Mi az a dependency property?

A property rendszer tovább gondolása, kiegészítése. Mindenképp az alapvető WPF funkcionalitások közé kell sorolni, ugyanúgy, mint az animációt, adatkötést és a stílusokat. WPF specifikus eszköz. Létrehozásuk célja egyértelműen az automatikus változásértesítés támogatása, de ezen kívül számos más funkcionalitással rendelkeznek (default érték, trigger…).

Két fő osztály: DependencyObject (az alaposztály) és a DependencyProperty (a property leírója).

Két lépésben hozzuk létre. Először a reprezentáló DependencyProperty objektumot regisztráljuk, utána pedig a property-t magát.

```
public static readonly DependencyProperty BackgroundProperty =
DependencyProperty.Register(„Background”, typeof( Brush ),
typeof( Control ), _ FrameworkPropertyMetadataOptions.AffectsRender,
_Brushes.White, _ validationCallback ) );

public Brush Background
{ get { return (Brush)this.GetValue( BackgroundProperty ); } }
```

Mi az attached property?

Általában olyan, egy adott objektumhoz tartozó beállítás, amit másik objektumok használnak.
Például:

- Szülő: *DockPanel* osztály
- Információ: melyik oldalra dokkoljon
- AttachedProperty: *DockProperty*, Dock.Left, stb.
- Bármelyik gyerek elemen be lehet állítani
- Előnye: Általános, kiterjeszthető

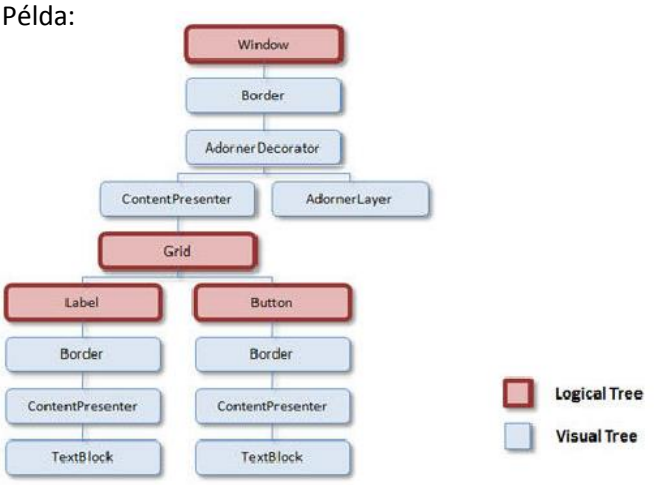
```
<DockPanel>
<Button DockPanel.Dock=„Left" />

DockProperty = DependencyProperty.RegisterAttached( _ );
DockPanel.SetDock( uiElement, Dock.Left );
uiElement.SetValue( DockPanel.DockProperty, Dock.Left );
```

Mi a különbség a logical tree es a visual tree között?

A *Logical tree* a vezérlők hierarchiája, alapvetően a XAML definiálja, ez a tartalom modell.
A *Visual tree* a megjelenítésért felelős, Visual-ok fája, ahol a Visual nem más, mint a megjelenés vektorgrafikus összeállításáért felelős osztály. Példa:

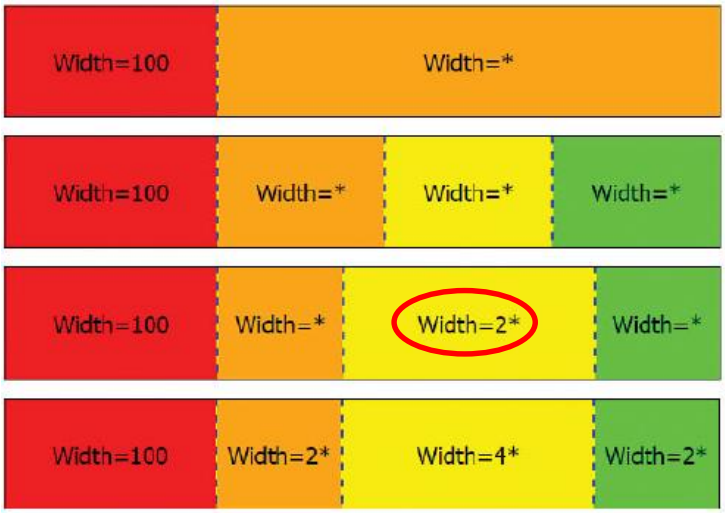
```
<Window>
<Grid>
<Label Content="Label" />
<Button Content="Button" />
</Grid>
</Window>
```



