C#12, net 8.0 LTS, winforms.

Het blijkt dat bv een combobox niet out of the box de IDataGridViewEditingControl met bijbehorende cell en Column ondersteunt.

Kennelijk doet een TextBox dat wel, want dat is het standaard edit control van een DataGridView cell?

Is het 't typen van enter dat de validatie start en het beeindigen van het editen? (CellValidating en CellEndEdit)?

Deepseek:  
  
**1. Standaard gedrag van controls in DataGridView**

* **TextBox**: Ondersteunt inderdaad standaard editing (basisimplementatie van IDataGridViewEditingControl)
* **ComboBox**: Vereist extra implementatie omdat het complexer gedrag heeft:

csharp

public class DataGridViewComboBoxEditingControl : ComboBox, IDataGridViewEditingControl

{

// Vereiste implementatie...

}

**2. Enter-toets en validatieflow**

De complete edit-cycle ziet er zo uit:

1. **Start editing**:
   * Gebruiker dubbelklikt of drukt op F2/Enter
   * EditingControlShowing event treedt op
2. **During editing**:
   * Voor TextBox: Directe tekstwijzigingen
   * Voor ComboBox: Selectie wijzigingen
3. **End editing** (triggered door):
   * **Enter**: Bevestigt de wijziging → triggers validatie
   * **Escape**: Annuleert → behoudt originele waarde
   * **Klik buiten de cel**: Bevestigt (mits validatie slaagt)
4. **Validatievolgorde**: CellValidating → CellValidated → CellEndEdit

**3. Essentiële implementatie voor custom controls**

Voor je CheckedComboBox moet je deze interface correct implementeren:

csharp

Copy

public class CheckedComboBox : ComboBox, IDataGridViewEditingControl

{

// Vereiste properties

public DataGridView EditingControlDataGridView { get; set; }

public int EditingControlRowIndex { get; set; }

public bool EditingControlValueChanged { get; set; }

// Belangrijkste methodes

public object GetEditingControlFormattedValue(DataGridViewDataErrorContexts context)

{

return string.Join(",", this.GetCheckedValues());

}

public void PrepareEditingControlForEdit(bool selectAll)

{

// Optionele voorbereiding

}

public bool EditingControlWantsInputKey(Keys keyData, bool dataGridViewWantsInputKey)

{

// Laat speciale keys door (bv. pijltjes voor dropdown)

return keyData switch {

Keys.Up or Keys.Down or Keys.Enter => true,

\_ => false

};

}

}

**4. Keyboard handling die de flow beïnvloedt**

| **Toets** | **Gedrag** | **Triggers Validatie** |
| --- | --- | --- |
| Enter | Beëindigt edit | Ja |
| Escape | Annuleert edit | Nee |
| Tab | Volgende cel | Ja |
| F2 | Start edit | N/A |

**5. Praktijkvoorbeeld voor CheckedComboBox**

csharp

Copy

protected override void OnKeyDown(KeyEventArgs e)

{

base.OnKeyDown(e);

if (e.KeyCode == Keys.Enter)

{

// Forceer validatie

this.EditingControlValueChanged = true;

this.EditingControlDataGridView.EndEdit();

}

}

**Belangrijkste verschillen tussen controls:**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Conclusie:**

1. Alleen TextBox heeft volledige out-of-the-box ondersteuning
2. Voor andere controls moet je:
   * IDataGridViewEditingControl implementeren
   * Keyboard handling toevoegen
   * Explicitiete validatie afhandelen
3. De Enter-toets is inderdaad de primaire trigger, maar de exacte werking hangt af van je implementatie van PrepareEditingControlForEdit en EditingControlWantsInputKey

Wil je een volledig werkend voorbeeld van een CheckedComboBox met alle benodigde interface-implementaties zien? Ik kan je een complete implementatie sturen die naadloos met de DataGridView samenwerkt.  
  
  
  
===  
Gedrag combobox:  
  
Ja! In een standaard **WinForms ComboBox**, als je this.SelectedIndex = 0; zet, dan gebeurt het volgende automatisch:

**✅ Wat wordt aangepast?**

| **Property** | **Wat gebeurt er bij SelectedIndex = 0?** |
| --- | --- |
| SelectedItem | Wordt het eerste item in de Items-lijst |
| SelectedValue | Wordt afgeleid (alleen als ValueMember is gezet) |
| Text | **Wordt gezet op de stringrepresentatie van dat item** ✔️ |

**🔍 Voorbeeld**

comboBox1.Items.Add("Apple");

comboBox1.Items.Add("Banana");

comboBox1.SelectedIndex = 0;

Console.WriteLine(comboBox1.Text); // "Apple"

**⚠️ Let op met DropDownStyle**

| **DropDownStyle waarde** | **Gedrag van Text** |
| --- | --- |
| DropDown (standaard) | Text kan door gebruiker worden aangepast |
| DropDownList | Text is altijd gesynchroniseerd met SelectedItem |
| Simple | Gedraagt zich als een gecombineerde lijst en tekstbox |

Als je dus DropDownList gebruikt, is Text **readonly voor de gebruiker**, maar nog steeds automatisch gesynchroniseerd met SelectedIndex.

