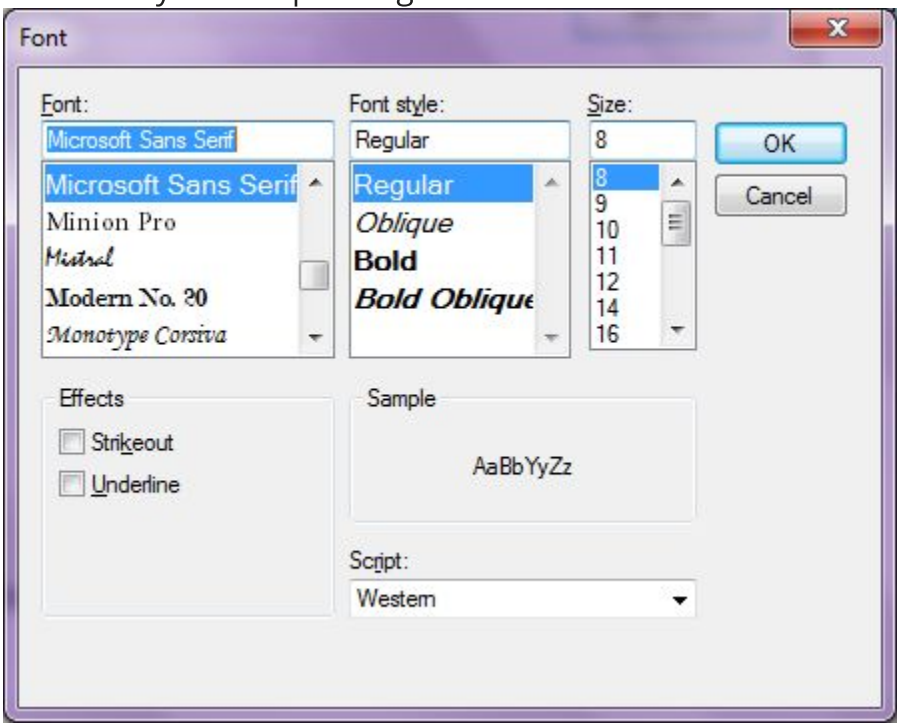
Prog 9: Introduction to Dialog control and their properties

a. **Font Dialog**
b. **Open File Dialog**
c. **Save File Dialog**

A **FontDialog** control in WinForms is used to select a font from available fonts installed on a system.

A typical Font Dialog looks like Figure 1 where you can see there is a list of fonts, styles, size and other options. Please note a FontDialog may have different fonts on different system depending on what fonts are installed on a system.



FontDialog

We can create a FontDialog control using a Forms designer at design-time or using the FontDialog class in code at run-time (also known as dynamically). Unlike other Windows Forms controls, a FontDialog does not have and not need visual properties like others. You use a FontDialog to list all the fonts and select one of them and usually apply selected fonts on controls or some contents.

Note

Even though you can create a FontDialog at design-time as well as at run-time, I recommend using run-time method.

Design-time

To create a FontDialog control at design-time, you simply drag and drop a FontDialog control from Toolbox to a Form in Visual Studio. After you drag and drop a FontDialog on a Form, the FontDialog looks like Figure 2.

Adding a FontDialog to a Form adds following two lines of code.

```
1. private System.Windows.Forms.FontDialog fontDialog1;
2. this.fontDialog1 = new System.Windows.Forms.FontDialog();
```

Run-time

Creating a FontDialog control at run-time is merely a work of creating an instance of FontDialog class, setting its properties and adding FontDialog class to the Form controls.

First step to create a dynamic FontDialog is to create an instance of FontDialog class.

The following code snippet creates a FontDialog control object.

```
1. FontDialog fontDlg = new FontDialog();
```

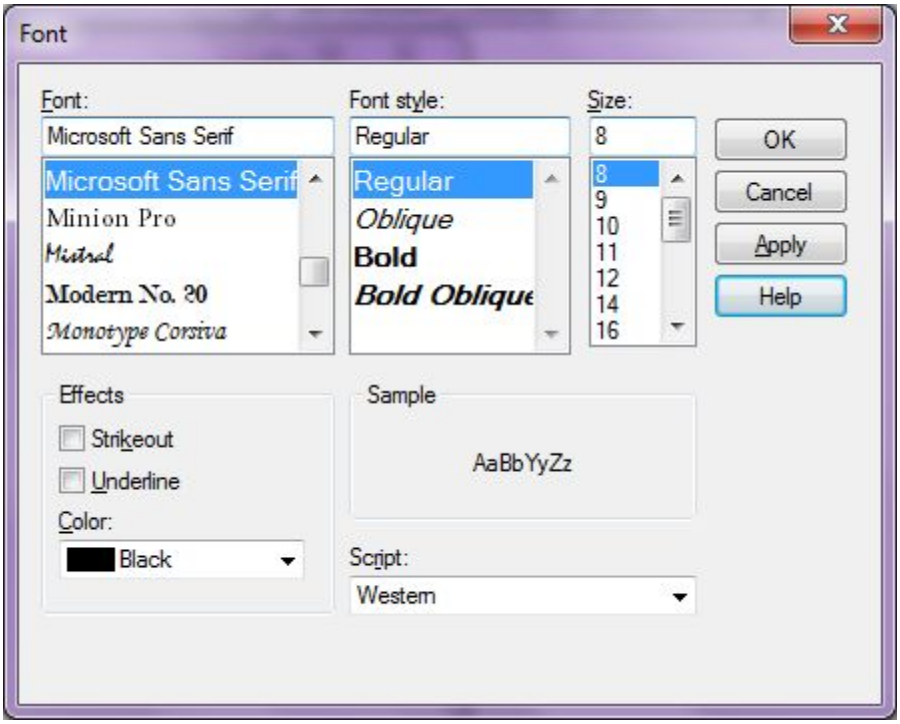
ShowDialog method of FontDialog displays the FontDialog. The following code snippet calls the ShowDialog method.

```
1. fontDlg.ShowDialog();
```

