DE L’USAGE DES GRILLES DE LAZARUS

Traduction de “USING THE LAZARUS GRIDS”

Michaël Van Canneyt

August 4, 2008

Le contrôle de la grille dans Lazarus est dans une très large mesure compatible avec son cousin de Delphi, de sorte que le programmeur expérimenté n'aura pas trop de difficultés à l'utiliser. Néanmoins, il y a quelques ajouts intéressants et quelques différences marquées par rapport à la grille de Delphi. Cet article les explore et montre comment la grille peut être étendue.

1 INTRODUCTION

Lorsqu'une application doit présenter une vue d'ensemble des données, le programmeur peut choisir entre une TListView, une TTreeView, ou une TCustomGrid, et son équivalent orienté données TDBGrid. Cet article examine l'implémentation de la grille dans Lazarus, le lecteur est supposé être à peu près familier avec l'implémentation du contrôle Delphi TStringGrid, donc seules les différences seront expliquées.

Tout comme dans Delphi, il existe 3 types de grilles dans Lazarus :

* TDrawGrid : Ici, toutes les cellules de la grille doivent être définies par le programmeur. Le composant n'a pas de stockage pour les données associées aux cellules.
* TStringGrid : Cette grille a la capacité d'associer une chaîne de caractères à chaque cellule de la grille, et elle sait comment présenter les chaînes de caractères. et sait comment présenter les chaînes de caractères.
* TDBGrid est une grille spéciale qui prend en compte les données. Elle sait comment afficher les données d'un descendant de TDataset.

Les trois descendent de TCustomGrid, et la plupart des fonctionnalités sont implémentées dans cet ancêtre commun.

2 POSSIBILITES SUPPLEMENTAIRES

L'implémentation de Stringgrid de Lazarus a quelques possibilités supplémentaires qui sont exposées dans des propriétés supplémentaires :

