

Contents

[Aree WPF avanzate](#)

[Architettura WPF](#)

[XAML in WPF](#)

[Panoramica di XAML \(WPF\)](#)

[Descrizione dettagliata della sintassi XAML](#)

[Code-behind e XAML in WPF](#)

[Classi XAML e personalizzate per WPF](#)

[Estensioni di markup e XAML WPF](#)

[Spazi dei nomi XAML e mapping dello spazio dei nomi per XAML WPF](#)

[Ambiti dei nomi XAML WPF](#)

[Stili e modelli inline](#)

[TypeConverter e XAML](#)

[Estensioni XAML WPF](#)

[Associazione dell'estensione di markup](#)

[Estensione del markup ColorConvertedBitmap](#)

[Estensione del markup ComponentResourceKey](#)

[Sintassi DateTime XAML](#)

[Estensione del markup DynamicResource](#)

[Estensione del markup RelativeSource](#)

[Estensione del markup StaticResource](#)

[Estensione del markup TemplateBinding](#)

[Estensione di markup ThemeDictionary](#)

[Sintassi XAML di PropertyPath](#)

[Attributo PresentationOptions:Freeze](#)

[Compatibilità dei markup \(mc:\) Funzionalità del linguaggio](#)

[Attributo mc:Ignorable](#)

[Attributo mc:ProcessContent](#)

[Classi degli elementi di base](#)

[Cenni preliminari sugli elementi di base](#)

Cenni preliminari sugli oggetti Freezable

Panoramica su allineamento, margini e spaziatura interna

Procedure relative

Rendere trasparente o semitrasparente un oggetto UIElement

Aggiungere un'animazione alle dimensioni di un oggetto FrameworkElement

Determinare se un oggetto Freezable è bloccato

Gestire un evento caricato

Impostare i margini di elementi e controlli

Impostare la proprietà di sola lettura per un oggetto Freezable

Ottenerne una copia scrivibile di un oggetto Freezable di sola lettura

Capovolgere un oggetto UIElement orizzontalmente o verticalmente

Usare un oggetto ThicknessConverter

Gestire l'evento ContextMenuOpening

Struttura ad albero e serializzazione degli elementi

Strutture ad albero in WPF

Limitazioni relative alla serializzazione di XamlWriter.Save

Inizializzazione di elementi oggetto non presenti in una struttura ad albero di oggetti

Procedure relative

Trovare un elemento in base al nome

Eseguire l'override dell'albero logico

Sistema di proprietà WPF

Panoramica sulle proprietà di dipendenza

Cenni preliminari sulle proprietà associate

Proprietà di dipendenza personalizzate

Metadati delle proprietà di dipendenza

Callback e convalida delle proprietà di dipendenza

Metadati delle proprietà del framework

Precedenza del valore della proprietà di dipendenza

Proprietà di dipendenza di sola lettura

Ereditarietà del valore della proprietà

Sicurezza delle proprietà di dipendenza

Modelli di costruttore sicuri per DependencyObject

[Proprietà di dipendenza di tipo raccolta](#)

[Caricamento XAML e proprietà di dipendenza](#)

[Procedure relative](#)

[Implementare una proprietà di dipendenza](#)

[Aggiungere un tipo di proprietario per una proprietà di dipendenza](#)

[Registrare una proprietà associata](#)

[Eseguire l'override dei metadati per una proprietà di dipendenza](#)

[Eventi](#)

[Cenni preliminari sugli eventi indirizzati](#)

[Cenni preliminari sugli eventi associati](#)

[Eventi di durata degli oggetti](#)

[Impostazione degli eventi indirizzati come gestiti e gestione delle classi](#)

[Eventi di anteprima](#)

[Eventi di modifica delle proprietà](#)

[Visual Basic e la gestione degli eventi WPF](#)

[Modelli di eventi deboli](#)

[Procedure relative](#)

[Aggiungere un gestore eventi usando il codice](#)

[Gestire un evento indirizzato](#)

[Creare un evento indirizzato personalizzato](#)

[Cercare l'elemento di origine in un gestore eventi](#)

[Aggiungere la gestione di classi per un evento indirizzato](#)

[Input](#)

[Cenni preliminari sull'input](#)

[Cenni preliminari sull'esecuzione di comandi](#)

[Cenni preliminari sullo stato attivo](#)

[Applicazione di stili per lo stato attivo nei controlli e FocusVisualStyle](#)

[Procedura dettagliata: Creazione della prima applicazione a tocco](#)

[Procedure relative](#)

[Attivare un comando](#)

[Modificare il tipo di cursore](#)

[Modificare il colore di un elemento usando eventi focus](#)

[Applicare un oggetto FocusVisualStyle a un controllo](#)

[Rilevare il momento in cui è stato premuto il tasto INVIO](#)

[Creare un effetto di attivazione usando gli eventi](#)

[Fare in modo che un oggetto segua il puntatore del mouse](#)

[Creare un oggetto RoutedCommand](#)

[Implementare ICommandSource](#)