Once the ShowDialog method is called, you can pick a font on the dialog.

FontDialog Properties - Show Properties

A FontDialog control can have a color drop down that allows users to select a color of the selected font. A FontDialog can also have Apply and Help buttons as you can see in Figure 3.



ShowColor, ShowApply, and ShowHelp properties can be set to true to show these options. The ShowEffects property represents whether the dialog box contains controls that allow the user to specify strikethrough, underline, and text color options.

The following code snippet sets these properties to true.

```
1. fontDlg.ShowColor = true;
2. fontDlg.ShowApply = true;
3. fontDlg.ShowEffects = true;
4. fontDlg.ShowHelp = true;
```

Maximum and Minimum Font Size

We can restrict the maximum and minimum font size by setting MaxSize and MinSize properties. The following code snippet sets these properties to 40 and 22 respectively.

```
1. fontDlg.MaxSize = 40;
2. fontDlg.MinSize = 22;
```

Font and Color

The Font and Color properties represent the selected font and color in a FontDialog. The following code snippet gets these properties and sets font and color of a TextBox and a Label controls.

```
if (fontDlg.ShowDialog() != DialogResult.Cancel) {
    textBox1.Font = fontDlg.Font;
    label1.Font = fontDlg.Font;
    textBox1.BackColor = fontDlg.Color;
    label1.ForeColor = fontDlg.Color;
}
```

Summary

A FontDialog control allows users to launch Windows Font Dialog and let them select a font and font color. In this article, we discussed how to use a Windows Font Dialog and set its properties in a Windows Forms application.

C# **OpenFileDialog**

C# OpenFileDialog control allows us to browse and select files on a computer in an application. A typical Open File Dialog looks like Figure 1 where you can see Windows Explorer like features to navigate through folders and select a file.