* **AlternateColor** : Si cette propriété de couleur est définie à une couleur différente de la propriété color, la grille sera affichée avec des couleurs de ligne alternées : une ligne est peinte dans la couleur définie dans la propriété 'Color', et la ligne suivante dans la couleur définie dans la propriété AlternateColor.
* **AutoAdvance** : Lorsque le contenu d'une cellule est modifié et que l'utilisateur appuie sur les touches "Enter" ou "Tab cette propriété détermine quelle cellule est sélectionnée ensuite. Par défaut (aaRight) il s'agit de la cellule suivante dans la ligne actuelle, jusqu'à ce que la fin de la ligne soit atteinte. La définition de cette propriété à aaNone désactive l'avance automatique.
* **AutoEdit** : Cliquer sur une cellule la met en mode édition. La principale différence avec l'option goAlwaysShowEditor est qu'avec cette dernière, les cellules sont toujours en mode édition. L'option **AutoEdit** met une cellule en mode édition si elle est cliquée. Si ni AutoEdit, ni goAlwaysShowEditor ne sont spécifiés, alors 2 clics sont nécessaires pour mettre une cellule en mode édition : un pour sélectionner la cellule et un autre pour la mettre en mode édition. pour sélectionner la cellule et un autre pour la mettre en mode édition.
* **AutoFillColumns** : Si la valeur est True, les colonnes sont automatiquement redimensionnées de manière à ce qu'elles occupent toute la largeur de la grille. Toutes les colonnes sont redimensionnées de manière égale, mais cela peut être arrangé à l'aide de certaines propriétés.
* **ExtendedSelect** : Permet de sélectionner plusieurs cellules même si la grille est éditable. Dans Delphi, les sélections de plage ne sont pas possibles pour les grilles éditables.
* **Flat** Si cette propriété est définie sur True, les cellules ﬁxes seront dessinées en utilisant un aspect plat. (Cette propriété peut disparaître pour être remplacée par une option TitleStyle supplémentaire).
* **HeaderHotZones** : Détermine les cellules ﬁxes qui réagissent au hotracking (voir ci-dessous).
* **HeaderPushZones** : Détermine les cellules ﬁxes qui réagissent à la poussée (voir ci-dessous).
* **TitleStyle** : Détermine la manière dont les cellules ﬁxes sont dessinées. Il peut s'agir de tsLazarus, tsNative ou tsDefault. Pour tsLazarus, un aspect plat (ﬂat) est utilisé et la propriété Flat est ignorée.
* **UseXORFeatures** : Si cette propriété vaut True, le rectangle de mise au point sera dessiné à l'aide d'un stylo XOR, de sorte qu'il sera visible quelle que soit la couleur d'arrière-plan. La propriété Options est une propriété d'ensemble et possède les éléments de Delphi, mais en plus de ceux-ci, elle possède des propriétés supplémentaires.
* **goColSpanning** : Cette option permet au texte de s'étendre sur autant de colonnes que nécessaire pour afficher la cellule.
* **goDblClickAutoSize** : Cette option permet d'activer une fonction que l'on trouve par exemple dans Excel : double-cliquer sur la bordure entre deux cellules d'en-tête de colonne permet de redimensionner automatiquement la colonne à gauche : la colonne sera élargie ou réduite de manière à ce que sa largeur corresponde à l'entrée de texte la plus large de la colonne. L'option goColsizing doit être spécifiée pour que cette option prenne effet.
* **goFixedRowNumbering** : Si cette option est activée, le numéro de ligne est affiché dans la première colonne.
* **goHeaderHotTracking** : Si cette option est activée, la cellule d'en-tête sur laquelle le pointeur de la souris est actuellement positionné sera affichée dans une fenêtre de dialogue. Les cellules pour lesquelles le suivi à chaud est activé peuvent être spécifiées dans la propriété HeaderHotZones.
* **goHeaderPushedLook** : Si cette option est activée, le clic sur une cellule de l'en-tête ressemblera à un bouton pressé. Les cellules ﬁxes pour lesquelles cette option est activée peuvent être spécifiées dans la propriété HeaderPushZones.
* **goRelaxedRowSelect** :Si cette propriété est spécifiée, la cellule actuellement sélectionnée est dessinée à l'envers par rapport au reste de la cellule sélectionnée.
* **goScrollKeepVisible** : Si cette propriété est spécifiée, lors du défilement avec la barre de défilement, la cellule sélectionnée reste à la même position par rapport au haut de la. Par défaut la cellule sélectionnée défile avec les autres cellules.
* **goSmoothScroll** : Si cette option n'est pas activée, la grille défile cellule par cellule, c'est-à-dire que la rangée supérieure affiche toujours une cellule complète. Si cette option est activée, la grille défile pixel par pixel. Évidemment, cette option n'a d'effet que si goThumbTracking est spécifié.

L'option goDblClickAutoSize ne fonctionne que pour les colonnes qui ne sont pas des colonnes ﬁxes. Heureusement, la grille dispose d’un bouton qui, lorsqu'il est cliqué, va redimensionner les colonnes ﬁxes :

*procedure TMainForm.BAutoSizeFixedClick(Sender: TObject);*

*Var I : integer;*

*begin*

*For I:=0 to SGDemo.FixedCols-1 do*

*SGDemo.AutoSizeColumn(i);*

*end;*

La méthode AutoSizeColumns de la grille dimensionne automatiquement toutes les colonnes de la grille.

En plus de ces propriétés supplémentaires, la grille peut être modifiée lors de la conception : il est ainsi possible de définir les chaînes de la grille à l'aide de l'éditeur de composants (disponible dans le menu contextuel du concepteur en cliquant avec le bouton droit de la souris sur le composant de la grille).

3 COLONNES

La plus grande différence avec la grille de Delphi est peut-être que l'implémentation de Lazarus possède également une propriété Columns : une collection d'éléments TGridColumn qui décrivent les propriétés d'une colonne dans une grille, similaire à la propriété columns d'une TDBGrid. La propriété est mutuellement exclusive avec la propriété ColCount : dès que des colonnes sont utilisées, la propriété ColCount ne peut plus être définie. La propriété Columns décrit uniquement les colonnes non ﬁxes. Par exemple, s'il y a :

* 2 colonnes ﬁxées (FixedColCount=2) et
* 3 colonnes ﬁxées (FixedColCount=2),
* la grille sera composée de 3 colonnes ﬁxes et avec Columns[I], la grille affichera 5 colonnes.

La propriété Columns.Enabled détermine si les colonnes sont utilisées.

