ListBox

Namespace: `System.Windows.Forms`

Properties / Eigenschaften

- ***Property*** – *Standardwert*
  Beschreibung oder Erläuterung der Eigenschaft

- **AllowDrop** – *$false*
  Erlaubt Drag & Drop auf die ListBox
- **Anchor** – *(Top, Left)*
  Bestimmt, wie sich die ListBox bei Größenänderung des Containers verhält
- **BackColor** – *SystemColors.Window*
  Hintergrundfarbe der ListBox
- **BorderStyle** – *Fixed3D*
  Rahmenstil (`None`, `FixedSingle`, `Fixed3D`)
- **Dock** – *None*
  Layout innerhalb des Parent-Containers (z.B. `Fill`)
- **DrawMode** – Normal
  Zeichenmodus (`Normal`, `OwnerDrawFixed`, `OwnerDrawVariable`)
- **Enabled** – $true
  Aktiviert oder deaktiviert die ListBox
- **Font** – *Standard-Systemfont*
  Schriftart der Einträge
- **ForeColor** – *SystemColors.WindowText*
  Textfarbe der Einträge
- **FormattingEnabled** – *$true*
  Aktiviert Formatierung für komplexe Objekte
- **HorizontalScrollbar** – *$false*
  Zeigt horizontale Scrollbar an
- **Location** – *(0,0)*
  Position innerhalb des Containers
- **Name** – ""
  Interner Name der ListBox
- **ScrollAlwaysVisible** – *$false*
  Scrollbar immer anzeigen, auch wenn nicht nötig
- **TabIndex** – *0*
  Reihenfolge beim Durchtabben

- **TabStop** – *$true*
  Ob die ListBox per Tab erreichbar ist
- **TopIndex** – *0*
  Index des obersten sichtbaren Elements
- **Visible** – *$true*
  Sichtbarkeit der ListBox

Items

- **Items** – *(leer)*
  Sammlung aller Listeneinträge
- **MultiColumn** – *$false*
  Mehrspaltige Darstellung aktivieren
- **SelectedIndex** – *-1*
  Index des aktuell ausgewählten Elements (`-1` = nichts)
- **SelectedItem** – *$null*
  Aktuell ausgewähltes Element
- **SelectedItems** – *(leer)*
  Collection aller ausgewählten Elemente (bei `Multi-Select`)
- **SelectionMode** – *One*
  Auswahlmodus
  - `One` → Nur ein Eintrag
  - `MultiSimple` → Mehrere ohne STRG
  - `MultiExtended` → Mehrere mit STRG/SHIFT
- **Sorted** – *$false*
  Sortiert Einträge automatisch alphabetisch

Größe

- **ColumnWidth** – *0*
  Breite der Spalten bei MultiColumn (`0` = automatisch)
- **Height** – *(abhängig vom Layout)*
  Höhe der ListBox
- **HorizontalExtent** – *0*
  Virtuelle Breite für horizontales Scrollen
- **IntegralHeight** – *$true*
  Passt Höhe automatisch an volle Einträge an (kein halbes Item unten)
- **ItemHeight** – *(abhängig von Font)*
  Höhe eines einzelnen Eintrags
- **Size** – *(Width=120, Height=96)*
  Größe der ListBox
- **Width** – *(abhängig vom Layout)*
  Breite der ListBox

Image Image

Die **ListBox** ist eines dieser Controls, die simpel wirken, aber erstaunlich schnell chaotisch werden, wenn man sie nicht im Griff hat. Im Kern zeigt sie eine Liste von Einträgen an, aus denen der Benutzer auswählen kann.

```
# ListBox erstellen
$listBox = New-Object System.Windows.Forms.ListBox
$listBoxNew = [System.Windows.Forms.ListBox]::new()

# Größe & Position
$listBox.Size = New-Object System.Drawing.Size(200,150)
$listBox.Location = New-Object System.Drawing.Point(10,10)

# Items hinzufügen
$listBox.Items.Add("Apfel")
$listBox.Items.Add("Banane")
$listBox.Items.Add("Kirsche")

# Mehrere auf einmal
$listBox.Items.AddRange(@("Orange","Mango","Traube"))
```

□□ Werte auslesen

```
# Einzelne Auswahl
$selected = $listBox.SelectedItem

# Index
$index = $listBox.SelectedIndex

# Mehrere auswählen
$selectedItems = $listBox.SelectedItems
```

Events

- ***Event*** *– Hinweistext*
  Auslöser / Trigger dieses Events

- **SelectedIndexChanged**
  Wird ausgelöst, sobald sich die Auswahl ändert.

