



Developers' Guide to DataGridView Extension 1.1.1



Revision date 27 February 2008



Contents

- 1. General 4
 - 1.1. Installation structure 4
 - 1.2. Assemblies 4
- 2. Common 5
 - 2.1. Supported data sources..... 5
 - 2.2. Manage grid view programmatically 5
 - 2.3. Open settings editor for a managed grid view 6
 - 2.4. Explicit refresh of a managed grid view..... 6
 - 2.5. Changing Xml persistence folder..... 7
 - 2.6. Custom columns 7
- 3. Export 7
 - 3.1. MS Excel 8
 - 3.2. Html..... 8
 - 3.3. Pdf..... 9
 - 3.4. Export using the default settings..... 9
 - 3.5. Export using the exporting manager 10
 - 3.6. Events and error handling 11
 - 3.7. Export using the appropriate export settings editor dialog..... 11
 - 3.8. Creating custom exporter 12
 - 3.9. Changing exporters folder 12
- 4. Security 13
 - 4.1. Default permissions for all grid view controls in the application..... 13
 - 4.2. Default permissions for all columns in the application 14
 - 4.3. Permissions for a particular grid view control 14
 - 4.4. Permissions for a particular column 15
 - 4.5. General permissions for the extension..... 16
- 5. Find..... 17
 - 5.1. How to find specific cell value 17



- 5.2. How to find a value using the find dialog..... 17
- 6. Print 17
 - 6.1. Silent print (No print dialog)..... 18
 - 6.2. Print using the print dialog..... 18
 - 6.3. Print preview..... 18
- 7. FAQ 19
 - 7.1. Which component does the DataGridView Extension extends?..... 19
 - 7.2. Does DataGridView Extension component inherit the standard DataGridView control?..... 19
 - 7.3. In what programming language is DataGridView Extension written? 19
 - 7.4. Which version of .NET framework does DataGridView Extension support? 19
 - 7.5. Does DataGridView Extension have design time support?..... 19
 - 7.6. What are the requirements in order to have a DataGridView control managed by the DataGridView Extension? 19
 - 7.7. To which assemblies do I have to add references in my project?..... 19
 - 7.8. How is DataGridView Extension integrated?..... 19
 - 7.9. How DataGridView Extension stores its data?..... 20
 - 7.10. Where is all Xml data stored on the user's computer?..... 20
 - 7.11. What is the structure of the persistence folder?..... 20
 - 7.12. Does DataGridView Extension support custom columns?..... 20
 - 7.13. How does the export functionality work?..... 20
 - 7.14. Which version of MS Excel is supported by the export? 20
 - 7.15. The export to Excel does not work what should I do?..... 20
 - 7.16. How to add DataGridView Extension to my VS.NET 2005 Toolbox? 21
 - 7.17. How many DataGridView controls can be managed by a single instance of the Extension? 21
 - 7.18. How many instance of the extension can I have in single .NET application? . 21
 - 7.19. Why the colors of my column and row headers are not applied although I have changed them? 21



1. General

DataGridView Extension is a component that extends the basic DataGridView control (provided with .NET 2.0 Framework) with additional functionality without inheritance.

DataGridView Extension allows both developers and end users to control specific features of the DataGridView control in runtime environment. As part of these features, the appearance and behavior of the control could be modified also.

DataGridView Extension provides additional functionality, such as printing, exporting of data and visualization to MS Excel, Pdf documents and Html pages, searching for text in the control and theme management.

Major functionality of DataGridView Extension is that it allows persistence and export/import of all changes made to the control: its columns widths, ordering, headers width and height, appearance and behavior.

DataGridView Extension has the ability to manage multiple DataGridView controls in a single project and to store all user settings for every and each of them.

DataGridView Extension allows saving of themes defined by the customer. Themes greatly reduce the labor needed to set a professional looking application. Customers can change the entire application's look and feel with a single click by changing the default theme. Themes can be saved in different applications and even backup.

The **DataGridView Extension** comes with a set of custom column types that will give the developer additional functionalities of the grid itself.

1.1. Installation structure

The default installation folder is 'C:\Program Files\CompleTIT\DataGridView Extension' but during the installation the user can change it. After the installation has finished successfully, the following folder structure is created:

```
\v1.1
  \Bin - Contains all DLLs you need to develop applications using DGV Extension;
  \Help - Contains the end-user manual as Pdf file;
  \PIAs - Contains the Primary Interop Assemblies (PIAs) and the registry files needed for the export to MS
Excel;
  \Samples
    \Demo - Demo application DLLs and executable
    \CSharp - VS.2005 sample project written in C#
    \VisualBasic.Net - VS.2005 sample project written in VisualBasic.Net
```

1.2. Assemblies

- **'DataGridViewExtension.dll'** – This is the main assembly of the DataGridView Extension. It contains some major functionality such as Print, Find, Themes support, Xml persistence and others.
- **'DataGridViewExtension.XmlSerializers.dll'** – Xml persistence optimization assembly; this assembly is optional and can be removed freely but keep in mind that it reduces the initial loading time for all persisted Xml settings drastically.



- **'DGVEExcelExporting.dll'** – Export to MS Excel; this assembly is optional and can be removed if the export to MS Excel file format should not be supported;
- **'DGVEHtmlExporting.dll'** – Export to Html; this assembly is optional and can be removed if the export to Html file format should not be supported;
- **'DGVEPdfExporting.dll'** – Export to Pdf; this assembly is optional and can be removed if the export to Pdf file format should not be supported.