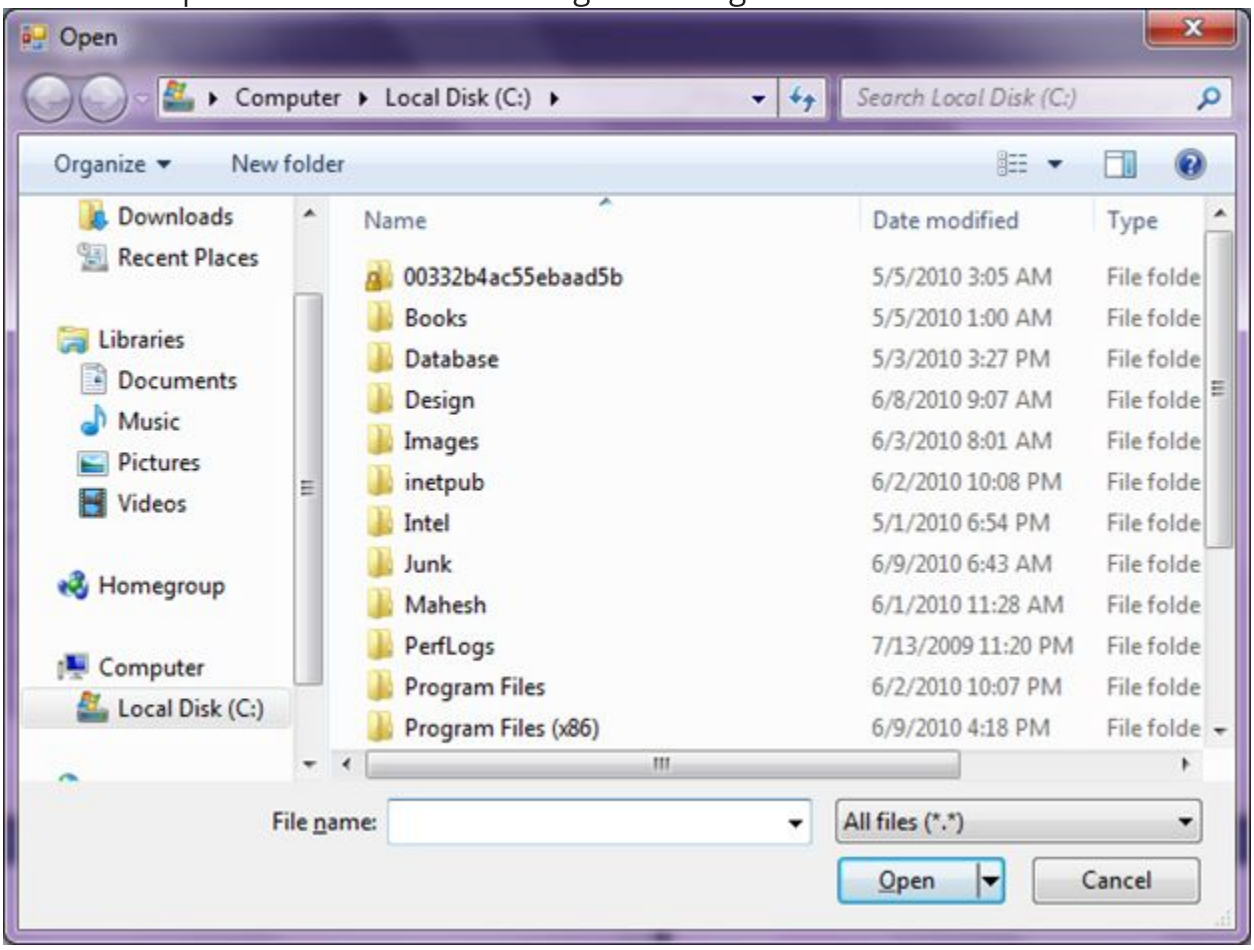


Figure 1

Creating a OpenFileDialog

We can create an OpenFileDialog control using a Forms designer at design-time or using the OpenFileDialog class in code at run-time (also known as dynamically). Unlike other Windows Forms controls, an OpenFileDialog does not have and not need visual properties like others. The only purpose of OpenFileDialog to display available colors, create custom colors and select a color from these colors. Once a color is selected, we need that color in our code so we can apply it on other controls.

Again, you can create an OpenFileDialog at design-time but it is easier to create an OpenFileDialog at run-time.

Design-time

To create an OpenFileDialog control at design-time, you simply drag and drop an OpenFileDialog control from Toolbox to a Form in Visual Studio. After you drag and drop an OpenFileDialog on a Form, the OpenFileDialog looks like Figure 2.

Figure 2

Adding an OpenFileDialog to a Form adds following two lines of code.

```
1.
   private System.Windows.Forms.OpenFileDialog openFileDialog1;

2.
   this.openFileDialog1 = new System.Windows.Forms.OpenFileDial
   og();
```

Run-time

Creating a OpenFileDialog control at run-time is merely a work of creating an instance of OpenFileDialog class, set its properties and add OpenFileDialog class to the Form controls.

First step to create a dynamic OpenFileDialog is to create an instance of OpenFileDialog class. The following code snippet creates an OpenFileDialog control object.

```
1. OpenFileDialog openFileDialog1 = new OpenFileDialog();
```

ShowDialog method displays the OpenFileDialog.

```
1. openFileDialog1.ShowDialog();
```

Once the ShowDialog method is called, you can browse and select a file.

Setting OpenFileDialog Properties

After you place an OpenFileDialog control on a Form, the next step is to set properties.

The easiest way to set properties is from the Properties Window. You can open Properties window by pressing F4 or right click on a control and select Properties menu item. The Properties window looks like Figure 3.

