Windows Forms

.NET Framework (current version)

As forms are the base unit of your application, it is essential that you give some thought to their function and design. A form is ultimately a blank slate that you, as a developer, enhance with controls to create a user interface and with code to manipulate data. To that end, Visual Studio provides you with an integrated development environment (IDE) to aid in writing code, as well as a rich control set written with the .NET Framework. By complementing the functionality of these controls with your code, you can easily and quickly develop the solutions you need.

In This Section

Getting Started with Windows Forms
: Provides links to topics about how to harness the power of Windows Forms to display data, handle user input, and deploy your applications easily and with more robust security.

Enhancing Windows Forms Applications
: Provides links to topics about how to enhance your Windows Forms with a variety of features.

Related Sections

Windows Forms Controls
: Contains links to topics that describe Windows Forms controls and show how to implement them.

Windows Forms Data Binding
: Contains links to topics that describe the Windows Forms data-binding architecture.

Graphics Overview (Windows Forms)
: Discusses how to create graphics, draw text, and manipulate graphical images as objects using the advanced implementation of the Windows graphics design interface.

ClickOnce Security and Deployment
: Discusses the principles of ClickOnce deployment.

C3E8F7CF-456B-41DF-B5A5-E0370039E525
: Describes how forms have changed in the new version.

Windows Forms/MFC Programming Differences
: Discusses the differences between MFC applications and Windows Forms.

Accessing Data in Visual Studio
: Discusses incorporating data access functionality into your applications.

Debugging Preparation: Windows Forms Applications
: Discusses the process of debugging applications created with the Windows Application project template, as well as how to change the Debug and Release configurations.

Deploying Applications, Services, and Components

Describes the process by which you distribute a finished application or component to be installed on other computers.

Building Console Applications in the .NET Framework

Describes the basics of creating a console application using the Console class.

Enhancing Windows Forms Applications

.NET Framework (current version)

Windows Forms contains many features that you can use to enhance your Windows-based applications to meet the specific needs of your users. The following topics describe these features and how to use them.

In This Section

Graphics and Drawing in Windows Forms
Contains links to topics that describe and show how to use the graphics interface in Windows Forms.

Application Settings for Windows Forms.
Contains links to topics that describe and show how to use the **Application Settings** feature.

Windows Forms Print Support
Contains links to topics that describe and show how to print files from Windows Forms applications.

Drag-and-Drop Operations and Clipboard Support
Contains links to topics that describe and show how to use the drag-and-drop feature and the Clipboard in Windows Forms.

Networking in Windows Forms Applications
Contains links to topics that describe and show how to use networking in Windows Forms.

Globalizing Windows Forms
Contains links to topics that show how to globalize Windows Forms applications.

Windows Forms and Unmanaged Applications
Contains links to topics that describe and show how to access COM components from Windows Form applications.

System Information and Windows Forms
Describes how to use system information in Windows Forms.

Power Management in Windows Forms
Describes how to manage power use in Windows Forms applications.

Windows Forms Visual Inheritance
Describes how to inherit from a base form.

Multiple-Document Interface (MDI) Applications
Describes how to create multiple-document interface (MDI) applications.

Integrating User Help in Windows Forms
Describes how to integrate user help in your applications.

Windows Forms Accessibility
Describes how to make your applications available to a wide variety of users.

Using WPF Controls

Describes how to use WPF controls in your Windows Forms-based applications.

Related Sections

Help Systems in Windows Forms Applications

Contains links to topics that describe and show how to provide user help in Windows Forms applications.

Getting Started with Windows Forms

Contains links to topics that describe how to use the basic features of Windows Forms.

Application Settings for Windows Forms

.NET Framework (current version)

The Applications Settings feature of Windows Forms makes it easy to create, store, and maintain custom application and user preferences on the client. With Application Settings, you can store not only application data such as database connection strings, but also user-specific data, such as toolbar positions and most-recently used lists.

In This Section

Application Settings Overview
: Discusses how to create and store settings data on behalf of your application and your users.

Application Settings Architecture
: Describes how the Application Settings feature works, and explores advanced features of the architecture such as grouped settings and settings keys.

Application Settings Attributes
: Lists and describes the attributes that can be applied to an application settings wrapper class or its settings properties.

Application Settings for Custom Controls
: Discusses what must be done to give your custom controls the ability to persist application settings when hosted in third-party applications.

How to: Create Application Settings
: Demonstrates creating new application settings that are persisted between application sessions.