Milyen elrendezést definiál az a grid, ami két sort definiál a következő Height propertykkel: Height="*" illetve Height="2*"?

A méretezés kicsit trükkös, lehet arányokat is megadni. A 2* például azt jelenti, hogy a fixen lefoglalt méretek mellett fennmaradó hely, n-felé osztásakor 2 egységnyit kap meg a fennmaradó helyből. Itt van pár példa, talán így világosabb.

- Cellák méretezése (GridLength):
- Abszolút méretezés: Height="100"
- Auto méretezés: Height="Auto"
- Star méretezés: Height="2*"



Mire szolgál a binding osztály?

Adatkötés célja: egy vezérlő tulajdonságainak hozzákötése az adatforráshoz. A Binding az adatkötés egyik alaposztálya, az adatkötés jellemzőit írja le. Például:

- *Source* : forrás objektum, tetszőleges típus
- *Path* : forrás property, elmaradhat

Milyen értékeket tartalmazhat a Binding Mode property-je?

Az adatkötés módjat alapértelmezetten a cél-property adja meg vagy explicit beállítható a ***Mode property*** értékével.
Pl.: TextBox.Text, stb kétirányú, a többi tipikusan egyirányú

Lehetséges értékei:

TwoWay: kétirányú kötés, a forrás- és cél property változása frissíti a másikat
OneWay: a cél property frissül, amikor a forrás megváltozik
OneTime: csak az alkalmazás indulásakor vagy a DataContext változásakor kerül frissítésre
OneWayToSource: a forrás property frissül, a mikor a cél property megváltozik
Default: alapértelmezett, a cél propertytől függ

Milyen interface-t kell megvalósítania egy converter-nek?

Kötött adatok konvertálása az adatcsere előtt lehetséges a Binding.Converter beállításával.

A *Converter* nem más, mint egy, az ***IValueConverter*** interfészt megvalósító osztály. A ValueConversionAttribute attribútummal a konverter osztály külön megjelölhető.

A konvertáláshoz a konverter bemenő argumentumot, paramétert és még kultúra információt is kap. A paraméter a kötésnél adható meg, értékei lehetnek:

- Convert: az egyirányú adatkötéshez
- ConvertBack: a kétirányú adatkötéshez

Mi a különbség egy List<T> és egy ObservableCollection<T> típusú property között változásértesítés szempontjából?

A változásértesítési mechanizmus WPF alatt is a standard ***INotifyProperty*** interfész megvalósításán alapul (Ez biztosítja az ***INotifypropertyChanged*** event elsütését változások esetén.) Ez skalár adatoknál tökéletes, viszont kollekcióknál problémás lehet. Hol változott meg? Hogyan? Például egy ***List<T>*** megváltozása bejárás közben exception-t dobna. Ezen kívül egy 1000000 elemű lista újbóli bejárása egy elem megváltozása miatt elég erőforrás pazarló is volna.

Ennek megoldására vezették be az ***ObservableCollection<T>*** típust és az ***INotifyCollectionCahnged*** interfészt és eseményt, amely jelzi, ha a kollekciónak egy adott eleme megváltozott (felülírták, törölték, mozgatták, vagy újat szúrtak be…).

Property változások jelzése

- INotifyPropertyChanged interfész
- PropertyChanged event minta
    - A szöveggel hivatkozott tulajdonság megváltozott

Collection megváltozása

- INotifyCollectionChanged interfész
    - Új elem, elem mozgatása/törlése/változása, reset
- ObseverableCollection<T> : INotifyCollectionChanged