

Layout-Grundlagen: Docking, Anchoring, AutoSize, AutoScroll und Shell-Aufbau

Lektionsleitfaden · Lab- und Prüfungsleitfaden

● LearnWisej

Lektionsleitfaden

Die Standard-Layout-Engine, Dock-Reihenfolge und Padding, Anchoring für formularartige Bereiche, AutoSize und AutoScroll sicher eingesetzt und der Aufbau der Shell aus verschachtelten Containern und UserControls statt aus Resize-Handlern. Was das für Wisej.NET-Entwickler bedeutet – und wie Sie dieses Modul lesen.



Fokus: Produktivbetrieb

Die Standard-Layout-Engine, Dock-Reihenfolge und Padding, Anchoring für formularartige Bereiche, AutoSize und AutoScroll sicher eingesetzt und der Aufbau der Shell aus verschachtelten Containern und UserControls statt aus Resize-Handlern.

1. Das Problem im Produktivbetrieb

Die Adaptive Operations Console hat jetzt ein Theme und bereichsbezogenes CSS, doch ihre Shell verschiebt Controls noch von Hand: Ein **Resize**-Handler berechnet bei jeder Größenänderung des Browsers die **Bounds** für Toolbar, Navigationsleiste, Ticket-Grid und Details-Editor neu. Das funktioniert bei der Breite, die der Entwickler getestet hat, bricht bei jeder anderen und übersteht die Tablet- und Smartphone-Profile nicht, die Modul 6 einführt. Adaptives Design in Wisej.NET beruht auf Containern und Layout-Regeln, nicht auf Koordinaten: Bereiche, die Ränder beanspruchen, ein Arbeitsbereich, der den Rest füllt, und Editoren, die scrollen, statt sich zu überlappen.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
Standard-Layout-Engine	Jedes Wisej.NET-Control hat eine Layout-Engine; die Standard-Engine unterstützt absolute Positionierung, Docking und Anchoring, sodass ein Bildschirm präzise Platzierung im Designer mit Bereichen kombinieren kann, die ihre Größe selbst anpassen.
Docking-Priorität	Gedockte Children belegen Platz in der Reihenfolge, in der sie in der Control-Collection des Parents stehen; deshalb ergeben dieselben fünf Panels je nach Child-Reihenfolge eine korrekte oder eine überlappende Shell.
DisplayRectangle und Padding	Gedockte Controls werden innerhalb des <code>DisplayRectangle</code> des Parents angeordnet, das <code>Padding</code> verkleinert; die Standard-Engine ignoriert <code>Margin</code> bei gedockten Children, Abstände entstehen also durch Padding oder Platzhalter-Controls.
Anchoring	Ein Anchor hält den Abstand eines Controls zu ausgewählten Seiten des Parents konstant, was zu formularartigen Bereichen passt: Labels oben links, Textfelder links-rechts, Befehls-Buttons unten rechts.
AutoSize und zirkuläres Layout	Ein Container, der seine Größe aus Children ableitet, die wiederum an ihm gedockt oder verankert sind, hat keine stabile Antwort; diese Schleife ist die zentrale AutoSize-Warnung des Moduls.
AutoScroll mit ausgeblendeten Leisten	<code>AutoScroll</code> lässt einen Container scrollen, wenn der Inhalt nicht hineinpasst, und <code>ScrollBars.Hidden</code> erhält das Touch-Scrollen auf Smartphones, ohne Scrollbars zu zeichnen.
UserControl-Bereich	Ein Shell-Bereich mit eigenem lokalen Layout und kleiner öffentlicher Oberfläche, sodass die Shell den ganzen Bereich pro Profil anzeigt, ausblendet, dockt oder verschiebt, statt seine internen Koordinaten zu kennen.