How to: Validate Application Settings
: Demonstrates validating application settings before they are persisted.

See also:

- Windows Forms

Graphics and Drawing in Windows Forms

.NET Framework (current version)

The common language runtime uses an advanced implementation of the Windows Graphics Device Interface (GDI) called GDI+. With GDI+ you can create graphics, draw text, and manipulate graphical images as objects. GDI+ is designed to offer performance and ease of use. You can use GDI+ to render graphical images on Windows Forms and controls. Although you cannot use GDI+ directly on Web Forms, you can display graphical images through the Image Web Server control.

In this section, you will find topics that introduce the fundamentals of GDI+ programming. Although not intended to be a comprehensive reference, this section includes information about the Graphics, Pen, Brush, and Color objects, and explains how to perform such tasks as drawing shapes, drawing text, or displaying images. For more information, see "GDI+ Reference" in the MSDN library at http://msdn.microsoft.com/library.

In This Section

Graphics Overview (Windows Forms)
: Provides an introduction to the graphics-related managed classes.

About GDI+ Managed Code
: Provides information about the managed GDI+ classes.

Using Managed Graphics Classes
: Demonstrates how to complete a variety of tasks using the GDI+ managed classes.

Reference

System.Drawing
: Provides access to GDI+ basic graphics functionality.

System.Drawing.Drawing2D
: Provides advanced two-dimensional and vector graphics functionality.

System.Drawing.Imaging
: Provides advanced GDI+ imaging functionality.

System.Drawing.Text
: Provides advanced GDI+ typography functionality. The classes in this namespace can be used to create and use collections of fonts.

System.Drawing.Printing
: Provides printing functionality.

Related Sections

Custom Control Painting and Rendering

Details how to provide code for painting controls.

Three Categories of Graphics Services

.NET Framework (current version)

The graphics offerings in Windows Forms fall into the following three broad categories:

- Two-dimensional (2-D) vector graphics
- Imaging
- Typography

2-D Vector Graphics

Two-dimensional vector graphics are primitives; such as lines, curves, and figures; that are specified by sets of points on a coordinate system. For example, a straight line is specified by its two endpoints, and a rectangle is specified by a point giving the location of its upper-left corner and a pair of numbers giving its width and height. A simple path is specified by an array of points that are connected by straight lines. A Bézier spline is a sophisticated curve specified by four control points.

GDI+ provides classes and structures that store information about the primitives themselves, classes that store information about how the primitives will be drawn, and classes that actually do the drawing. For example, the Rectangle structure stores the location and size of a rectangle; the Pen class stores information about line color, line width, and line style; and the Graphics class has methods for drawing lines, rectangles, paths, and other figures. There are also several Brush classes that store information about how closed figures and paths will be filled with colors or patterns.

You can record a vector image, which is a sequence of graphics commands, in a metafile. GDI+ provides the Metafile class for recording, displaying, and saving metafiles. With the MetafileHeader and MetaHeader classes, you can inspect the data stored in a metafile header.

Imaging

Certain kinds of pictures are difficult or impossible to display with the techniques of vector graphics. For example, the pictures on toolbar buttons and the pictures that appear as icons are difficult to specify as collections of lines and curves. A high-resolution digital photograph of a crowded baseball stadium is even more difficult to create with vector techniques. Images of this type are stored as bitmaps, which are arrays of numbers that represent the colors of individual dots on the screen. GDI+ provides the Bitmap class for displaying, manipulating, and saving bitmaps.

Typography

Typography is the display of text in a variety of fonts, sizes, and styles. GDI+ provides extensive support for this complex task. One of the new features in GDI+ is subpixel antialiasing, which gives text rendered on an LCD screen a smoother appearance.

In addition, Windows Forms offers the option to draw text with GDI capabilities in its TextRenderer class.

See Also

Graphics Overview (Windows Forms)
About GDI+ Managed Code
Using Managed Graphics Classes

Windows Forms Print Support

.NET Framework (current version)

Printing in Windows Forms consists primarily of using the PrintDocument Component (Windows Forms) component to enable the user to print, and the PrintPreviewDialog Control (Windows Forms) control, PrintDialog Component (Windows Forms) and PageSetupDialog Component (Windows Forms) components to provide a familiar graphical interface to users accustomed to the Windows operating system.

Typically, you create a new instance of the PrintDocument component, set the properties that describe what to print using the PrinterSettings and PageSettings classes, and call the Print method to actually print the document.