*procedure TMainForm.BColumnsClick(Sender: TObject);*

*Var I, ACount : Integer;*

*begin*

*ACount := SGDemo.ColCount-SGDemo.FixedColCount;*

*SGDemo.Columns.Clear;*

*For I := 1 to ACount do*

*With TGridColumn(SGDemo.Columns.Add) do*

*Title.Caption := ’Column ’+IntToStr(i);*

*end;*

Les propriétés d'une colonne peuvent être examinées et définies en double-cliquant sur la colonne :

*procedure TMainForm.SGDemoDblClick(Sender: TObject);*

*Var C : TGridColumn;*

*F : TColumnPropertiesForm;*

*begin*

*If (SGDemo.Col >= SGDemo.FixedCols) and SGDemo.Columns.Enabled then*

*begin*

*If (FColProps = Nil) then*

*begin*

*F := TColumnPropertiesForm.Create(Self);*

*F.OnDestroy := @DoFree;*

*FColProps := F*

*end*

*else*

*F := TColumnPropertiesForm(FColProps);*

*F.Column := SGDemo.Columns[SGDemo.Col-SGDemo.FixedCols];*

*F.Show;*

*end;*

*end;*

Le code vérifie d'abord si des colonnes sont utilisées et, si c'est le cas, une instance TColumnPropertiesform est créée, si nécessaire, et affichée. La colonne cliquée est transmise à ce formulaire. Tout comme le formulaire principal, le formulaire TColumnPropertiesform contient une TIPropertyGrid, qui sera utilisée pour afficher les propriétés de la colonne cliquée. Si une entrée précédemment visible est présente, elle est alors affichée et réutilisé. Pour ce faire, le formulaire principal est détruit : dans DoFree, la référence FColProps au formulaire TColumnProperties est mise à zéro.

4. EDITEUR

La grille de Lazarus n'utilise pas la classe TInPlaceEditor comme le fait la grille de Delphi. Au lieu de cela, elle utilise directement les contrôles TButton, TMaskEdit ou TComboBox.

Le choix de l'un ou l'autre de ces contrôles utilisé dans une cellule est contrôlé par la propriété ButtonStyle de TGridColumn : le contrôle à utiliser peut être défini pour chaque colonne de la grille.

L'apparence des contrôles d'édition ne peut pas être définie par les propriétés de la grille. En revanche, l'événement OnSelectEditor permet de personnaliser l'éditeur lorsqu'il est sur le point d'être affiché.

Voici un exemple de gestionnaire d'événement OnSelectEditor :

*procedure TMainForm.DoSelectEditor(Sender: TObject; aCol, aRow: Integer; var Editor: TWinControl);*

*begin*

*If Editor is TStringCellEditor then*

*CustomizeEdit(aCol,ARow,Editor as TStringCellEditor)*

*else*

*if Editor is TButtonCellEditor then*

*CustomizeButton(aCol,ARow,Editor as TButtonCellEditor)*

*else*

*if Editor is TPickListCellEditor then*

*CustomizePickList(aCol,ARow,Editor as TPickListCellEditor);*

*end;*

Étant donné que l'éditeur réel peut appartenir à plusieurs classes différentes, l'argument Editor est de type TWinControl, ce qui nécessite de le convertir en son type réel. L'argument TStringCellEditor, TButtonCellEditor et TPickListCellEditor sont les descendants de TCustomMaskedEdit, TButton et TComboBox, utilisés par la grille.

Voici un exemple de personnalisation :

*procedure TMainForm.CustomizeEdit(Acol,ARow : Integer; Editor : TStringCellEditor);*

*begin*

*Editor.PasswordChar:=’\*’;*

*Editor.MaxLength:=5;*

*end;*

4.1 Placer et utiliser un dBLookUpCombobox dans un dBGrid [[1]](#footnote-1)

Comment faire la meilleure grille d'édition de données de tous les temps? Ci-dessous sont indiquées les instructions pour créer une interface utilisateur capable de modifier les champs de recherche à l’intérieur d’un dBGrid. Plus précisément, nous allons chercher comment implémenter un DBLookupComboBox dans une cellule d'un dBGrid. Cela fera appel à des informations provenant d'une source de données qui seront utilisées pour remplir une boîte déroulante.