3. Der rote Faden der Lektion

Docking, Child-Reihenfolge und der verbleibende Platz

Docking heftet ein Child an einen Rand seines Parents (`DockStyle.Top` , `Bottom` , `Left` , `Right`) oder lässt es den verbleibenden Platz füllen (`DockStyle.Fill`). Die Engine verteilt den Platz nacheinander an die gedockten Children, in ihrer Reihenfolge in der Collection des Parents, und jedes gedockte Child verkleinert das Rechteck, das dem nächsten bleibt. Das ist das ganze Geheimnis der Konsolen-Shell: Die Toolbar nimmt sich oben einen Streifen, der Statusbereich unten einen, die Navigationsleiste links einen, das Details-Panel rechts einen, und der Arbeitsbereich füllt den Rest. Ändern Sie die Reihenfolge, und das Status-Panel verdeckt die letzte Grid-Zeile. Im Visual-Studio-Designer ändern `BringToFront` , `SendToBack` und die Operationen für den Child-Index diese Reihenfolge; wenn eine Shell falsch aussieht, prüfen Sie also die Reihenfolge der Children, bevor Sie eine Größe anfassen. Der Shell-Code des Pakets ist bewusst kurz, und das Lab bildet ihn im Designer nach:

```
// MainPage.cs – die Konsolen-Shell, ein Dock pro Bereich
this.toolbarPanel.Dock = DockStyle.Top;    this.toolbarPanel.Height = 56;
this.statusPanel.Dock = DockStyle.Bottom; this.statusPanel.Height = 28;
this.navigationPanel.Dock = DockStyle.Left; this.navigationPanel.Width = 240;
this.detailsPanel.Dock = DockStyle.Right;  this.detailsPanel.Width = 360;

this.workspacePanel.Dock = DockStyle.Fill;
this.workspacePanel.MinimumSize = new System.Drawing.Size(320, 240);

// Abstände zwischen gedockten Bereichen kommen vom Container, nicht von Margin
this.Padding = new Padding(8);
this.workspacePanel.Padding = new Padding(12, 8, 12, 8);
```

Gedockte Children werden innerhalb des `DisplayRectangle` des Parents platziert, und `Padding` verkleinert dieses Rechteck von allen Seiten. Die Standard-Layout-Engine verwendet `Margin` bei gedockten Controls nicht: Ein Margin an der Navigationsleiste ändert nichts am Abstand daneben, geben Sie also dem Container `Padding` oder docken Sie ein schmales Platzhalter-Panel. Anchoring regelt die Controls innerhalb eines Bereichs. Ein Anchor bewahrt den Abstand zu den ausgewählten Seiten des Parents: Labels oben links, damit sie stehen bleiben, Textfelder links-rechts, damit sie mit dem Details-Panel wachsen, der breite Notizen-Editor oben-links-rechts und die Buttons Save und Cancel unten rechts. Ein Control ohne Anchor auf einer Achse schwebt und behält eine proportionale Position – richtig, um eine Anmeldekarte zu zentrieren, falsch für ein Grid.

Container statt Resize-Handler

Das `MainPage_Resize` der Konsole ist genau das Warnsignal, das dieses Modul beseitigen soll. Jede Zeile, die als Reaktion auf eine Größenänderung `Bounds`, `Location` oder `Size` setzt, ist eine Regel, die die Layout-Engine für Sie hätte anwenden können. Anders als die Engine läuft sie erst, nachdem der Browser das falsche Layout gezeichnet hat, weiß nichts über `Padding` und lässt sich nicht kombinieren: Sobald das Details-Panel ein UserControl wird, muss der Handler auch dessen Interna kennen. Das Refactoring ist mechanisch: die Bereiche auflisten, für jeden den beanspruchten Rand festlegen, sie in dieser Reihenfolge mit `Dock` hinzufügen und den Handler löschen. Absolute Koordinaten bleiben nützlich für präzise Designer-Arbeit in einem Dialog fester Größe; für einen Bildschirm, der auch auf einem Smartphone laufen muss, sind sie das falsche Werkzeug.

Eine dauerhafte Shell ist eine Menge verschachtelter Bereiche: Befehlsleiste oben, Navigation links, Arbeitsbereich in der Mitte, Details rechts, Status unten. Jeder Bereich ist ein `Panel`, `UserControl`, `TabControl`, `SplitContainer` oder eine `Page` mit eigenem lokalen Layout, und die Verschachtelung macht die Shell stabil: Den Arbeitsbereich kümmert nicht, wie die Toolbar ihre Buttons anordnet, und die Toolbar kümmert nicht, dass der Arbeitsbereich Kennzahlenkarten über einem Ticket-Grid enthält. Auch das Verankern vieler Controls in komplizierten Kombinationen ist ein Signal: Ein `TableLayoutPanel` oder `FlexLayoutPanel`, beide in Modul 5 behandelt, drückt ausgerichtete Felder oder proportionale Bereiche klarer aus als zwölf Anchor-Einstellungen.

AutoSize, AutoScroll und die Leitplanken für Größen

`AutoSize` ist wertvoll, wo der Inhalt die Größe bestimmt: ein Label, in das übersetzter Text passen muss, ein Badge, das mit seiner Zahl wächst, eine Karte, die dynamischen Inhalt umbricht; `AutoSizeMode`

entscheidet, ob das Control nur wachsen oder auch schrumpfen darf. Die Gefahr ist zirkuläres Layout. Leitet ein Container seine Größe aus Children ab, die an genau diesem Container gedockt oder verankert sind, fragt der Parent die Children, wie groß sie sein sollen, die Children fragen den Parent, wie groß sie sind, und die Engine findet keine stabile Antwort. Der typische Unfall ist ein Panel mit `AutoSize`, das zugleich seinen Parent füllt. Beschränken Sie `AutoSize` auf Blatt-Controls und auf Container, deren Children feste oder inhaltsabhängige Größen haben.