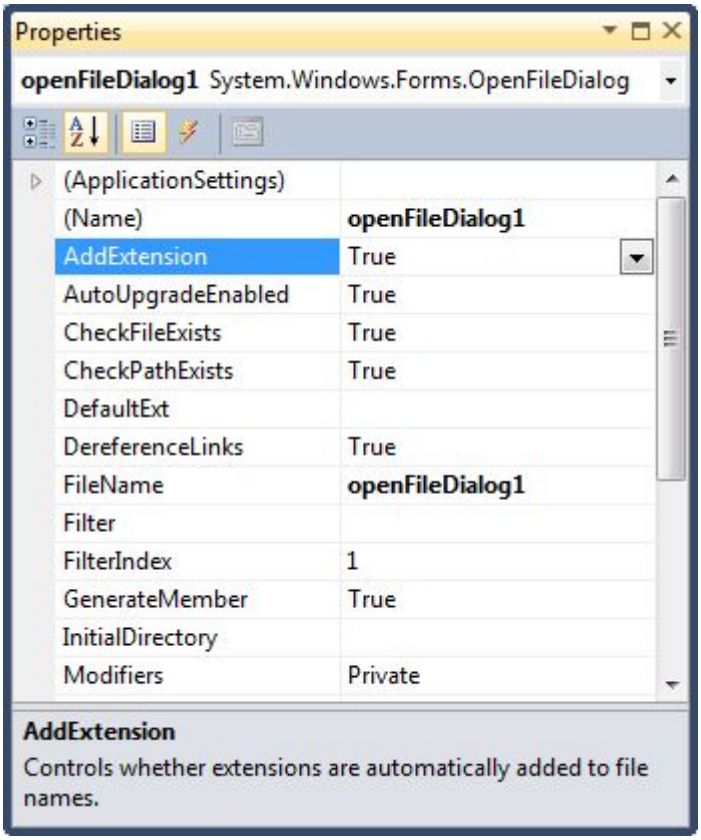


Figure 3

Initial and Restore Directories

InitialDirectory property represents the directory to be displayed when the open file dialog appears first time.

```
1. openFileDialog1.InitialDirectory = @"C:\";
```

If RestoreDirectory property set to true that means the open file dialog box restores the current directory before closing.

```
1. openFileDialog1.RestoreDirectory = true;
```

Title

Title property is used to set or get the title of the open file dialog.

```
1. openFileDialog1.Title = "Browse Text Files";
```

Default Extension

DefaultExtn property represents the default file name extension.

```
openFileDialog1.DefaultExt = "txt";
```

Filter and Filter Index

Filter property represents the filter on an open file dialog that is used to filter the type of files to be loaded during the browse option in an open file dialog. For example, if you need users to restrict to image files only, we can set Filter property to load image files only.

```
openFileDialog1.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*";
```

FilterIndex property represents the index of the filter currently selected in the file dialog box.

```
openFileDialog1.FilterIndex = 2;
```

Check File Exists and Check Path Exists

CheckFileExists property indicates whether the dialog box displays a warning if the user specifies a file name that does not exist. CheckPathExists property indicates whether the dialog box displays a warning if the user specifies a path that does not exist.

```
openFileDialog1.CheckFileExists = true;
openFileDialog1.CheckPathExists = true;
```

File Name and File Names

FileName property represents the file name selected in the open file dialog.

```
textBox1.Text = openFileDialog1.FileName;
```

If MultiSelect property is set to true that means the open file dialog box allows multiple file selection. The FileNames property represents all the files selected in the selection.

```
this.openFileDialog1.Multiselect = true;
foreach (String file in openFileDialog1.FileNames)
{
  MessageBox.Show(file);
}
```

Read Only Checked and Show Read Only Files

ReadOnlyChecked property represents whether the read-only checkbox is selected and ShowReadOnly property represents whether the read-only checkbox is available or not.

```
openFileDialog1.ReadOnlyChecked = true;
openFileDialog1.ShowReadOnly = true;
```

Implementing OpenFileDialog in a C# and WinForms Applications

Now let's create a WinForms application that will use an OpenFileDialog that has two Button controls, a TextBox, and a container control. The Form looks like Figure 4.

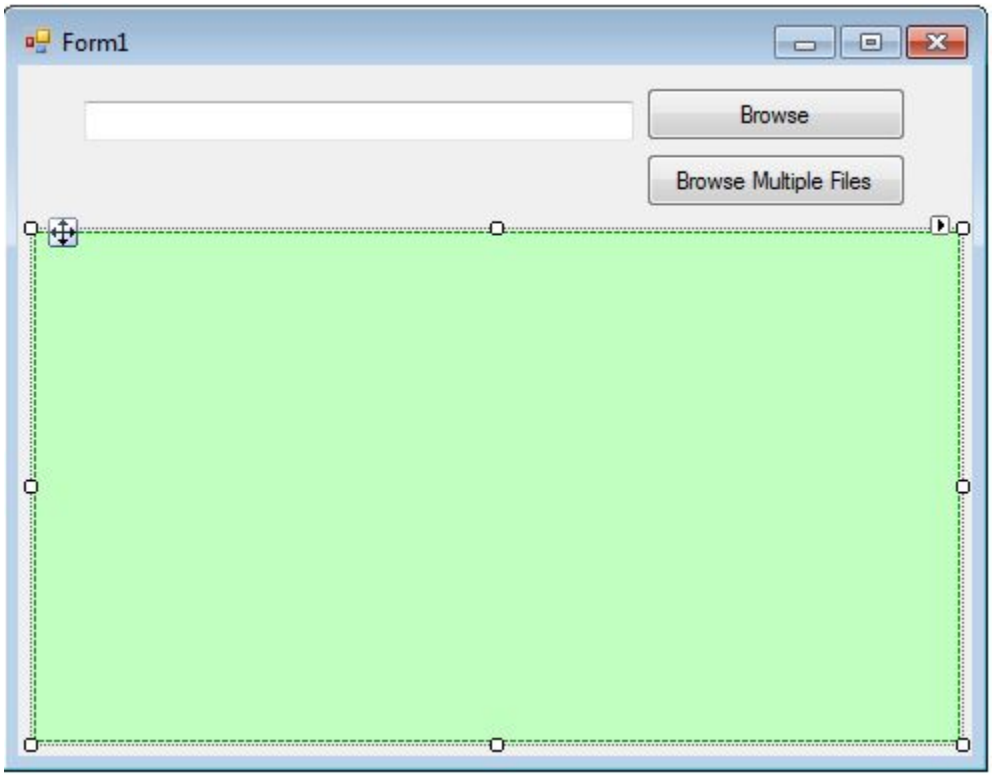


Figure 4

The Browse button click event handler will show an open file dialog and users will be able to select text files. The open file dialog looks like Figure 5.