During the course of printing from a Windows-based application, the PrintDocument component will show an abort print dialog box to alert users to the fact that printing is occurring and to allow the print job to be canceled.

In This Section

How to: Create Standard Windows Forms Print Jobs
: Explains how to use the PrintDocument component to print from a Windows Form.

How to: Capture User Input from a PrintDialog at Run Time
: Explains how to modify selected print options programmatically using the PrintDialog component.

How to: Choose the Printers Attached to a User's Computer in Windows Forms
: Describes changing the printer to print to using the PrintDialog component at run time.

How to: Print Graphics in Windows Forms
: Describes sending graphics to the printer.

How to: Print a Multi-Page Text File in Windows Forms
: Describes sending text to the printer.

How to: Complete Windows Forms Print Jobs
: Explains how to alert users to the completion of a print job.

How to: Print a Windows Form
: Shows how to print a copy of the current form.

How to: Print in Windows Forms Using Print Preview
: Shows how to use a PrintPreviewDialog for printing a document.

Related Sections

PrintDocument Component (Windows Forms)
: Explains usage of the PrintDocument component.

PrintDialog Component (Windows Forms)
: Explains usage of the PrintDialog component.

PrintPreviewDialog Control (Windows Forms)

Explains usage of the PrintPreviewDialog control.

PageSetupDialog Component (Windows Forms)

Explains usage of the PageSetupDialog component.

System.Drawing.Printing

Describes the classes in the System.Drawing.Printing namespace.

Drag-and-Drop Operations and Clipboard Support

.NET Framework (current version)

You can enable user drag-and-drop operations within a Windows-based application by handling a series of events, most notably the DragEnter, DragLeave, and DragDrop events.

You can also implement user cut/copy/paste support and user data transfer to the Clipboard within your Windows-based applications by using simple method calls.

In This Section

Walkthrough: Performing a Drag-and-Drop Operation in Windows Forms
: Explains how to start a drag-and-drop operation.

How to: Perform Drag-and-Drop Operations Between Applications
: Illustrates how to accomplish drag-and-drop operations across applications.

How to: Add Data to the Clipboard
: Describes a way to programmatically insert information on the Clipboard.

How to: Retrieve Data from the Clipboard
: Describes how to access the data stored on the Clipboard.

Related Sections

Drag-and-Drop Functionality in Windows Forms
: Describes the methods, events, and classes used to implement drag-and-drop behavior.

QueryContinueDrag
: Describes the intricacies of the event that asks permission to continue the drag operation.

DoDragDrop
: Describes the intricacies of the method that is central to beginning a drag operation.

Clipboard

Also see How to: Send Data to the Active MDI Child.

Networking in Windows Forms Applications

.NET Framework (current version)

The .NET Framework provides classes for displaying Web pages, downloading Web content, interacting with file transfer protocol (FTP) sites, and consuming Web Services, making it easy to build network functionality into your application. The following resources will help you understand the networking technologies of the .NET Framework and how you can integrate them into Windows Forms.

Reference

System.Net
: The root namespace for classes in the .NET Framework that handle network connectivity.

WebClient
: A convenient class for retrieving Web or HTTP-based content programmatically.

FtpWebRequest
: A class for retrieving and sending files with FTP.

WebBrowser
: A managed wrapper class for the **WebBrowser** control that is included with Windows.

Related Sections

Network Programming in the .NET Framework
: An introduction to networking in the .NET Framework.

Windows Forms Data Binding
: Describes how to display database content in your application, either from a local data store or a database located on a network.

Globalizing Windows Forms

.NET Framework (current version)

Globalization is the process of designing and developing a software product that functions for multiple cultures.

In This Section

Encoding and Windows Forms Globalization
Describes full Unicode support and its implications.

International Fonts in Windows Forms and Controls
Explains when and how to select fonts for display of international characters on Windows Forms.

Display of Asian Characters with the ImeMode Property
Introduces the **ImeMode** property, which is used to control the type of input a Windows Form or control accepts.