`AutoScroll` ist die sichere Antwort, wenn Inhalt nicht hineinpasst: Der Container scrollt, statt die Shell zu dehnen. Der Details-Editor der Konsole hat mehr Felder, als ein Smartphone-Bildschirm zeigen kann, daher schaltet das Lab `AutoScroll` ein und setzt bei schmalen Breiten `ScrollBars` auf `Hidden`; so bleibt das Touch-Scrollen erhalten, ohne Scrollbars zu zeichnen, die Breite kosten. `AutoScrollMargin` schafft Raum nach dem letzten Control, und `AutoScrollMinSize` garantiert einen minimalen scrollbaren Bereich.

`MinimumSize` und `MaximumSize` sind die anderen Leitplanken: Eine `MinimumSize` am Arbeitsbereich und am Grid bedeutet, dass der füllende Bereich nie zu einem schmalen Streifen zusammenfällt, wenn der Browser schmaler wird, und eine `MaximumSize` verhindert, dass das Details-Panel einen Ultra-Wide-Monitor verschluckt.

UserControl-Bereiche, die die Shell neu anordnen kann

Der letzte Schritt des Labs zahlt sich in Modul 6 aus. Sobald Navigationsleiste und Details-Editor `UserControl`s sind, jedes mit eigenem internen Docking und Anchoring, kennt die Shell die internen Koordinaten keines Editors mehr. Sie kennt fünf Bereiche und zeigt, verbirgt, dockt oder verschiebt pro Profil ganze Bereiche: Die Leiste kann zu einem schmalen Symbolstreifen werden, der Details-Editor kann auf dem Tablet unter das Grid wandern oder auf dem Smartphone in einer modalen `Form` öffnen, und keine dieser Verschiebungen berührt ein Textfeld. Der Bereich stellt wenige aussagekräftige Member bereit, etwa eine Eigenschaft `Ticket` und ein Event `Saved`, und hält seine Child-Controls privat, damit eine Umbenennung im Inneren die Seite nie bricht.

Die Layout-Checkliste des Moduls ist kurz: Container statt Resize-Handler bevorzugen, die Dock-Reihenfolge prüfen, `Padding` an gedockte Bereiche setzen, `MinimumSize` an füllende Bereiche setzen und nicht passenden Inhalt scrollen lassen. Die Liste der Refactoring-Warnsignale des Pakets ist das Spiegelbild dieser Checkliste, und einen Bildschirm vor jedem Profilttest daran zu prüfen, ist eine gute Gewohnheit.



Refactoring-Warnsignale

Manueller `Resize`-Code, der wiederholt `Bounds` an vielen Controls setzt. `Dock` und `Anchor` gemischt an Controls, die sich gegenseitig bekämpfen. Absolute Koordinaten in einem Bildschirm, der auf Smartphones laufen muss. Grids ohne `MinimumSize`. Panels mit `AutoSize` und `Fill`-Docking, die Schleifen erzeugen. Formular-Layouts, die Controls ausblenden, statt nachrangige Inhalte in einen scrollbaren Abschnitt zu verschieben.



Gewohnheit für das visuelle System

Nutzen Sie den Wisej.NET-Mechanismus, der für das Problem zuständig ist: das Theme für die Identität der Standard-Controls, AppearanceKey für semantische Varianten, CssClass für bereichsbezogenes anwendungsspezifisches CSS, CssStyle nur für einmalige dynamische Werte, Layout-Container statt Resize-Code, responsive Eigenschaften für im Designer gesetzte Profilwerte und ResponsiveProfileChanged für Verhalten, das nicht rein visuell ist.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ Docking und Anchoring bewusst einsetzen und verstehen, wie die Child-Reihenfolge die Docking-Priorität beeinflusst.
- ✓ Bildschirme aus verschachtelten Containern statt aus Resize-Handlern und manuellen Koordinatenänderungen aufbauen.
- ✓ AutoSize, AutoSizeMode, AutoScroll, ScrollBars, MinimumSize, MaximumSize, Padding und Margin sicher verwenden.
- ✓ wiederverwendbare UserControl-Bereiche erstellen, die sich über Desktop-, Tablet- und Smartphone-Profile hinweg neu anordnen lassen.

Lab- und Prüfungsleitfaden

Was Sie im praktischen Lab bauen, das vorgeschlagene Vorgehen und die geforderten Ergebnisse.



Praktisches Lab

Bauen Sie die Shell der Adaptive Operations Console per Refactoring in verschachtelte Container um: eine Toolbar oben, eine Navigationsleiste links, ein füllender Arbeitsbereich, ein Details-Panel rechts und ein Statusbereich, jeweils ein Panel oder UserControl mit eigenem lokalen Layout. Löschen Sie jeden Resize-Handler, der Bounds setzt, korrigieren Sie die Dock-Reihenfolge im Designer, verwenden Sie Padding für Abstände zwischen gedockten Bereichen, geben Sie Arbeitsbereich und Grid eine MinimumSize und lassen Sie den Details-Editor bei schmalen Breiten mit AutoScroll und ausgeblendeten Scrollbars scrollen.