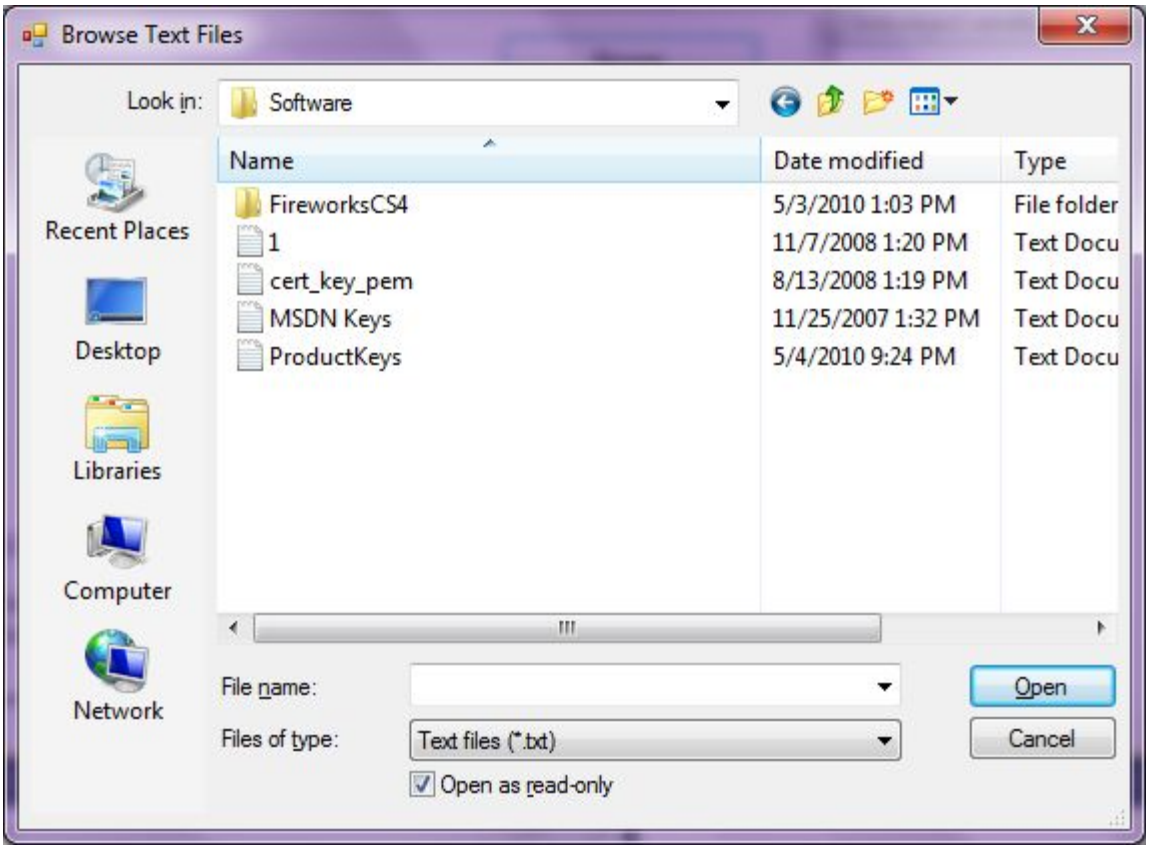


Figure 5

The following code snippet is the code for Browse button click event handler. Once a text file is selected, the name of the text file is displayed in the TextBox.

```
1.  private void BrowseButton_Click(object sender, EventArgs e)
2.  {
3.      OpenFileDialog openFileDialog1 = new OpenFileDialog
4.      {
5.          InitialDirectory = @"D:\",
6.          Title = "Browse Text Files",
7.
8.          CheckFileExists = true,
9.          CheckPathExists = true,
10.
11.         DefaultExt = "txt",
12.         Filter = "txt files (*.txt)|*.txt",
13.         FilterIndex = 2,
14.         RestoreDirectory = true,
15.
16.         ReadOnlyChecked = true,
```