The assemblies described above are available in folder '<InstallationPath>\v1.1\Bin' created during the installation. All assemblies have version 1.1 and are strongly named so they can be registered in the Global Assembly Cache (GAC). By default they are not.

For more information about the assemblies you have to reference from your projects please read ['To which assemblies do I have to add references in my project'](#) in the FAQ section.

2. Common

Many of the functionalities developed for the DataGridView Extension are accessible not only runtime via the visual editors but also through an API.

2.1. Supported data sources

DataGridView Extension supports both bound and unbound grid view controls. All kinds of data sources are supported i.e. BindingSource, DataView, DataTable, DataSet, DataViewManager, custom collections and others. It is important to know that all customizations made to a bound grid view controls are related to its current data source; this means that in a single grid view control you can have different visual representation (themes, sizes, selection etc) depending on the underlying data source. In case the data source is changed the extension will immediately apply the last saved visual settings for the new data source if such exists. Further in the guide you will see how to force such kind of refresh manually using the API.

Tip: If you have two or more grid controls in your application that visualize the same type of data i.e. Clients they will share one Xml settings file thus one visual representation, if you set one and the same name to both controls i.e. 'dgrClients'.

2.2. Manage grid view programmatically

Grid view control can be managed by the extension either design time or manually using the API. The sample bellow shows how to do it manually from within your code.

Add reference to the main assembly 'DataGridViewExtension.dll' and all export DLLs you wish to be supported by the extension in your application like 'DGVEExcelExporting.dll'.

C#

```
using CompletIT.Windows.DataGridViewExtension;  
  
//Create instance of the extension and manage the grid control  
DataGridViewExtensionComponent dgvExtension = new DataGridViewExtensionComponent();  
dgvExtension.SetManagedByExtension( dataGridView, true );  
//Manage other grids here ...
```

VB.NET



```
Imports CompletIT.Windows.DataGridViewExtension
```

```
'Create instance of the extension and manage the grid control
```

```
Dim DGVExtension As DataGridViewExtensionComponent = New DataGridViewExtensionComponent()
```

```
DGVExtension.SetManagedByExtension( Me.DataGridViewControl, True )
```

```
'Manage other grids here ...
```

2.3. Open settings editor for a managed grid view

Run-time the settings editor is accessible through the Extension button available in the upper left corner of the managed grid view control or through its context menu. Below you can see a code snippet showing you how to popup the settings editor using the API. Keep in mind that the settings editor can be opened only for extended grid view controls.

After a grid has been managed either manually or design-time by the extension, its settings editor can be opened in the way shown below.

C#

```
using CompletIT.Windows.DataGridViewExtension;
```

```
//Note! You can show the settings editor
```

```
//dialog only for already managed grid view.
```

```
DataGridViewExtensionComponent.ExtensionUI.ShowEditor( dataGridView );
```

VB.NET

```
Imports CompletIT.Windows.DataGridViewExtension
```

```
'Note! You can show the settings editor
```

```
'dialog only for already managed grid view.
```

```
DataGridViewExtensionComponent.ExtensionUI.ShowEditor( Me.DataGridViewControl )
```

2.4. Explicit refresh of a managed grid view

Visual representation (themes, hidden/frozen columns etc) of a grid view is bound to the underlying data source, so in case the data source is changed the visualization should be also refreshed. In rare cases it might be possible this not to happen automatically. In such cases you can force this refresh manually like it is shown in the sample below or reset the 'DataSource' and 'DataMember' properties of the grid view control.

C#

```
using CompletIT.Windows.DataGridViewExtension;
```

```
//Note! You can refresh only managed grid views.
```

```
//Keep in mind that when you call this method all of your data bound columns
```

```
//will be recreated, while the unbound columns will stay the same
```

```
DataGridViewExtensionComponent.Extension.Refresh( dataGridView );
```

VB.NET

```
Imports CompletIT.Windows.DataGridViewExtension
```

```
'Note! You can refresh only managed grid views.
```

```
'Keep in mind that when you call this method all of your data bound columns
```

```
'will be recreated, while the unbound columns will stay the same
```

```
DataGridViewExtensionComponent.Extension.Refresh( Me.DataGridViewControl );
```



2.5. Changing Xml persistence folder

All changes made run-time by the user to a managed grid view control are persisted in Xml format on your hard drive. By default all Xml files are stored in the following folder 'C:\Documents and Settings\{Username}\Application Data\{CompanyName}\{ApplicationName}\{ApplicationVersion}\' but it can be easily changed to any other location.

Add reference to the main assembly 'DataGridViewExtension.dll'.

C#

```
using CompletIT.Windows.DataGridViewExtension;  
using CompletIT.Windows.DataGridViewExtension.Persistence;  
using CompletIT.Windows.DataGridViewExtension.Persistence.Xml;  
  
//Set the new destination folder  
DGVEXmlPersistenceManager.ExtensionSettingsPath = "C:\\MyNewPersistenceFolder";  
//Force update of all controls with the settings from the new folder  
DataGridViewExtensionComponent.Extension.UpdateAllGridsWithPersistedSettings();  
DGVEThemesManager.UpdateListWithSavedThemes();
```

VB.NET

```
Imports CompletIT.Windows.DataGridViewExtension  
Imports CompletIT.Windows.DataGridViewExtension.Persistence  
Imports CompletIT.Windows.DataGridViewExtension.Persistence.Xml  
  
'Set the new destination folder  
DGVEXmlPersistenceManager.ExtensionSettingsPath = "C:\\MyNewPersistenceFolder"  
'Force update of all controls with the settings from the new folder  
DataGridViewExtensionComponent.Extension.UpdateAllGridsWithPersistedSettings()  
DGVEThemesManager.UpdateListWithSavedThemes()
```

2.6. Custom columns

The DataGridView Extension main assembly contains a set of custom columns that can be used in your applications. The columns are:

- '**DGVENumericUpDownColumn**' – allows display and edit of numeric data with values between 0 and 100 (For example - percentage editing).
- '**DGVEDateTimePickerColumn**' – allows display and edit of date time data.