- **SelectedValueChanged**
  Fast wie `SelectedIndexChanged`, aber subtil anders.
  Feuert, wenn sich der **Value** ändert (relevant bei `ValueMember`)

- **Click**
  Löst bei jedem Klick auf das Control aus
- **DoubleClick**
  Feuert, beim Doppelklick auf das Control / oder ein Item
- **MouseDown**
  Feuert **vor** dem Click
- **MouseUp**
  Feuert **nach** dem Click

- **KeyDown**
  Wird ausgelöst, wenn eine Taste gedrückt wird

- **KeyUp**
  Sowie `KeyDown`, aber erst wenn losgelassen wird

Events - *ListBox*

SelectedIndexChanged

Wird ausgelöst, sobald sich die Auswahl ändert.

```
$listBox.Add_SelectedIndexChanged({
    Write-Host "Ausgewählt:" $listBox.SelectedItem
})
```

SelectedValueChanged

Fast wie `SelectedIndexChanged`... aber subtil anders.
Feuert, wenn sich der **Value** ändert (relevant bei `ValueMember`).

```
$listBox.Add_SelectedValueChanged({
    Write-Host "Value geändert:" $listBox.SelectedItem
})
```

□□ Unterschied merkst du erst, wenn du mit Objekten arbeitest.

Click

Wird bei jedem Klick ausgelöst.
Ja, auch wenn sich **nichts ändert**. Klassiker für doppelte Logik.

```
$listBox.Add_Click({
    Write-Host "ListBox wurde geklickt"
})
```

DoubleClick

Wenn der User doppelt klickt. Perfekt für „öffnen“, „starten“, etc.

```
$listBox.Add_DoubleClick({
    Write-Host "Doppelklick auf:" $listBox.SelectedItem
})
```

□□ UX-technisch oft sinnvoller als Button daneben.

KeyDown

Für Tastatursteuerung. Wird ausgelöst, wenn eine Taste gedrückt wird.

```
$listBox.Add_KeyDown({
    if ($_.KeyCode -eq "Enter") {
        Write-Host "Enter auf:" $listBox.SelectedItem
    }
})
```

□□ Das ist der Moment, wo dein UI sich plötzlich „professionell“ anfühlt.

KeyUp

Wie KeyDown, nur nachdem losgelassen wurde.

```
$listBox.Add_KeyUp({
    Write-Host "Taste losgelassen:" $_.KeyCode
})
```

MouseDown

Feuert **vor** Click. Gut für spezielle Logik.

```
$listBox.Add_MouseDown({
    Write-Host "MouseDown erkannt"
})
```

MouseUp

Nach dem Klick.

```
$listBox.Add_MouseUp({
    Write-Host "MouseUp erkannt"
})
```

Tipps & Tricks - *TabControl*

Ich sag’s dir direkt, weil ich genau weiß, wie das läuft:

Du kombinierst sowas:

```
Click
SelectedIndexChanged
DoubleClick
```

...und wunderst dich, warum dein Code **mehrfach läuft**.

□□ Beispiel:

- Klick → `MouseDown`
- Klick → `Click`
- Auswahl ändert sich → `SelectedIndexChanged`

→ Boom, drei Events für einen simplen Klick.

□□ Mini-Leitfaden (der dir später Nerven spart)

- Auswahl reagieren → `SelectedIndexChanged`

- Aktion starten → `DoubleClick` oder `Enter`
- Nur Klick erkennen → `Click`
- Präzise Kontrolle → `MouseDown`

Items verwalten

```
# Entfernen
$listBox.Items.Remove("Apfel")

# Alles löschen
$listBox.Items.Clear()

# Einfügen an Position
$listBox.Items.Insert(0, "Neu")
```

Nützliche Tricks

Automatisch sortieren

```
$listBox.Sorted = $true
```

Mehrspaltig anzeigen

```
$listBox.MultiColumn = $true
```

Scrollbar erzwingen

```
$listBox.HorizontalScrollbar = $true
```

⚠ Typische Stolperfallen

- `SelectedItem` ist `$null`, wenn nichts gewählt ist → obvious, aber wird ständig vergessen
- `SelectedItems` ist **kein Array**, sondern Collection → verhält sich leicht anders
- Bei `MultiExtended`: Benutzer müssen STRG drücken → sonst denkt jeder, dein UI ist kaputt
- `Items.AddRange()` erwartet ein Array → kein wild zusammengebauter String-Müll

---

□□ Best Practice

- Für einfache Auswahl → `ListBox`
- Für strukturierte Daten → **ListView** (sonst wird’s hässlich)
- Für kleine Auswahl → lieber **ComboBox**

Ich greif einen Punkt raus, den du wahrscheinlich unterschätzt:

Was speicherst du eigentlich in der ListBox? Strings oder Objekte?

Wenn du nur Strings reinwirfst, verbaust du dir später jede sinnvolle Logik.
Pack lieber direkt Objekte rein:

```
$listBox.Items.Add([PSCustomObject]@{
    Name = "Chrome"
    Version = "123"
})
```

Und dann:

```
$listBox.DisplayMember = "Name"
```

Das ist der Unterschied zwischen „funktioniert irgendwie“ und „ich hab Kontrolle über meinen Code“.

Das ist so ein klassischer Punkt, wo Leute sich später selbst hassen, weil sie am Anfang “einfach schnell Strings genommen haben”.

Revision #10
Created 2026-03-31 12:41:05 UTC by John-Andreas Borinas
Updated 2026-04-07 04:06:43 UTC by John-Andreas Borinas