Ausgangspunkt

Öffnen Sie die Solution `AdaptiveOperationsConsole` so, wie Sie sie nach Modul 3 verlassen haben: Die Haupt-`Page` trägt das AdaptiveOps-Theme, das bereichsbezogene Stylesheet und den Theme-Umschalter pro Session, doch ihre fünf Bereiche sind noch einfache Panels, deren `Bounds` in `MainPage_Resize` neu berechnet werden, und unter etwa 1100 Pixeln überlappt der Details-Editor das Ticket-Grid. Starten Sie sie einmal in Desktop-Breite, ziehen Sie dann den Browser schmäler und notieren Sie, was abgeschnitten wird, sich überlappt oder verschwindet, bevor Sie etwas ändern.

Vorgeschlagenes Vorgehen

- ✓ Listen Sie jede `Resize` -, `Layout` - oder Ladezeit-Methode auf, die `Bounds` , `Location` oder `Size` setzt, schreiben Sie die fünf Bereiche und den Rand, den jeder beansprucht, in Ihre Lab-Notizen und löschen Sie diesen Code, bevor Sie den Designer anfassen.
- ✓ Setzen Sie im Designer `Dock` an den fünf Panels (Toolbar Top, Status Bottom, Navigation Left, Details Right, Arbeitsbereich Fill) und korrigieren Sie die Child-Reihenfolge mit `BringToFront` und `SendToBack`, bis die Ränder zuerst beansprucht werden und der Arbeitsbereich den Rest erhält.
- ✓ Geben Sie der Seite und den Bereichen `Padding` für die Abstände zwischen gedockten Flächen, setzen Sie `MinimumSize` am Arbeitsbereich und am Ticket-Grid und überzeugen Sie sich, dass die Standard-Engine `Margin` bei gedockten Children ignoriert.
- ✓ Verankern Sie im Details-Panel Labels oben links, Editoren links-rechts, den Notizen-Editor oben-links-rechts und die Befehls-Buttons unten rechts; schalten Sie `AutoScroll` ein, setzen Sie `ScrollBars` für schmale Breiten auf `Hidden` und justieren Sie `AutoScrollMargin` .
- ✓ Lagern Sie `NavigationRail` und `DetailsEditor` als `UserControl` s mit eigenem lokalen Layout und kleiner öffentlicher Oberfläche aus, setzen Sie sie wieder in die Shell ein und verkleinern Sie den Browser von Desktop- auf Smartphone-Breite, um zu prüfen, dass sich nichts überlappt oder abgeschnitten wird.

Geforderte Ergebnisse

- ✓ Shell aus verschachtelten Containern mit Toolbar-, Navigations-, Arbeitsbereichs-, Details- und Statusbereich neu aufgebaut

- ✓ Sämtlicher manueller Resize-Code, der Bounds setzte, entfernt
- ✓ Dock-Reihenfolge, Padding und MinimumSize in jedem Bereich bewusst gesetzt
- ✓ Details-Editor scrollt bei schmalen Breiten über AutoScroll mit ausgeblendeten Scrollbars
- ✓ Navigations- und Detailbereich in UserControls ausgelagert

Wiederholungsfragen

- ✓ Wäre das Arbeitsbereichs-Panel in der Child-Reihenfolge vor dem Status-Panel gedockt: Was sähe der Benutzer am unteren Rand des Grids, und welcher Designer-Befehl behebt das, ohne eine einzige Größe zu ändern?
- ✓ Sie haben an der Navigationsleiste einen `Margin` von 12 Pixeln gesetzt, und der Abstand ist nicht erschienen. Warum nicht, und welche zwei Mechanismen erzeugen tatsächlich einen Abstand zwischen gedockten Bereichen?
- ✓ Der Details-Editor ist ein `UserControl` mit `AutoScroll`. Welche seiner Member muss die Shell kennen, um ihn auf dem Tablet unter das Grid zu verschieben, und welche darf sie nie benötigen?



Gewohnheit für das visuelle System

Nutzen Sie den Wisej.NET-Mechanismus, der für das Problem zuständig ist: das Theme für die Identität der Standard-Controls, AppearanceKey für semantische Varianten, CssClass für bereichsbezogenes anwendungsspezifisches CSS, CssStyle nur für einmalige dynamische Werte, Layout-Container statt Resize-Code, responsive Eigenschaften für im Designer gesetzte Profilwerte und ResponsiveProfileChanged für Verhalten, das nicht rein visuell ist.