Note: The custom columns will be available in "Add column" dialog of Visual Studio after you add reference to the assembly 'DataGridViewExtension.dll'.

3. Export

The export mechanism of the DataGridView Extension is developed in a way that allows usage of the different exports either as part of the extension application or as separate exports accessed programmatically. The system is open and you can develop your own exports to any desired file formats in just a few steps described in more details at the end of this section.



For more information you can read '[How does the export functionality work](#)' in the FAQ section.

Note: If you want to export the content of a DataGridView control using the API, you do not have to manage it; just pass it as a parameter to the appropriate exporter.

3.1. MS Excel

Add reference to the main assembly '*DataGridViewExtension.dll*' and export DLL '*DGVEExcelExporting.dll*'.

C#

```
using CompletIT.Windows.Forms.Export.Excel;
```

```
DGVEExcelExportSettings exportSettings = new DGVEExcelExportSettings();  
exportSettings.ExportFileName = "C:\\Export.xls" //Absolute path to the export file  
exportSettings.OpenFileAfterGeneration = true; //Open generated file after export
```

```
DGVEExcelExporter exporter = new DGVEExcelExporter();  
exporter.Export( dataGridView, exportSettings );
```

VB.NET

```
Imports CompletIT.Windows.Forms.Export.Excel
```

```
Dim ExportSettings As DGVEExcelExportSettings = New DGVEExcelExportSettings()  
ExportSettings.ExportFileName = "C:\\Export.xls" 'Absolute path to the export file  
ExportSettings.OpenFileAfterGeneration = True 'Open generated file after export
```

```
Dim Exporter As DGVEExcelExporter = New DGVEExcelExporter()  
Exporter.Export( Me.DataGridViewControl, ExportSettings )
```

3.2. Html

Add reference to the main assembly '*DataGridViewExtension.dll*' and export DLL '*DGVEHtmlExporting.dll*'.

C#

```
using CompletIT.Windows.Forms.Export.Html;
```

```
DGVEHtmlExportSettings exportSettings = new DGVEHtmlExportSettings();  
exportSettings.ExportFileName = "C:\\Export.html" //Absolute path to the export file  
exportSettings.OpenFileAfterGeneration = true; //Open generated file after the export
```

```
DGVEHtmlExporter exporter = new DGVEHtmlExporter();  
exporter.Export( dataGridView, exportSettings );
```

VB.NET

```
Imports CompletIT.Windows.Forms.Export.Excel
```

```
Dim ExportSettings As DGVEHtmlExportSettings = New DGVEHtmlExportSettings()  
ExportSettings.ExportFileName = "C:\\Export.html" 'Absolute path to the export file  
ExportSettings.OpenFileAfterGeneration = True 'Open generated file after export
```



```
Dim Exporter As DGVEHtmlExporter = New DGVEHtmlExporter()  
Exporter.Export( Me.DataGridViewControl, ExportSettings )
```

3.3. Pdf

Add reference to the main assembly '*DataGridViewExtension.dll*' and export DLL '*DGVEPdfExporting.dll*'.

C#

```
using CompletIT.Windows.Forms.Export.Pdf;  
  
DGVEPdfExportSettings exportSettings = new DGVEPdfExportSettings();  
exportSettings.ExportFileName = "C:\\Export.pdf" //Absolute path to the export file  
exportSettings.OpenFileAfterGeneration = true; //Open generated file after export  
  
DGVEPdfExporter exporter = new DGVEPdfExporter();  
exporter.Export( dataGridView, exportSettings );
```

VB.NET

```
Imports CompletIT.Windows.Forms.Export.Pdf  
  
Dim ExportSettings As DGVEPdfExportSettings = New DGVEPdfExportSettings()  
ExportSettings.ExportFileName = "C:\\Export.pdf" 'Absolute path to the export file  
ExportSettings.OpenFileAfterGeneration = True 'Open generated file after export  
  
Dim Exporter As DGVEPdfExporter = New DGVEPdfExporter()  
Exporter.Export( Me.DataGridViewControl, ExportSettings )
```

3.4. Export using the default settings

Each of the exporters has default settings located in the exporter class. You can access them using the property '*DefaultSettings*'. These settings will be used only in case you have started the export through the overload of the '*Export*' method which does not accept settings as parameter. See the sample below.