Pour montrer une DBLookupComboBox à l'intérieur d'une cellule d'un dBGrid, il faut d'abord en rendre un disponible au moment de l'exécution...

4.2 Comment créer une recherche avec une DBLookupComboBox

NOTA : Le fichier du dBGrid ne doit pas comporter de champ calculé pointant sur le champ ListField/KeyField du fichier dBLookUpCombobox[[2]](#footnote-2) car il pointerait alors sur lui-même et renverrait une zone blanche.

Sélectionner la page "Contrôle des données" sur la palette des composants et choisissez un DBLookupComboBox. En déposer un n'importe où sur le formulaire et laissez le nom par défaut de "DBLookupComboBox1." Peu importe où il sera car la plupart du temps, il sera invisible ou flottant sur la grille.

Ajoutez un autre composant DataSource et DataSet pour "remplir" la zone de liste déroulante avec des valeurs. Déposer un TDataSource (avec le nom DataSource2) et TAGoQuery (le nommer AdoQuery1) n'importe où sur le formulaire.

Pour qu'un DBLookupComboBox fonctionne correctement, plusieurs autres propriétés doivent être définies; ils sont la clé de la connexion de recherche:

* *DataSource* et *DataField* déterminer la connexion principale. Le DataField est un champ dans lequel nous insérons les valeurs de recherche.
* *ListSource* est la source de l'ensemble de données de recherche.
* *KeyField* identifie le champ dans le *ListSource* qui doit correspondre à la valeur de la *DataField* champ.
* *ListFields* est le champ, ou les champs, de l'ensemble de données de recherche qui sont réellement affichés dans le combo. ListField peut afficher plus d'un champ mais les multiples doivent être séparés par des points-virgules. Dans ce cas, il faut définir une valeur suffisamment grande pour *DropDownWidth* (d'un ComboBox) pour vraiment voir plusieurs colonnes de données.

Définir toutes les propriétés importantes à partir du code (dans le formulaire du gestionnaire d’[Événement OnCreate](https://www.thoughtco.com/delphi-programming-tutorials-4133473)) ou directement dans un datamodule :

*procedure TForm1.FormCreate(Sender: TObject);*

*begin*

*with DBLookupComboBox1 do*

*begin*

*DataSource := DataSource1; // -> AdoTable1 -> DBGrid1*

*ListSource := DataSource2;*

*DataField := 'AuthorEmail'; // from AdoTable1 - displayed in the DBGrid*

*KeyField := 'Email';*

*ListFields := 'Name; Email';*

*Visible := False;*

*end;*

*DataSource2.DataSet := AdoQuery1;*

*AdoQuery1.Connection := AdoConnection1;*

*AdoQuery1.SQL.Text := 'SELECT Name, Email FROM Authors';*

*AdoQuery1.Open;*

*end;*

**Remarque:** Lorsque l’on souhaite afficher plusieurs champs dans une DBLookupComboBox, comme dans l'exemple ci-dessus, il convient de s’assurer que toutes les colonnes sont visibles. Cela se fait en définissant la propriété DropDownWidth.

Cependant, on peut observer qu'au départ, on doit définir cela sur une très grande valeur, ce qui fait que la liste des résultats est trop large (dans la plupart des cas). Une solution consiste à définir la largeur d'affichage d'un champ particulier indiquée dans une [liste déroulante](https://www.thoughtco.com/drop-down-pick-list-into-dbgrid-4077749).

Ce code, placé à l'intérieur de l'événement OnCreate pour le formulaire, garantit que le nom de l'auteur et son e-mail sont affichés à l'intérieur de la liste déroulante:

*AdoQuery1.FieldByName ( 'Email' ).DisplayWidth: = 10;*

*AdoQuery1.FieldByName ( 'Name' ).DisplayWidth: = 10;*

*AdoQuery1.DropDownWidth: = 150;*

Il reste à faire passer une boîte de liste déroulante sur une cellule (en mode édition), affichant le champ AuthorEmail. Tout d'abord, nous devons nous assurer que la DBLookupComboBox1 est déplacée et dimensionnée sur la cellule dans laquelle le champ AuthorEmail est affiché.