- Walkthrough: Downloading Satellite Assemblies on Demand with the ClickOnce Deployment API Using the Designer
- Localizing ClickOnce Applications
- Walkthrough: Downloading Satellite Assemblies on Demand with the ClickOnce Deployment API
- How to: Set the Culture and UI Culture for Windows Forms Globalization
- How to: Create Mirrored Windows Forms and Controls
- How to: Support Localization on Windows Forms Using AutoSize and the TableLayoutPanel Control
- Walkthrough: Localizing Windows Forms
- Walkthrough: Creating a Layout That Adjusts Proportion for Localization
- How to: Create Message Boxes for Bi-Directional Windows Forms
- Walkthrough: Downloading Satellite Assemblies on Demand with the ClickOnce Deployment API Using the Designer
- Localizing ClickOnce Applications
- Walkthrough: Downloading Satellite Assemblies on Demand with the ClickOnce Deployment API

Related Sections

1. Globalizing and Localizing Applications
2. Globalizing Applications
3. Globalizing and Localizing Applications

4. Globalizing Applications

Windows Forms and Unmanaged Applications

.NET Framework (current version)

Windows Forms applications and controls can interoperate with unmanaged applications, with some caveats. The following sections describe the scenarios and configurations that Windows Forms applications and controls support and those that they do not support.

In This Section

Windows Forms and Unmanaged Applications Overview
: Offers general information about how to use and implement Windows Forms controls that work with unmanaged applications.

How to: Support COM Interop by Displaying a Windows Form with the ShowDialog Method
: Provides a code example that shows how to use the Form.ShowDialog method to run a Windows Form in an unmanaged application.

How to: Support COM Interop by Displaying Each Windows Form on Its Own Thread
: Provides a code example that shows how to run a Windows Form on its own thread.

Also see Walkthrough: Supporting COM Interop by Displaying Each Windows Form on Its Own Thread.

Reference

Form.ShowDialog
: Used to create a separate thread for a Windows Form.

Application.Run
: Starts a message loop for a thread.

Invoke
: Marshals calls from an unmanaged application to a form.

Related Sections

Exposing .NET Framework Components to COM
: Offers general information about how to use .NET Framework types in unmanaged applications.

System Information and Windows Forms

.NET Framework (current version)

Sometimes it is necessary to gather information about the computer that your application is running on in order to make decisions in your code. For example, you might have a function that is only applicable when connected to a particular network domain; in this case you would need a way to determine the domain and disable the function if the domain is not present.

Windows Forms applications can use the SystemInformation class to determine a number of things about a computer at run time. The following example demonstrates using the SystemInformation class to retrieve the UserName and UserDomainName:

VB

```
Dim User As String = Windows.Forms.SystemInformation.UserName
Dim Domain As String = Windows.Forms.SystemInformation.UserDomainName

MessageBox.Show("Good morning " & User & ". You are connected to " _
& Domain)
```

All members of the SystemInformation class are read-only; you cannot modify a user's settings. There are over 100 members of the class, returning information on everything from the number of monitors attached to the computer (MonitorCount) to the spacing of icons in Windows Explorer (IconHorizontalSpacing and IconVerticalSpacing).

Some of the more useful members of the SystemInformation class include ComputerName, DbcsEnabled, PowerStatus, and TerminalServerSession.

See Also

SystemInformation
Power Management in Windows Forms

Power Management in Windows Forms

.NET Framework (current version)

Your Windows Forms applications can take advantage of the power management features in the Windows operating system. Your applications can monitor the power status of a computer and take action when a status change occurs. For example, if your application is running on a portable computer, you might want to disable certain features in your application when the computer's battery charge falls under a certain level.

The .NET Framework provides a PowerModeChanged event that occurs whenever there is a change in power status, such as when a user suspends or resumes the operating system, or when the AC power status or battery status changes. The PowerStatus property of the SystemInformation class can be used to query for the current status, as shown in the following code example.

VB

```
Public Sub New()
    InitializeComponent()
    AddHandler Microsoft.Win32.SystemEvents.PowerModeChanged, AddressOf PowerModeChanged
End Sub

Private Sub PowerModeChanged(ByVal Sender As System.Object, ByVal e As
Microsoft.Win32.PowerModeChangedEventArgs)
    Select Case SystemInformation.PowerStatus.BatteryChargeStatus
        Case BatteryChargeStatus.Low
            MessageBox.Show("Battery is running low.", "Low Battery",
MessageBoxButtons.OK, _
                            System.Windows.Forms.MessageBoxIcon.Exclamation)
        Case BatteryChargeStatus.Critical
            MessageBox.Show("Battery is critically low.", "Critical Battery",
MessageBoxButtons.OK, _
                            System.Windows.Forms.MessageBoxIcon.Stop)
        Case Else
            ' Battery is okay.
            Exit Select
    End Select
End Sub
```

Besides the BatteryChargeStatus enumerations, the PowerStatus property also contains enumerations for determining battery capacity (BatteryFullLifetime) and battery charge percentage (BatteryLifePercent, BatteryLifeRemaining).