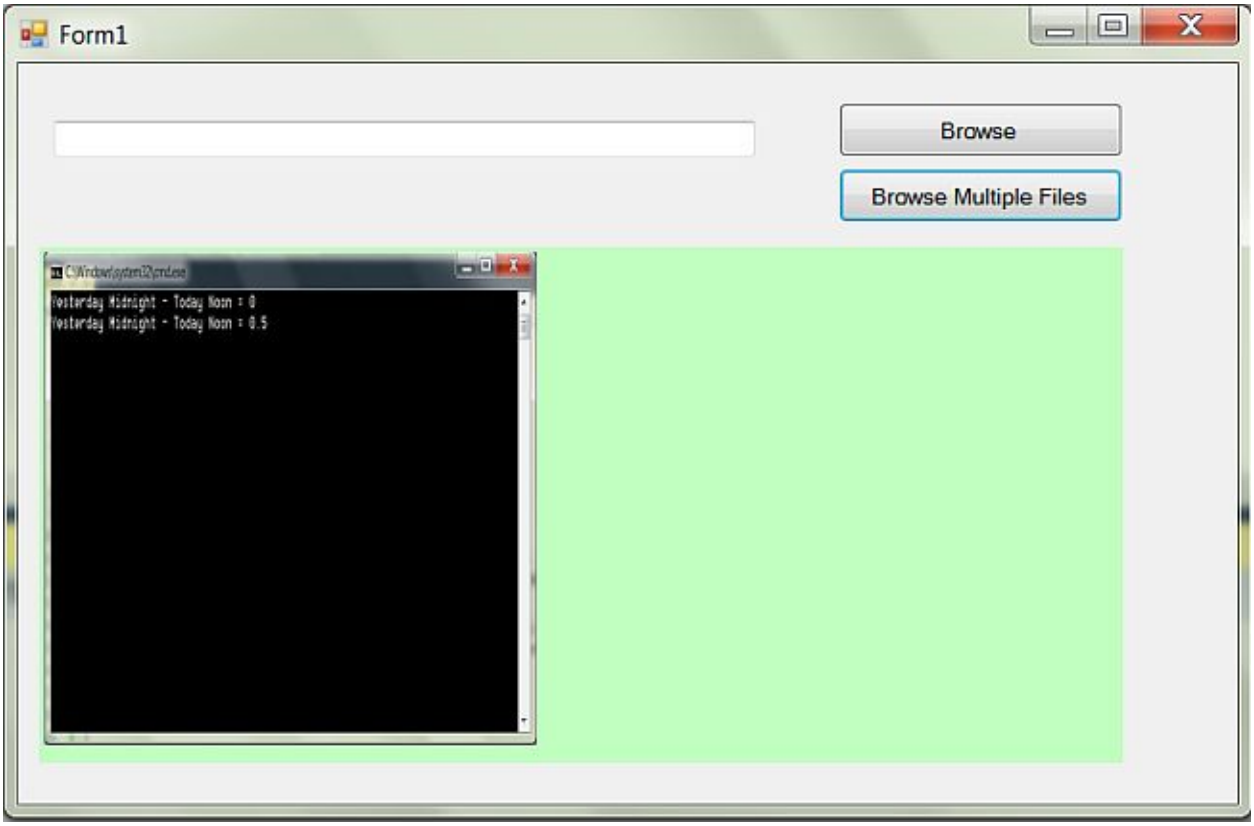
```
        ShowReadOnly = true
    };

    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        textBox1.Text = openFileDialog1.FileName;
    }
}
```

The code for Browse Multiple Files button click event handler looks like following.

```
private void BrowseMultipleButton_Click(object sender, EventArgs e)
{
    this.openFileDialog1.Filter =
"Images (*.BMP;*.JPG;*.GIF,*.PNG,*.TIFF)|*.BMP;*.JPG;*.GIF;*.PNG;*.TIFF|" +
"All files (*.*)|*.*";

    this.openFileDialog1.Multiselect = true;
    this.openFileDialog1.Title = "Select Photos";

    DialogResult dr = this.openFileDialog1.ShowDialog();
    if (dr == System.Windows.Forms.DialogResult.OK)
    {
        foreach (String file in openFileDialog1.FileNames)
        {
            try
            {
                PictureBox imageControl = new PictureBox();

                imageControl.Height = 400;
                imageControl.Width = 400;

                Image.GetThumbnailImageAbort myCallback =
                        new Image.GetThumbnailImageAbort(ThumbnailCallback);
                Bitmap myBitmap = new Bitmap(file);

                Image myThumbnail = myBitmap.GetThumbnailImage(300, 300,
                    myCallback, IntPtr.Zero);
                imageControl.Image = myThumbnail;
```

```
                PhotoGallary.Controls.Add(imageControl);
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
            }
        }
    }
}
public bool ThumbnailCallback()
{
    return false;
}
```

The output looks like Figure 6.



Summary

An OpenFileDialog control allows users to launch Windows Open File Dialog and let them select files. In this article, we discussed how to use a Windows Open File Dialog and set its properties in a Windows Forms application.

SaveFileDialog In C#

A SaveFileDialog control is used to save a file using Windows SaveFileDialog. A typical SaveFileDialog looks like Figure 1 where you can see the Windows Explorer type features to navigate through folders and save a file in a folder.