*TForm1.DBGrid1DrawColumnCell (Sender: TObject; const Rect: TRect; DataCol: Integer; Column: TColumn; State: TGridDrawState);*

***begin***

***if*** *(gdFocused* ***in*** *State)* ***then***

***begin***

***if*** *(Column.Field.FieldName = DBLookupComboBox1.DataField)* ***then***

***with*** *DBLookupComboBox1* ***do***

***begin***

*Left := Rect.Left + DBGrid1.Left + 2;*

*Top := Rect.Top + DBGrid1.Top + 2;*

*Width := Rect.Right - Rect.Left;*

*Width := Rect.Right - Rect.Left;*

*Height := Rect.Bottom - Rect.Top;*

*Visible := True;*

***end****;*

*DataCol := 1 ; //Le numéro de la colonne…*

***end;***

***end****;*

Ensuite, lorsque nous quittons la cellule, nous devons cacher la zone de liste déroulante:

***procedure*** *TForm1.DBGrid1ColExit(Sender: TObject);*

***begin***

***if*** *DBGrid1.SelectedField.FieldName = DBLookupComboBox1.DataField* ***then***

*DBLookupComboBox1.Visible := False ;*

***end****;*

Noter que lorsque dBGrid est en mode édition, toutes les frappes vont dans la cellule du DBGrid, mais nous devons nous assurer qu'elles sont envoyées au DBLookupComboBox. Dans le cas d'un DBLookupComboBox, la clé [Tab] est intéresante car elle doit déplacer la mise au point d'entrée vers la cellule suivante.

***procedure*** *TForm1.DBGrid1KeyPress(Sender: TObject; var Key: Char);*

*Var WM\_Char : word = 0;*

***begin***

***if*** *(key = Chr(9))* ***then*** *Exit;*

***if*** *(DBGrid1.SelectedField.FieldName = DBLookupComboBox1.DataField)* ***then***

***begin***

*DBLookupComboBox1.SetFocus;*

*SendMessage(DBLookupComboBox1.Handle, WM\_Char, word(Key), 0);*

***end;***

***end****;*

Lorsqu’un article (une "ligne") est choisi dans une DBLookupComboBox, la valeur ou la KeyField du champ est stocké comme valeur du champ DataField.

5 ÉDITEURS ALTERNATIFS

Le gestionnaire d'événement OnSelectEditor transmet le paramètre Editor par référence peut être utilisé pour définir le contrôle comme n'importe quel descendant de TWinControl. Pour le démontrer, un petit programme de démonstration qui remplit la grille avec des dates et remplace l'éditeur par un contrôle TDateEdit dans le gestionnaire OnSelectEditor :

*procedure TMainForm.DoSelectEditor(Sender: TObject; aCol, aRow: Integer; var Editor: TWinControl);*

*begin*

*if (ACol>0) and (ARow>0) then*

*begin*

*If (DE=Nil) then*

*begin*

*DE:=TStringDateEditor.Create(Self);*

*DE.Parent:=StringGrid1;*

*DE.Visible:=False;*

*end;*

*Editor:=DE;*

*end;*

*end;*

Le contrôle TStringDateEditor est un descendant du contrôle TDateEdit. Le contrôle TDateEdit ne peut pas être utilisé directement : la grille utilise certains messages pour communiquer avec le contrôle d'édition :

* GM\_SETVALUE est envoyé lorsque la grille veut définir le texte à éditer.
* GM\_GETVALUE est envoyé lorsque la grille veut récupérer le texte édité.
* GM\_SETGRID est envoyé avec l'instance de grille pour définir l'instance de grille.
* GM\_SETBOUNDS est envoyé pour définir la position du contrôle de l'éditeur.
* GM\_SELECTALL est envoyé lorsque le contrôle doit effectuer une "sélection complète".
* GM\_SETMASK est envoyé lorsque le masque d'édition doit être défini.
* GM\_SETPOS est envoyé pour définir les coordonnées de la cellule à éditer.