You can use the SetSuspendState method of the Application to put a computer into hibernation or suspend mode. If the *force* argument is set to **false**, the operating system will broadcast an event to all applications requesting permission to suspend. If the *disableWakeEvent* argument is set to **true**, the operating system disables all wake events.

The following code example demonstrates how to put a computer into hibernation.

VB

```
If SystemInformation.PowerStatus.BatteryChargeStatus =
System.Windows.Forms.BatteryChargeStatus.Critical Then
    Application.SetSuspendState(PowerState.Hibernate, False, False)
End If
```

See Also

PowerModeChanged
PowerStatus
SetSuspendState
SessionSwitch

Help Systems in Windows Forms Applications

.NET Framework (current version)

One of the most important courtesies you, as a developer of applications, can furnish your users with is a competent Help system. This is where they will turn when they become confused or disoriented. Providing a Help system in a Windows-based application is easily done by using the HelpProvider Component (Windows Forms).

Different Types of Help

The Windows Forms HelpProvider component is used to associate an HTML Help 1.x Help file (either a .chm file, produced with the HTML Help Workshop, or an .htm file) with your Windows-based application. The HelpProvider component can be used to provide context-sensitive Help for controls on Windows Forms or specific controls. Additionally, the HelpProvider component can open a Help file to specific areas, such as the main page of a table of contents, an index, or a search function. For general information about the HelpProvider component, see HelpProvider Component Overview (Windows Forms). For information on how to use the HelpProvider component to show pop-up Help on Windows Forms, see How to: Display Pop-up Help. For information on using the ToolTip component to show control-specific Help, see Control Help Using ToolTips.

You can generate HTML Help 1.x files with the HTML Help Workshop. For more information on HTML Help, see the "HTML Help Workshop" or the other "HTML Help" topics in MSDN.

See Also

Integrating User Help in Windows Forms
HelpProvider Component (Windows Forms)
ToolTip Component (Windows Forms)
Windows Forms Overview
Windows Forms

Windows Forms Visual Inheritance

.NET Framework (current version)

Occasionally, you may decide that a project calls for a form similar to one that you have created in a previous project. Or, you may want to create a basic form with settings such as a watermark or certain control layout that you will then use again within a project, with each iteration containing modifications to the original form template. Form inheritance enables you to create a base form and then inherit from it and make modifications while preserving whatever original settings you need.

You can create derived-class forms programmatically or by using the Visual Inheritance picker.

In This Section

How to: Inherit Windows Forms
: Gives directions for creating inherited forms in code.

How to: Inherit Forms Using the Inheritance Picker Dialog Box
: Gives directions for creating inherited forms with the Inheritance Picker.

Effects of Modifying a Base Form's Appearance
: Gives directions for changing a base form's controls and their properties.

Walkthrough: Demonstrating Visual Inheritance
: Describes how to create a base Windows Form and compile it into a class library. You will import this class library into another project, and create a new form that inherits from the base form.

How to: Use the Modifiers and GenerateMember Properties
: Gives directions for using the **GenerateMember** and **Modifiers** properties, which are relevant when the Windows Forms Designer generates a member variable for a component.

Related Sections

NOT IN BUILD: Inheritance in Visual Basic
: Describes how to define Visual Basic classes that serve as the basis for other classes.

class (C# Reference)
: Describes the C# approach of classes, in which single inheritance is allowed.

Troubleshooting Inherited Event Handlers in Visual Basic
: Lists common issues that arise with event handlers in inherited components

Multiple-Document Interface (MDI) Applications

.NET Framework (current version)

Multiple-document interface (MDI) applications enable you to display multiple documents at the same time, with each document displayed in its own window. MDI applications often have a Window menu item with submenus for switching between windows or documents.

Note

There are some behavior differences between MDI forms and single-document interface (SDI) windows in Windows Forms. The **Opacity** property does not affect the appearance of MDI child forms. Additionally, the CenterToParent method does not affect the behavior of MDI child forms.

In This Section

How to: Create MDI Parent Forms
Gives directions for creating the container for the multiple documents within an MDI application.

How to: Create MDI Child Forms
Gives directions for creating one or more windows that operate within an MDI parent form.