**🧪 Bonus: als item een object is**

Als je een lijst van objecten gebruikt:

comboBox1.DisplayMember = "Name";

comboBox1.ValueMember = "Id";

comboBox1.DataSource = listOfObjects;

comboBox1.SelectedIndex = 0;

Dan wordt Text = object.ToString() **als DisplayMember niet goed werkt**, of anders de waarde van DisplayMember.

Laat maar weten of je een ComboBox gebruikt met custom objecten — dan toon ik wat er precies in Text, SelectedItem en SelectedValue zit.

**Vraag 1: In InitializeEditingControl (DataGridViewComboBoxCell), is object initialFormattedValue altijd gelijk aan FormattedValue? (of type public override Type FormattedValueType , usually typeof(string))?**

Ja, in de meeste gevallen is de initialFormattedValue die wordt doorgegeven aan de InitializeEditingControl-methode van een DataGridViewComboBoxCell **gelijk aan de FormattedValue van de cel op het moment dat de bewerking begint. (GetFormattedValue, EditingControlFormattedValue)**

* **FormattedValue:** Dit is de waarde van de cel zoals deze aan de gebruiker wordt getoond voordat de bewerking start.
* **initialFormattedValue:** Dit is de waarde die in de bewerkingscontrole (de ComboBox) wordt geplaatst wanneer deze voor het eerst wordt geopend om de cel te bewerken. Het is logisch dat dit de waarde is die de gebruiker zag voordat hij begon met bewerken.

Wat betreft FormattedValueType:

* Voor een DataGridViewComboBoxCell is het **gebruikelijk dat de FormattedValueType typeof(string) is.** Dit komt omdat de tekst die in de cel en in de dropdown wordt weergegeven, meestal een string is.
* Het is echter **niet strikt noodzakelijk** dat het altijd typeof(string) is. Als je een aangepaste formattering hebt geïmplementeerd (bijvoorbeeld door de Format gebeurtenis van de DataGridView te behandelen of door een afgeleide celklasse te maken), kan de FormattedValue een ander type zijn. Maar voor de standaard DataGridViewComboBoxCell is string de meest voorkomende en logische keuze voor de weergave.

**Vraag 2: Gebruikt de display cell object.ToString() voor display value als object niet een string is?**

**Ja, standaard gebruikt de DataGridView de ToString()-methode van het object om de weergavewaarde van een cel te bepalen als de Value van de cel geen string is.**

Dit geldt ook voor de DataGridViewComboBoxCell. Als de Value van de cel (het geselecteerde item) een object is en de cel geen specifieke manier heeft om te bepalen hoe dit object moet worden weergegeven, dan zal de ToString()-methode van dat object worden aangeroepen om een stringrepresentatie te krijgen voor de weergave in de cel.

**Belangrijke uitzonderingen en overwegingen voor DataGridViewComboBoxCell:**

* **DisplayMember:** De DataGridViewComboBoxCell heeft een eigenschap genaamd DisplayMember. Als je de DataSource van de combobox instelt op een lijst van objecten, kun je de DisplayMember instellen op de naam van de eigenschap van die objecten die je wilt weergeven in de dropdown en in de cel. Als DisplayMember is ingesteld, wordt de waarde van die specifieke eigenschap gebruikt voor de weergave in plaats van ToString() van het hele object.
* **Formattering:** Zoals eerder genoemd, kun je formattering toepassen op de DataGridView om te bepalen hoe waarden worden weergegeven, inclusief objecten.
* **Cel-specifieke formattering:** Je kunt ook de FormattedValue property van de cel direct bepalen in een afgeleide celklasse.

**Samenvattend:**

* initialFormattedValue is meestal gelijk aan FormattedValue in InitializeEditingControl.
* FormattedValueType is doorgaans typeof(string) voor een DataGridViewComboBoxCell.
* De DataGridView gebruikt standaard object.ToString() om de weergavewaarde te bepalen als de celwaarde geen string is, tenzij er mechanismen zoals DisplayMember of specifieke formattering zijn ingesteld.  
    
    
  Here's a breakdown of why this might seem like a flaw and how it's typically handled:
* **ValueType:** This property correctly indicates the underlying data type of the cell's value. This is important for data storage and manipulation.
* **GetFormattedValue:** This method is responsible for converting the cell's value (of type T) into a format suitable for display in the DataGridView. For a ComboBoxCell, this formatted value is usually a string that the user sees in the cell before editing.
* **InitializeEditingControl:** This method prepares the editing control (the ComboBox) for editing. It receives initialFormattedValue, which is the string representation of the cell's value. The ComboBox within the editing control needs to select the correct item based on this formatted value.

Stel dat je een DataGridViewComboBoxCell hebt waarin een object van het type Klant wordt opgeslagen. De ValueType van de cel is typeof(Klant). De DisplayMember van de ComboBox is ingesteld op de eigenschap Naam van de Klant-klasse.