Le contrôle de l'éditeur doit traiter ces messages et prendre les mesures qui s'imposent. Tous ces messages ne doivent pas être pris en compte. La déclaration de classe suivante du contrôle TStringDateEditor montre à quels messages elle répond :

*TStringDateEditor=class(TDateEdit)*

*private*

*FGrid: TCustomGrid;*

*FCol,FRow:Integer;*

*protected*

*procedure msgSetValue(var Msg: TGridMessage); message GM\_SETVALUE;*

*procedure msgGetValue(var Msg: TGridMessage); message GM\_GETVALUE;*

*procedure msgSetGrid(var Msg: TGridMessage); message GM\_SETGRID;*

*procedure msgSetBounds(var Msg: TGridMessage); message GM\_SETBOUNDS;*

*procedure msgSelectAll(var Msg: TGridMessage); message GM\_SELECTALL;*

*end;*

L'enregistrement TGridMessage se définit comme suit :

*TGridMessage=record*

*LclMsg: TLMessage;*

*Grid: TCustomGrid;*

*Col,Row: Integer;*

*Value: string;*

*CellRect: TRect;*

*Options: Integer;*

*end;*

La signification des champs doit être claire d'après leur nom. Le champ Options peut être utilisé pour communiquer avec le réseau lorsque le message GM\_SETGRID est reçu :

*procedure TStringDateEditor.msg\_SetGrid(var Msg: TGridMessage);*

*begin*

*FGrid:=Msg.Grid;*

*Msg.Options:=EO\_SELECTALL or EO\_HOOKKEYPRESS or EO\_HOOKKEYUP;*

*end;*

* EO\_AUTOSIZE Indique à la grille qu'elle peut définir directement la taille et la position de l'éditeur. Si cela n'est pas spécifié, la grille enverra un appel GM\_SETBOUNDS.
* EO\_HOOKKEYDOWN, EO\_HOOKKEYPRESS et EO\_HOOKKEYUP Indiquent à la grille de définir les gestionnaires d'événements OnKeyDown, OnKeyPress et OnKeyUp du contrôle de l'éditeur pour gérer certaines touches courantes. d'événements OnKeyDown, OnKeyPress et OnKeyUp du contrôle de l'éditeur pour gérer certaines touches courantes.
* EO\_SELECTALL Indique à la grille qu'elle peut envoyer un message GM\_SELECTALL au contrôle de l'éditeur.

Les gestionnaires de messages GM\_SETVALUE et GM\_GETVALUE sont assez simples :

*procedure TStringDateEditor.msgSetValue(var Msg: TGridMessage);*

*begin*

*Self.Date:=StrToDate(Msg.Value);*

*end;*

*procedure TStringDateEditor.msgGetValue(var Msg: TGridMessage);*

*begin*

*Msg.Value:=DateToStr(Self.Date);*

*end;*

Le message GM\_SETBOUNDS est utilisé pour certaines manipulations spéciales :

*procedure TStringDateEditor.msgSetBounds(var Msg: TGridMessage);*

*Var R : TRect;*

*begin*

*R := Msg.CellRect;*

*R.Right := R.Right-ButtonWidth;*

*BoundsRect := R;*

*end;*

La largeur du contrôle TDateEdit ne comprend pas la largeur du bouton de commande. C'est pourquoi bouton doit être soustraite du rectangle de délimitation avant de définir la propriété BoundsRect. la propriété BoundsRect.

Il ne reste plus qu'à implémenter le gestionnaire GM\_SELECTALL :

*procedure TStringDateEditor.msgSelectAll(var Msg: TGridMessage);*

*begin*

*SelectAll;*

*end;*

6 CONCLUSION

Le contrôle de grille de Lazarus présente quelques avantages par rapport à l'implémentation de la grille standard dans Delphi : il offre plus de propriétés pour personnaliser le comportement et l'apparence de la grille. Comme le montre le code de cet article, il est également très facile d'améliorer l'expérience d'édition de la grille. En fait, il est assez facile de créer un descendant qui permet d'utiliser des contrôles d'édition plus avancés.

1. Source : Gajic, Zarko. "Making a Drop Down List in a DBGrid." ThoughtCo, Feb. 16, 2021, thoughtco.com/place-dblookupcombobox-into-dbgrid-4077834. [↑](#footnote-ref-1)
2. Ajouté par Patrick Leroy le 4 août 2023 [↑](#footnote-ref-2)