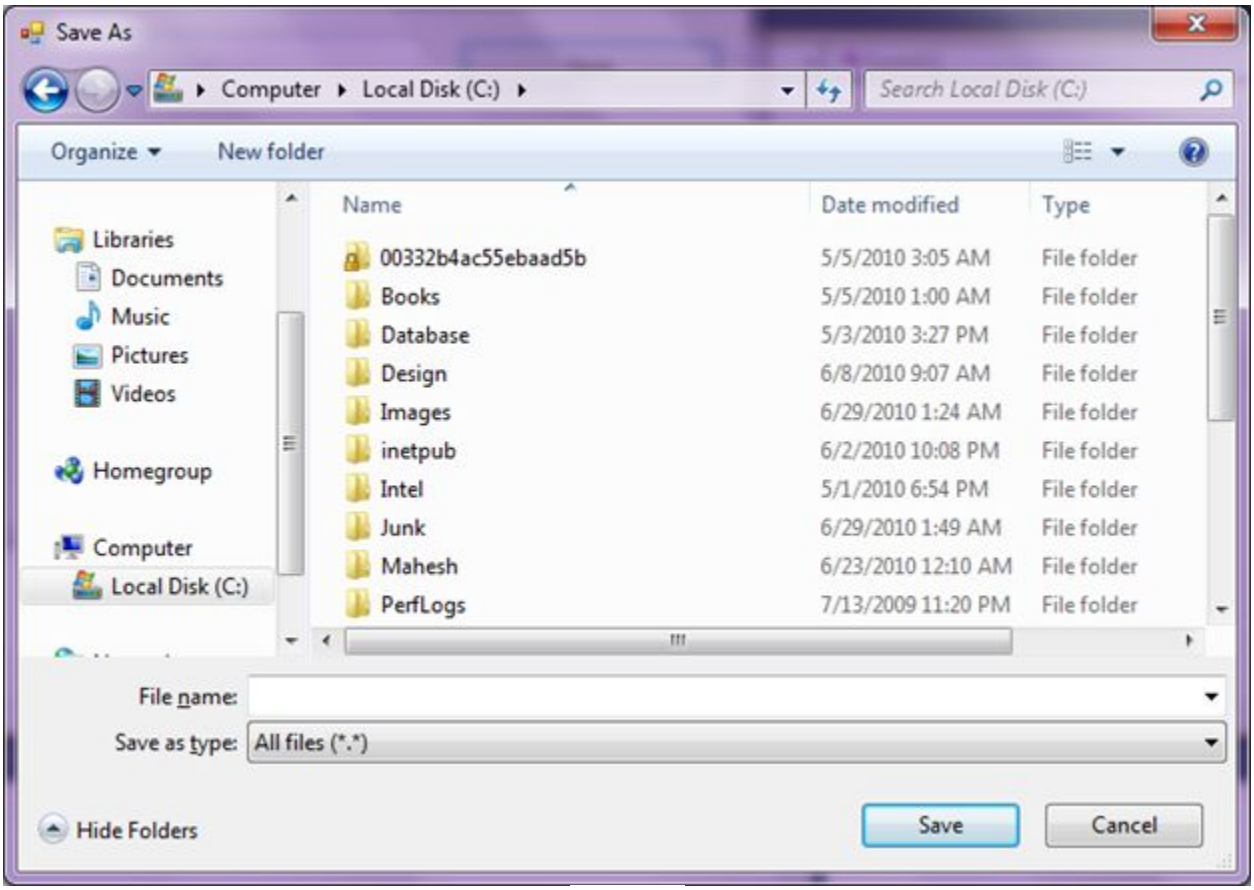


Figure 1

Creating a SaveFileDialog

We can create a SaveFileDialog control using a Forms designer at design-time or using the SaveFileDialog class in code at run-time (also known as dynamically). Unlike other Windows Forms controls, a SaveFileDialog does not have and not need visual properties like others.

Note
Even though you can create a SaveFileDialog at design-time it is easier to create a SaveFileDialog at run-time.

Design-time

To create a SaveFileDialog control at design-time, you simply drag and drop a SaveFileDialog control from Toolbox to a Form in Visual Studio. After you drag and drop a SaveFileDialog on a Form, the SaveFileDialog looks like Figure 2.

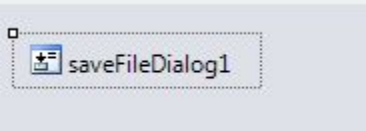


Figure 2

Adding a SaveFileDialog to a Form adds the following two lines of code.

```
1.
   private System.Windows.Forms.SaveFileDialog saveFileDialog1;
2.
   this.saveFileDialog1 = new System.Windows.Forms.SaveFileDial
   og();
```

Run-time

Creating a SaveFileDialog control at run-time is merely a work of creating an instance of SaveFileDialog class, setting its properties and adding SaveFileDialog class to the Form controls.

The first step to create a dynamic SaveFileDialog is to create an instance of SaveFileDialog class. The following code snippet creates a SaveFileDialog control object.

```
1. SaveFileDialog SaveFileDialog1 = new SaveFileDialog();
```

ShowDialog method displays the SaveFileDialog.

```
1. SaveFileDialog1.ShowDialog();
```

Once the ShowDialog method is called, you can browse and select a file.