Wanneer de cel niet in de bewerkingsmodus is, zal de DataGridView de GetFormattedValue-methode van de cel aanroepen. Deze methode zal waarschijnlijk de Naam-eigenschap van het Klant-object ophalen en deze als een string retourneren voor de weergave in de cel.

Wanneer de gebruiker de cel begint te bewerken, wordt InitializeEditingControl aangeroepen. De initialFormattedValue parameter zal dan de string zijn die de naam van de klant bevat (het resultaat van de GetFormattedValue-aanroep). De implementatie van de DataGridViewComboBoxCell gebruikt deze string vervolgens om de juiste klant in de dropdown te selecteren.

De DataGridViewComboBoxCell probeert dit probleem vaak te ondervangen door te werken met de DataSource, DisplayMember en ValueMember van de ComboBox. Wanneer de bewerking start, zoekt de cel naar het item in de DataSource waarvan de DisplayMember-waarde overeenkomt met de initialFormattedValue. Als er echter meerdere items zijn met dezelfde DisplayMember-waarde, is er geen garantie dat de juiste wordt geselecteerd.

Het gebruik van een unieke ValueMember kan hier helpen, maar zelfs dan moet de initialFormattedValue overeenkomen met de weergave van het item waarvan de ValueMember de daadwerkelijke celwaarde is.

De IDataGridViewEditingControl interface definieert de leden die een custom bewerkingscontrole moet implementeren om te kunnen worden gebruikt in een DataGridView. De EditingControlFormattedValue-property (en de getter ervan) en de DataGridViewComboBoxCell.GetFormattedValue-methode hebben beide betrekking op de geformatteerde waarde, maar in verschillende contexten:

**IDataGridViewEditingControl.EditingControlFormattedValue (getter):**

* Dit is een property van de interface IDataGridViewEditingControl.
* De **getter** van deze property wordt gebruikt door de DataGridView om de **huidige geformatteerde waarde op te vragen van de edit control** (de ComboBox in het geval van een DataGridViewComboBoxCell) **wanneer de cel in de bewerkingsmodus is.**
* Het vertegenwoordigt de waarde die de gebruiker momenteel ziet en/of heeft geselecteerd in de bewerkingscontrole.
* Wanneer de gebruiker de bewerking voltooit (bijvoorbeeld door op Enter te drukken of naar een andere cel te gaan), zal de DataGridView deze EditingControlFormattedValue waarschijnlijk gebruiken om de onderliggende Value van de cel bij te werken (via de SetEditingControlCellValue methode van de cel).

**DataGridViewComboBoxCell.GetFormattedValue:**

* Dit is een methode van de klasse DataGridViewComboBoxCell (en is overgeërfd van DataGridViewCell).
* Deze methode wordt gebruikt door de DataGridView om de **geformatteerde waarde van de cel op te vragen voor weergave wanneer de cel *niet* in de bewerkingsmodus is.**
* Deze methode neemt de onderliggende Value van de cel (van het type dat is gedefinieerd door ValueType) en converteert deze naar een object dat geschikt is voor weergave (meestal een string).
* De logica in GetFormattedValue bepaalt hoe de data in de cel wordt weergegeven wanneer deze niet wordt bewerkt. Dit kan bijvoorbeeld het weergeven van de DisplayMember van het geselecteerde object zijn.

**De relatie tussen de twee:**

* **GetFormattedValue** is relevant voor de **weergave van de cel wanneer deze niet wordt bewerkt.** Het bepaalt wat de gebruiker ziet voordat hij de cel gaat bewerken.
* **EditingControlFormattedValue** is relevant voor de **waarde binnen de bewerkingscontrole (de ComboBox) wanneer de cel wordt bewerkt.** Het vertegenwoordigt de waarde waarmee de gebruiker momenteel interageert.

**In essentie:**

* GetFormattedValue zorgt voor de **initiële weergave** van de celwaarde.
* EditingControlFormattedValue geeft de **huidige waarde** weer die de gebruiker in de bewerkingscontrole heeft geselecteerd of ingevoerd.

Wanneer een DataGridViewComboBoxCell in de bewerkingsmodus gaat:

1. De DataGridView roept GetFormattedValue aan om de initiële waarde voor de bewerkingscontrole te bepalen (deze waarde wordt doorgegeven als initialFormattedValue aan InitializeEditingControl).
2. De DataGridView maakt de edit control (een ComboBox) zichtbaar en stelt de initiële waarde in (vaak door de initialFormattedValue te gebruiken om de juiste optie in de ComboBox te selecteren).
3. Terwijl de gebruiker de ComboBox bedient, houdt de DataGridView de huidige geselecteerde waarde bij via de getter van de EditingControlFormattedValue-property van de IDataGridViewEditingControl-interface die door de ComboBox (of een wrapper eromheen) wordt geïmplementeerd.
4. Wanneer de gebruiker de bewerking voltooit, kan de DataGridView deze EditingControlFormattedValue gebruiken om de onderliggende Value van de DataGridViewComboBoxCell te updaten.

Het is dus een proces in twee stappen: eerst de weergave van de cel (via GetFormattedValue) en vervolgens de interactie met en de waarde van de bewerkingscontrole (via EditingControlFormattedValue).