C#

```
using CompletIT.Windows.Forms.Export.Export.Pdf;  
  
DGVEPdfExporter pdfExporter = new DGVEPdfExporter();  
pdfExporter.DefaultSettings.ExportFileName = "C:\\Export.pdf" //Absolute path to the export file  
pdfExporter.DefaultSettings.OpenFileAfterGeneration = true; //Open generated file after export  
  
pdfExporter.Export( dataGridView );
```

VB.NET

```
Imports CompletIT.Windows.Forms.Export.Export.Pdf  
  
Dim PdfExporter As DGVEPdfExporter = New DGVEPdfExporter()  
PdfExporter.DefaultSettings.ExportFileName = "C:\\Export.pdf" 'Absolute path to the export file  
PdfExporter.DefaultSettings.OpenFileAfterGeneration = True 'Open generated file after export  
  
PdfExporter.Export( Me.DataGridViewControl )
```



3.5. Export using the exporting manager

The export can be done using the exporting manager. There are two possible ways:

C#

```
using CompletIT.Windows.Forms.Export.Export;  
using CompletIT.Windows.Forms.Export.Export.Excel;  
  
ExcelExportDescription exportDescription = new ExcelExportDescription();  
DGVEExportingManager.Export( dataGridView, exportDescription, true );
```

VB.NET

```
Imports CompletIT.Windows.Forms.Export.Export  
Imports CompletIT.Windows.Forms.Export.Export.Excel  
  
Dim ExportDescription As ExcelExportDescription = New ExcelExportDescription ()  
DGVEExportingManager.Export( Me.DataGridViewControl, ExportDescription, True )
```

or

C#

```
using CompletIT.Windows.Forms.Export.Export;  
using CompletIT.Windows.Forms.Export.Export.Excel;  
  
ExcelExportDescription exportDescription = new ExcelExportDescription();  
  
DGVEExcelExportSettings exportSettings = new DGVEExcelExportSettings();  
exportSettings.ExportFileName = "C:\\Export.xls" //Absolute path to the export file  
exportSettings.OpenFileAfterGeneration = true; //Open generated file after export  
  
DGVEExportingManager.Export( dataGridView, exportDescription, exportSettings, true );
```

VB.NET

```
Imports CompletIT.Windows.Forms.Export.Export  
Imports CompletIT.Windows.Forms.Export.Export.Excel  
  
Dim ExportDescription As ExcelExportDescription = New ExcelExportDescription ()  
  
Dim ExportSettings As DGVEExcelExportSettings = New DGVEExcelExportSettings()  
ExportSettings.ExportFileName = "C:\\Export.xls" 'Absolute path to the export file  
ExportSettings.OpenFileAfterGeneration = True 'Open generated file after export  
  
DGVEExportingManager.Export( Me.DataGridViewControl, ExportDescription, ExportSettings, True )
```

As you can see the difference is that in the first example you do not have to pass settings to the exporting manager, but instead of this the last persisted settings will be used.

You can set a value whether the progress form should be visible or not during the export process.

C#

```
using CompletIT.Windows.Forms.Export.Export;
```



```
DGVEExportingManager.ShowProgressForm = true;
```

VB.NET

```
Imports CompletIT.Windows.Forms.Export.Export
```

```
DGVEExportingManager.ShowProgressForm = True
```

3.6. Events and error handling

Each exporter implements 'IDGVEExporter' and has 'ExportStart', 'ExportEnd' and 'ExportFailed' events; they are raised when you start the export using one of the overloaded versions of the method 'Export'. You can attach to these events so you will be notified for the start, end and for the errors occurred during the export.

3.7. Export using the appropriate export settings editor dialog

Each one of the exporters (MS Excel, Pdf and Html) developed by CompletIT has custom visual editor for the settings used during the export process. These editors are public and can be easily used both wrapped in dialogs or as editor controls.

List of the settings editor controls with their full names and the assemblies they belong to.

- **MS Excel** export settings editor control is
'CompletIT.Windows.Forms.Export.Excel.DGVEExcelExportSettingsEditor' located in assembly 'DGVEExcelExporting.dll'.
- **Pdf** export settings editor control is
'CompletIT.Windows.Forms.Export.Pdf.DGVEPdfExportSettingsEditor' located in assembly 'DGVEPdfExporting.dll'.
- **Html** export settings editor control is
'CompletIT.Windows.Forms.Export.Html.DGVEHtmlExportSettingsEditor' located in assembly 'DGVEHtmlExporting.dll'.

The sample code below show how to popup the settings editor control for Pdf export settings. You can use the same approach for MS Excel and Html export settings editors.

C#

```
using CompletIT.Windows.Forms.Export.Export;
using CompletIT.Windows.Forms.Export.Export.Pdf;

DGVEPdfExporter pdfExporter = new DGVEPdfExporter();
DGVEBaseExportSettingsEditorForm dialog =
    DGVEExportSettingsEditorFormBuilder.CreateWrappingForm( pdfExporter );
//You can also use the other overload of the CreateWrappingForm method
//DGVEBaseExportSettingsEditorForm dialog =
//    DGVEExportSettingsEditorFormBuilder.CreateWrappingForm( new DGVEPdfExportSettingsEditor() );
dialog.Settings = new DGVEPdfExportSettings();
if ( DialogResult.OK != dialog.ShowDialog() )
    return;
pdfExporter.Export( dataGridView, dialog.Settings );
```