Setting SaveFileDialog Properties

After you place a SaveFileDialog control on a Form, the next step is to set properties.

The easiest way to set properties is from the Properties Window. You can open Properties window by pressing F4 or right clicking on a control and selecting Properties menu item. The Properties window looks like Figure 3.

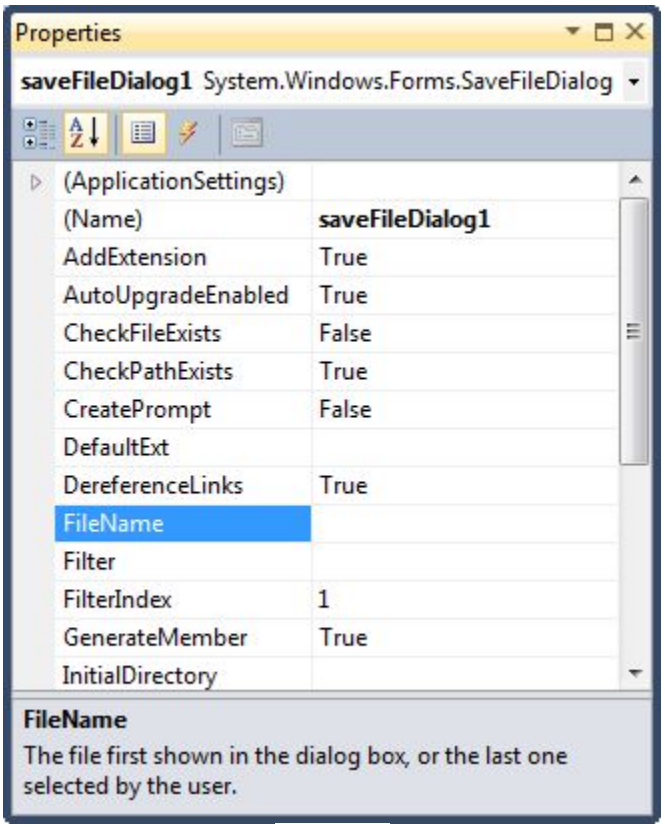


Figure 3

Initial and Restore Directories

InitialDirectory property represents the directory to be displayed when the open file dialog appears the first time.

```
1. SaveFileDialog1.InitialDirectory = @"C:\";
```

If RestoreDirectory property is set to true that means the open file dialog box restores the current directory before closing.

```
1. SaveFileDialog1.RestoreDirectory = true;
```

Title

Title property is used to set or get the title of the open file dialog.

```
SaveFileDialog1.Title = "Browse Text Files";
```

Default Extension

DefaultExtn property represents the default file name extension.

```
SaveFileDialog1.DefaultExt = "txt";
```

Filter and Filter Index

Filter property represents the filter on an open file dialog that is used to filter the type of files to be loaded during the browse option in an open file dialog. For example, if you need users to restrict to image files only, we can set Filter property to load image files only.

```
SaveFileDialog1.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*";
```

FilterIndex property represents the index of the filter currently selected in the file dialog box.

```
SaveFileDialog1.FilterIndex = 2;
```

Check File Exists and Check Path Exists

CheckFileExists property indicates whether the dialog box displays a warning if the user specifies a file name that does not exist. CheckPathExists property indicates whether the dialog box displays a warning if the user specifies a path that does not exist.

```
SaveFileDialog1.CheckFileExists = true;
SaveFileDialog1.CheckPathExists = true;
```

File Name and File Names

FileName property represents the file name selected in the open file dialog.

```
textBox1.Text = SaveFileDialog1.FileName;
```

If MultiSelect property is set to true that means the open file dialog box allows multiple file selection. The FileNames property represents all the files selected in the selection.

```
this.SaveFileDialog1.Multiselect = true;
foreach(String file in SaveFileDialog1.FileNames) {
    MessageBox.Show(file);
}
```

Sample Example

The following code snippet is the code for Save button click event handler. Once a text file is selected, the name of the text file is displayed in the TextBox.

```
private void SaveButton_Click(object sender, EventArgs e) {

    SaveFileDialog saveFileDialog1 = new SaveFileDialog();

    saveFileDialog1.InitialDirectory = @ "C:\";
    saveFileDialog1.Title = "Save text Files";
    saveFileDialog1.CheckFileExists = true;
    saveFileDialog1.CheckPathExists = true;
    saveFileDialog1.DefaultExt = "txt";

    saveFileDialog1.Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";
    saveFileDialog1.FilterIndex = 2;
    saveFileDialog1.RestoreDirectory = true;

    if (saveFileDialog1.ShowDialog() == DialogResult.OK) {

        textBox1.Text = saveFileDialog1.FileName;
    }
}
```

Summary

A SaveFileDialog control allows users to launch Windows Save File Dialog and let them save files. In this article, we discussed how to use a Windows Save File Dialog and set its properties in a Windows Forms application.