How to: Determine the Active MDI Child
Gives directions for verifying the child window that has focus (and sending its contents to the Clipboard).

How to: Send Data to the Active MDI Child
Gives directions for transporting information to the active child window.

How to: Arrange MDI Child Forms
Gives directions for tiling, cascading, or arranging the child windows of an MDI application.

Integrating User Help in Windows Forms

.NET Framework (current version)

An essential, but often overlooked, aspect of building Windows-based applications is the Help system, as this is where users turn for assistance in times of confusion. Windows Forms support two different types of Help, each provided by the HelpProvider Component (Windows Forms). The first involves pointing the user to a Help file of either HTML or HTML Help 1.*x* or greater format. The second can display brief "What's This"-type Help on individual controls; this is especially useful on dialog boxes. Both types of Help can be used on the same form.

Additionally, the ToolTip Component (Windows Forms) can be used to provide individual Help for controls on Windows Forms.

In This Section

How to: Provide Help in a Windows Application
: Explains how to use the **HelpProvider** component to link controls to files in a Help system.

How to: Display Pop-up Help
: Explains how to use the **HelpProvider** component to show pop-up Help on Windows Forms.

Control Help Using ToolTips
: Describes using the **ToolTip** component to show control-specific Help.

Related Sections

HelpProvider Component (Windows Forms)
: Explains the basics of the **HelpProvider** component.

ToolTip Component (Windows Forms)
: Explains the basics of the **ToolTip** component.

Windows Forms Overview
: Explains the basics of Windows Forms.

Windows Forms
: Provides an overview of Windows Forms.

Windows Forms Accessibility

.NET Framework (current version)

The accessibility functionality of Windows Forms allows you to make your application available to a wide variety of users.

In This Section

Walkthrough: Creating an Accessible Windows-based Application
Describes all of the features you should support to increase accessibility.

Reference

Accessibility
A namespace containing a number of classes related to accessibility.

AccessibleObject
Provides information that accessibility applications use to adjust an application's user interface (UI) for users with impairments.

Related Sections

Providing Accessibility Information for Controls on a Windows Form
Describes how to supply information that Windows Forms controls can use to assist users with impairments.

Automatic Scaling in Windows Forms
Describes how to make your Windows Forms application react to changes in the system font size.

Using WPF Controls

.NET Framework (current version)

You can use Windows Presentation Foundation (WPF) controls in your Windows Forms-based applications. Although these are two different view technologies, they interoperate smoothly.

The Windows Forms Designer provides a visual design environment for hosting Windows Presentation Foundation controls. A WPF control is hosted by a special Windows Forms control that is named ElementHost. This control enables the WPF control to participate in the form's layout and to receive keyboard and mouse messages. At design time, you can arrange the ElementHost control just as you would any Windows Forms control.

You can also use Windows Forms controls in your WPF-based applications. For more information, see WPF Designer.

In This Section

How to: Copy and Paste an ElementHost Control at Design Time
: Shows how to copy a Windows Presentation Foundation control on a Windows Form.

Walkthrough: Arranging WPF Content on Windows Forms at Design Time
: Shows how to use the Windows Forms layout features, such as anchoring and snaplines, to arrange Windows Presentation Foundation controls.

Walkthrough: Changing Properties of a Hosted WPF Element at Design Time
: Shows the workflow between the Windows Forms Designer and the WPF Designer for Visual Studio for changing properties on WPF controls.

Walkthrough: Creating New WPF Content on Windows Forms at Design Time
: Shows how to create a Windows Presentation Foundation control for use in your Windows Forms-based applications.

Walkthrough: Copying and Pasting an ElementHost Control into Separate Windows Forms
: Shows how to copy a Windows Presentation Foundation control from one Windows Form to another.

Walkthrough: Assigning WPF Content on Windows Forms at Design Time
: Shows how to select the Windows Presentation Foundation control types you want to display on your form.

Walkthrough: Styling WPF Content
: Shows the workflow between the Windows Forms Designer and the WPF Designer for applying styles to Windows Presentation Foundation controls.

Reference

ElementHost
: Describes a class which you can use to host Windows Presentation Foundation controls in your Windows Forms-based applications.

WindowsFormsHost
: Describes a class which you can use to host Windows Forms controls in your Windows Presentation

Foundation-based application.

Related Sections

Migration and Interoperability

Describes interoperation between the Windows Presentation Foundation and Windows Forms technologies.

WPF Designer

Describes how to design Windows Presentation Foundation controls in Visual Studio.