VB.NET

```
Imports CompletIT.Windows.Forms.Export.Export
Imports CompletIT.Windows.Forms.Export.Export.Pdf
```

```
Dim PdfExporter As DGVEPdfExporter = New DGVEPdfExporter()
```



```
Dim Dialog As DGVEBaseExportSettingsEditorForm = _
    DGVEExportSettingsEditorFormBuilder.CreateWrappingForm( PdfExporter )
'You can also use the other overload of the CreateWrappingForm method
'Dim Dialog As DGVEBaseExportSettingsEditorForm = _
'    DGVEExportSettingsEditorFormBuilder.CreateWrappingForm( New DGVEPdfExportSettingsEditor() )
Dialog.Settings = New DGVEPdfExportSettings()
If ( DialogResult.OK <> Dialog.ShowDialog() ) Then
    return
End If
PdfExporter.Export( Me.DataGridViewControl, Dialog.Settings )
```

3.8. Creating custom exporter

If you want to create custom export you have to follow the steps bellow:

- Create export settings class i.e. 'MyExportSettings' that extends 'DGVEBaseExportSettings';
- If the export settings need an editor control, create the control and implement 'IDGVEExportSettingsEditor';
- Create the exporter class itself i.e. 'MyExporter' that implements the interface 'IDGVEExporter'. This is the class where the actual export happens;
- Create class i.e. 'MyExportDescription' that inherits the 'BaseExportDescription'. This class is responsible for providing general information about the export such as name, description, icon etc;
- Set the attribute 'ExporterDescriptorAttribute' to the assembly (in the AssemblyInfo file) pointing to the descriptor class created in the previous step;
- Change the name of the assembly containing the export to end with the word 'Exporting' for example 'MyCustom**Exporting**.dll', 'MyCompany**Exporting**.dll' etc;
- Deploy the compiled assembly with your application.

For example of custom exporter, please refer to the sample application shipped with the Extension distribution.

Tip: One assembly can contain more than one exporter classes.

3.9. Changing exporters folder

This folder is the place where the export manager of the DataGridView Extensions looks for assemblies (like 'DGVEExcelExporting.dll') containing exporters to different file formats. By default folder is where the extension assembly resides, but it can be easily changed.

```
C#
using CompletIT.Windows.Forms.Export.Export;

DGVEExportingManager.ExportersDirectoryName = "C:\\Exporters";

VB.NET
Imports CompletIT.Windows.Forms.Export.Export

DGVEExportingManager.ExportersDirectoryName = "C:\\Exporters"
```



4. Security

DataGridView Extension has built in security permissions mechanism, which allows you to prevent changes of some editable properties in the user interface by the end-user of your application. When you restrict the access to a specific property the corresponding user control(s) will be disabled and thus the user will not be able to change it. For example you can restrict the access to the read-only property for all columns so it cannot be modified. There are two levels of permissions:

- **Default** – applied to all grid view controls and columns in your application;
- **Specific** - for a particular grid view control or a column (override the Default). Both grid view and column controls are recognized by their names (Name property).

It is important to remark that the specific permissions have higher priority than the default permissions. This means that if you explicitly restrict the access to a property in the default permissions and allow it in the specific permissions for a particular grid view or column then the user will be able to change it only for that particular grid view or column, because the specific permissions have the final word.

Note: Keep in mind that the permissions in a single application are shared and are not bound to a specific instance of the DataGridView Extension. This means that you can have several instances of the extension in your application but all of them will use the same permissions set you have defined elsewhere in your application.

4.1. Default permissions for all grid view controls in the application

You can define default permissions for the properties that can be modified in the user interface for all grid view controls in your application.

Default grid view permissions can be overridden by the permissions explicitly defined for a specific grid view control.

C#

```
using CompletIT.Windows.Forms.Permissions;
```

```
//The user will not be able to change the theme for any of the grids in your application  
DGVEPermissions.DefaultGridViewPermissions[ GridViewProperty.ChangeTheme ] = false;  
//The user will not be able to export any of the grids in your application  
DGVEPermissions.DefaultGridViewPermissions[ GridViewProperty.Export ] = false;  
//The user will not be able to search in any of the grids in your application  
DGVEPermissions.DefaultGridViewPermissions[ GridViewProperty.Find ] = false;  
//The user will not be able to print any of the grids in your application  
DGVEPermissions.DefaultGridViewPermissions[ GridViewProperty.Print ] = false;  
//The user will not be able to change the general settings in any of the grids in your application  
DGVEPermissions.DefaultGridViewPermissions[ GridViewProperty.EditGeneralSettings ] = false;  
//Set other permissions here...
```

VB.NET

```
Imports CompletIT.Windows.Forms.Permissions
```

```
'The user will not be able to change the theme for any of the grids in your application  
DGVEPermissions.DefaultGridViewPermissions( GridViewProperty.ChangeTheme ) = False  
'The user will not be able to export any of the grids in your application  
DGVEPermissions.DefaultGridViewPermissions( GridViewProperty.Export ) = False
```



```
'The user will not be able to search in any of the grids in your application
DGVEPermissions.DefaultGridViewPermissions( GridViewProperty.Find ) = False
'The user will not be able to print any of the grids in your application
DGVEPermissions.DefaultGridViewPermissions( GridViewProperty.Print ) = False
'The user will not be able to change the general settings in any of the grids in your application
DGVEPermissions.DefaultGridViewPermissions( GridViewProperty. EditGeneralSettings ) = False
'Set other permissions here...
```

4.2. Default permissions for all columns in the application

You can define default permissions for the properties that can be modified in the user interface for all columns in your application.

