tritt.

Revision #1

Created 2026-05-15 23:24:57 UTC by John-Andreas Borinas

Updated 2026-08-23 19:25:50 UTC by John-Andreas Borinas