Default columns permissions can be overridden by the permissions explicitly defined for a specific column.

```
C#
using CompletIT.Windows.Forms.Permissions;

//The user will not be able to edit any of the extension column properties in your application
DGVEPermissions.DefaultGridViewPermissions.ColumnPermissions[ ColumnProperty.Edit ] = false;
//The user will not be able to change the read-only property for any column in your application
DGVEPermissions. DefaultGridViewPermissions.ColumnPermissions[ ColumnProperty.ChangeReadOnly ] =
false;
//The user will not be able to change the frozen property for any column in your application
DGVEPermissions. DefaultGridViewPermissions.ColumnPermissions[ ColumnProperty.ChangeFrozen ] = false;
//The user will not be able to rename any column in your application
DGVEPermissions. DefaultGridViewPermissions.ColumnPermissions[ ColumnProperty.Rename ] = false;
//Set other column permissions here...

VB.NET
Imports CompletIT.Windows.Forms.Permissions

'The user will not be able to edit any of the extension column properties in your application
DGVEPermissions. DefaultGridViewPermissions. ColumnPermissions( ColumnProperty.Edit ) = False
'The user will not be able to change the read-only property for any column in your application
DGVEPermissions. DefaultGridViewPermissions. ColumnPermissions( ColumnProperty.ChangeReadOnly ) =
False
'The user will not be able to change the frozen property for any column in your application
DGVEPermissions. DefaultGridViewPermissions. ColumnPermissions( ColumnProperty.ChangeFrozen ) = False
'The user will not be able to rename any column in your application
DGVEPermissions. DefaultGridViewPermissions. ColumnPermissions( ColumnProperty. Rename ) = False
'Set other column permissions here...
```

4.3. Permissions for a particular grid view control

You can define permissions for the properties that can be modified in the user interface for a particular grid view control.

The explicitly defined permissions for a particular grid view override the default permissions.

```
C#
using CompletIT.Windows.Forms.Permissions;

//Create permissions for a particular grid view if not existing
GridViewPermissions gridPermissions = DGVEPermissions.GridViewPermissions[ dataGridView.Name ];
if ( gridPermissions == null )
```



```
gridPermissions = DGVEPermissions.GridViewPermissions.Add( dataGridView.Name );
```

```
//The user will not be able to change the theme of the grid view  
gridPermissions[ GridViewProperty.ChangeTheme ] = false;  
//The user will not be able to export the content of the grid view  
gridPermissions[ GridViewProperty.Export ] = false;  
//The user will not be able to search in the grid view  
gridPermissions[ GridViewProperty.Find ] = false;  
//The user will not be able to print the grid view  
gridPermissions[ GridViewProperty.Print ] = false;  
//The user will not be able to change the general settings of the grid view  
gridPermissions[ GridViewProperty.EditGeneralSettings ] = false;
```

VB.NET

```
Imports CompletIT.Windows.Forms.Permissions
```

```
'Create permissions for a particular grid view if not existing  
Dim gridPermissions As GridViewPermissions = DGVEPermissions.GridViewPermissions(dataGridView.Name)  
If gridPermissions Is Nothing Then  
    gridPermissions = DGVEPermissions.GridViewPermissions.Add(dataGridView.Name)  
End If
```

```
'The user will not be able to change the theme of the grid view  
gridPermissions(GridViewProperty.ChangeTheme) = False  
'The user will not be able to export the content of the grid view  
gridPermissions(GridViewProperty.Export) = False  
'The user will not be able to search in the grid view  
gridPermissions(GridViewProperty.Find) = False  
'The user will not be able to print the grid view  
gridPermissions(GridViewProperty.Print) = False  
'The user will not be able to change the general settings of the grid view  
gridPermissions(GridViewProperty.EditGeneralSettings) = False
```

4.4. Permissions for a particular column

You can define permissions for the properties that can be modified in the user interface for a particular grid view column.

The explicitly defined permissions for a particular grid view column override the default column permissions.

C#

```
using CompletIT.Windows.Forms.Permissions;  
  
// Create permissions for a particular grid view if not existing  
GridViewPermissions gridPermissions = DGVEPermissions.GridViewPermissions[ dataGridView.Name ];  
if ( gridPermissions == null )  
    gridPermissions = DGVEPermissions.GridViewPermissions.Add( dataGridView.Name );  
// Create permissions for a particular column from the grid view if not existing  
ColumnPermissions columnPermissions = gridPermissions.Columns[ column1.Name ];  
if ( columnPermissions == null )  
    columnPermissions = gridPermissions.Columns.Add( column1.Name );  
  
//The user will not be able to change the read-only state of the column  
columnPermissions[ ColumnProperty.ChangeReadOnly ] = false;  
//The user will not be able to change the frozen state of the column  
columnPermissions[ ColumnProperty.ChangeFrozen ] = false;
```



```
//The user will not be able to rename the column  
columnPermissions[ ColumnProperty.Rename ] = false;  
//Set other column permissions here...
```

VB.NET

```
Imports CompletIT.Windows.Forms.Permissions
```

```
'Create permissions for a particular grid view if not existing  
Dim gridPermissions As GridViewPermissions = DGVEPermissions.GridViewPermissions(dataGridView.Name)  
If gridPermissions Is Nothing Then  
    gridPermissions = DGVEPermissions.GridViewPermissions.Add(dataGridView.Name)  
End If
```

```
'Create permissions for a particular grid view if not existing  
Dim columnPermissions As ColumnPermissions = gridPermissions.Columns(dataGridView.Name)  
If gridPermissions Is Nothing Then  
    gridPermissions = DGVEPermissions.GridViewPermissions.Add(dataGridView.Name)  
End If
```

```
//The user will not be able to change the read-only state of the column  
columnPermissions( ColumnProperty.ChangeReadOnly ) = False  
//The user will not be able to change the frozen state of the column  
columnPermissions( ColumnProperty.ChangeFrozen ) = False  
//The user will not be able to rename the column  
columnPermissions( ColumnProperty.Rename ) = False  
//Set other column permissions here...
```

4.5. General permissions for the extension

You can apply permissions to actions that are general for the extension and are not bound to certain grid view control i.e. export/import themes, edit settings etc. These permissions are valid for the whole application and for all grid view controls in it.

C#

```
using CompletIT.Windows.Forms.Permissions;
```

```
//The user will not be able to open the settings editor for any grid view in the application  
DGVEPermissions.ExtensionPermissions[ ExtensionProperty.OpenSettingsEditor ] = false;  
//The user will not be able to export themes  
DGVEPermissions.ExtensionPermissions[ ColumnProperty.ExportThemes ] = false;  
//The user will not be able to import themes  
DGVEPermissions.ExtensionPermissions[ ColumnProperty.ImportThemes ] = false;  
//Set other extension permissions here...
```

VB.NET

```
Imports CompletIT.Windows.Forms.Permissions
```

```
'The user will not be able to open the settings editor for any grid view in the application  
DGVEPermissions.ExtensionPermissions( ExtensionProperty.OpenSettingsEditor ) = False  
'The user will not be able to export themes  
DGVEPermissions.ExtensionPermissions( ColumnProperty.ExportThemes ) = False  
'The user will not be able to import themes  
DGVEPermissions.ExtensionPermissions( ColumnProperty.ImportThemes ) = False  
//Set other extension permissions here...
```



5. Find

5.1. How to find specific cell value

Add reference to the main assembly 'DataGridViewExtension.dll'.

```
C#
using CompletIT.Windows.DataGridViewExtension;

//Create text search settings
DGVEFindAndReplaceTextSettings settings = new DGVEFindAndReplaceTextSettings();
settings.FindWhat = "SearchForText";
//Start searching from current cell
DataGridViewCell foundCell = DGVEFindAndReplaceEngine.Find(dataGridView, dataGridView.CurrentCell,
    settings);
if ( foundCell!= null )
    dataGridView.CurrentCell = foundCell;
else
    MessageBox.Show( "The searched item cannot be found." );

VB.NET
Imports CompletIT.Windows.DataGridViewExtension

'Create text search settings
Dim Settings As DGVEFindAndReplaceTextSettings = New DGVEFindAndReplaceTextSettings()
Settings.FindWhat = "SearchForText"
'Start searching from current cell
Dim FoundCell As DataGridViewCell = DGVEFindAndReplaceEngine.Find( Me.DataGridViewControl, _
    Me.DataGridViewControl.CurrentCell, Settings )
If FoundCell IsNot Nothing Then
    Me.DataGridViewControl.CurrentCell = FoundCell
Else
    MessageBox.Show( "The searched item cannot be found." )
End If
```

5.2. How to find a value using the find dialog

Add reference to the main assembly 'DataGridViewExtension.dll'.

```
C#
using CompletIT.Windows.DataGridViewExtension;

//Note! You can show the find and replace
//dialog only for already managed grid views.
DataGridViewExtensionComponent.ExtensionUI.ShowFindAndReplaceEditor( dataGridView );

VB.NET
Imports CompletIT.Windows.DataGridViewExtension

//Note! You can show the find and replace
//dialog only for already managed grid views.
DataGridViewExtensionComponent.ExtensionUI.ShowFindAndReplaceEditor( Me.DataGridViewControl )
```

6. Print

The print engine of the DataGridView Extension is developed in a way which allows it's usage as a separate component.



Note: If you want to print the content of a DataGridView control, you do not have to manage/extend it; just pass it as a parameter to the appropriate print method of the print engine.

6.1. Silent print (No print dialog)

Add reference to the main assembly '*DataGridViewExtension.dll*'.

C#

```
using CompletIT.Windows.Forms.Printing;  
  
//Note! The first parameter is the grid which has to be printed;  
//the second parameter indicates whether the print dialog should be shown.  
DGVEPrintManager.Print( dataGridView, false );
```

VB.NET

```
Imports CompletIT.Windows.Forms.Printing  
  
'Note! The first parameter is the grid which has to be printed;  
'the second parameter indicates whether the print dialog should be shown.  
DGVEPrintManager.Print( Me.DataGridViewControl, false )
```

6.2. Print using the print dialog

Exactly like the export settings editors the print has print settings editor form named '*CompletIT.Windows.Forms.Printing.DGVEPrintSettingsEditorForm*'. In general you will not need to use it directly but in case you have to do so you can use it as a regular form or dialog.

Add reference to the main assembly '*DataGridViewExtension.dll*'.

C#

```
using CompletIT.Windows.Forms.Printing;  
  
//Note! The first parameter is the grid which has to be printed;  
//the second parameter indicates whether the print dialog should be shown.  
DGVEPrintManager.Print( dataGridView, true );
```

VB.NET

```
Imports CompletIT.Windows.Forms.Printing  
  
'Note! The first parameter is the grid which has to be printed;  
'the second parameter indicates whether the print dialog should be shown.  
DGVEPrintManager.Print( Me.DataGridViewControl, true )
```

6.3. Print preview

Add reference to the main assembly '*DataGridViewExtension.dll*'.

C#

```
using CompletIT.Windows.Forms.Printing;  
  
//Note! The first parameter is the grid to be printed;  
//the second parameter indicates whether the print dialog should be shown.  
DGVEPrintManager.PrintPreview( dataGridView );
```

**VB.NET**

Imports CompletIT.Windows.Forms.Printing

'Note! The first parameter is the grid which has to be printed;
'the second parameter indicates whether the print dialog should be shown.
DGVEPrintManager.PrintPreview ([Me](#).DataGridViewControl)

7. FAQ

7.1. Which component does the DataGridView Extension extends?

The component extended by the DataGridView Extension is the standard DataGridView provided in .NET 2.0 Framework.

7.2. Does DataGridView Extension component inherit the standard DataGridView control?

No, DataGridView Extension component just decorates the standard DataGridView control provided in .NET 2.0 Framework without any inheritance and with just single line of code.

7.3. In what programming language is DataGridView Extension written?

DataGridView Extension is written entirely in C# but you can use it in any project written in any language supported by .NET framework.

7.4. Which version of .NET framework does DataGridView Extension support?

DataGridView Extension works under .NET 2.0 and is compatible with Visual Studio.NET 2005.

7.5. Does DataGridView Extension have design time support?

Yes, it has design time support. Its behavior is exactly like using ToolTip component. You just have to drag & drop DataGridView Extension component from the VS.NET 2005 Toolbox on your root container then select the grid you wish to manage and set the value of its property 'IsManagedByExtension' to true.

7.6. What are the requirements in order to have a DataGridView control managed by the DataGridView Extension?

There are two things you have to ensure: the first is a valid Name set to the DataGridView control and the second is permission to write the Xml files to the hard drive.

7.7. To which assemblies do I have to add references in my project?

In order to use DataGridView Extension in your project you have to add references to the main assembly 'DataGridViewExtension.dll', Xml persistence optimizer 'DataGridViewExtension.XmlSerializers.dll' and all other export DLLs like 'DGVEExcelExporting.dll', 'DGVEHtmlExporting.dll' etc. you wish to be supported by your application.

7.8. How is DataGridView Extension integrated?

DataGridView Extension integrates itself by adding extension button on the top left corner of the grid. It also integrates as sub-menu item in the DataGridView control's context menu. In case the underlying control does not have context menu, the extension sets its own menu as context menu.



7.9. How DataGridView Extension stores its data?

All of the data is stored entirely in Xml format and is ready for further processing.

7.10. Where is all Xml data stored on the user's computer?

By default all Xml files are stored in the following folder 'C:\Documents and Settings\{Username}\Application Data\{CompanyName}\{ApplicationName}\{ApplicationVersion}\' but it can be easily changed programmatically or runtime to any other location - see the [example](#) in [Common](#) section.

7.11. What is the structure of the persistence folder?

All changes made by the user are persisted in the following structure:

- **'Themes'** – Contains all themes for the application;
- **'NotSavedSettings'** – Changes to grid view controls settings that are still not saved as themes;
- **'DataSourceSpecificSettings'** – Saved settings related to specific data source such as columns;
- **'CurrentSettingsLocation'** – Contains the location of the current settings file (theme's related + data source related) that is applied to each managed grid view control.

7.12. Does DataGridView Extension support custom columns?

Yes, the DataGridView Extension supports custom columns. The only requirement for these columns is to have default constructor without parameters.

7.13. How does the export functionality work?

The export functionality of DataGridView Extension is plug-in based. On application startup the export manager looks for DLLs with names ending with *'...Exporting.dll'* and marked with a specific attribute and displays them in the Export list. In case you want to add or remove export to specific file format you just have to add or remove the appropriate DLL from the working folder. For example if you do not want to have export to MS Excel, remove the appropriate assembly *'DGVEExcelExporting.dll'*.

7.14. Which version of MS Excel is supported by the export?

DataGridView Extension can export to MS Excel XP and later versions. It is using the PIAs for MS Office XP.

7.15. The export to Excel does not work what should I do?

The export to MS Excel is COM based and it requires the following PIAs: *'Microsoft.Office.Interop.Excel.dll'*, *'Microsoft.Vbe.Interop.dll'* and *'Office.dll'*.

Check whether they are registered in your GAC (Global Assembly Cache) or exist in the folder where all assemblies of the Extension reside.

If you have to register the PIAs in your GAC use the register files from *'<InstallationPath>\v1.1\PIAs'* shipped with the distribution of the DataGridView Extension.



7.16. How to add DataGridView Extension to my VS.NET 2005 Toolbox?

You can manually add the DataGridView Extension to your IDE toolbox by simply drag-and-drop the control DLL '*DataGridViewExtension.dll*' from Windows explorer to your VS toolbox.

7.17. How many DataGridView controls can be managed by a single instance of the Extension?

One instance of the DataGridView Extension can manage as many DataGridView controls as you want. There is no memory overhead if you create separate instance for each DataGridView control in your application.

7.18. How many instance of the extension can I have in single .NET application?

You can have as many instances of the DataGridView Extension component as you want within an application, but remember that a DataGridView control should be managed by only one instance of the Extension.

7.19. Why the colors of my column and row headers are not applied although I have changed them?

If you want to change the colors of the row and column headers you have to ensure that the option '*Use Windows Visual Styles*' is not checked in the Header Styles editor - by default this option is checked which means that all headers are using the default colors defined by the system.