[Associare un comando a un controllo senza supporto del comando](#)

[Associare un comando a un controllo con supporto del comando](#)

[Input penna](#)

[Cenni preliminari](#)

[Introduzione all'input penna](#)

[Raccolta di input penna](#)

[Riconoscimento della grafia](#)

[Memorizzazione dell'input penna](#)

[Modello a oggetti input penna: confronto di Windows Form e COM con WPF](#)

[Gestione avanzata dell'input penna](#)

[Personalizzare il rendering dell'input penna](#)

[Intercettazione dell'input dello stilo](#)

[Creazione di un controllo di input penna](#)

[Modello di threading dell'input penna](#)

[Procedure relative](#)

[Selezionare l'input penna da un controllo personalizzato](#)

[Aggiungere dati personalizzati ai dati dell'input penna](#)

[Cancellare l'input penna da un controllo personalizzato](#)

[Riconoscere i movimenti dell'applicazione](#)

[Trascinare l'input penna](#)

[Eseguire l'associazione dati a un InkCanvas](#)

[Analizzare l'input penna con i suggerimenti dell'analisi](#)

[Ruotare l'input penna](#)

[Disabilitare RealTimeStylus per le applicazioni WPF](#)

[Trascinamento e rilascio](#)

[Cenni preliminari sul trascinamento della selezione](#)

Dati e oggetti dati

Procedura dettagliata: Abilitare il trascinamento della selezione in un controllo utente

Procedure relative

[Aprire un file rilasciato in un controllo RichTextBox](#)

[Creare un oggetto dati](#)

[Determinare se un formato dati è presente in un oggetto dati](#)

[Elencare i formati dati in un oggetto dati](#)

[Recuperare dati in un formato dati particolare](#)

[Archiviare più formati dati in un oggetto dati](#)

Risorse

Risorse XAML

[Risorse e codice](#)

[Dizionari risorse uniti](#)

Procedure relative

[Definire e fare riferimento a una risorsa](#)

[Usare le risorse dell'applicazione](#)

[Usare la classe SystemFonts](#)

[Usare le chiavi dei tipi di carattere del sistema](#)

[Usare la classe SystemParameters](#)

[Usare le chiavi dei parametri di sistema](#)

Documenti

Documenti in WPF

Serializzazione e archiviazione di documenti

Annotazioni

[Cenni preliminari sulle annotazioni](#)

[Schema annotazioni](#)

Contenuto del flusso

[Cenni preliminari sui documenti dinamici](#)

[Cenni preliminari sul modello di contenuto TextElement](#)

[Cenni preliminari sull'elemento Table](#)

Procedure relative

[Regolare la spaziatura tra i paragrafi](#)

[Creare una tabella a livello di codice](#)

[Modificare l'oggetto FlowDirection del contenuto a livello di codice](#)

[Modificare la proprietà TextWrapping a livello di codice](#)

[Definire una tabella tramite XAML](#)

[Modificare la tipografia del testo](#)

[Abilitare l'enumerazione TextTrimming](#)

[Inserire un elemento in un testo a livello di codice](#)

[Modificare elementi di contenuto di flusso tramite la proprietà Blocks](#)

[Modificare elementi di contenuto di flusso tramite la proprietà Inlines](#)

[Modificare un oggetto FlowDocument tramite la proprietà Blocks](#)

[Modificare le colonne di una tabella tramite la proprietà Columns](#)

[Modificare i gruppi di righe di una tabella tramite la proprietà RowGroups](#)

[Usare elementi di contenuto di flusso](#)

[Usare gli attributi di separazione delle colonne di un oggetto FlowDocument](#)

Opzioni tipografiche

[Funzionalità tipografiche di WPF](#)

[Cenni preliminari su ClearType](#)

[Impostazioni del Registro di sistema ClearType](#)

[Disegno di testo formattato](#)

[Formattazione del testo avanzata](#)

[Tipi di carattere](#)

[Funzionalità dei tipi di carattere OpenType](#)

[Includere i tipi di carattere nel pacchetto delle applicazioni](#)

[Esempio di pacchetto di tipi di carattere OpenType](#)

[Procedure relative](#)

[Enumerare i tipi di carattere del sistema](#)

[Usare la classe FontSizeConverter](#)

[Glifi](#)

[Introduzione all'oggetto GlyphRun e all'elemento Glyphs](#)

[Procedura: Creare testo utilizzando i glifi](#)

[Procedure relative](#)

[Creare un effetto testo](#)

[Specificare se un collegamento ipertestuale è sottolineato](#)

[Applicare trasformazioni al testo](#)

[Applicare animazioni al testo](#)

[Creare testo con un'ombreggiatura](#)

[Creare testo con contorni](#)

[Disegnare testo sullo sfondo di un controllo](#)

[Disegnare testo in un oggetto visivo](#)

[Usare caratteri speciali in XAML](#)

[Stampa e gestione di sistemi di stampa](#)

[Cenni preliminari sulla stampa](#)

[Procedure relative](#)

[Richiamare una finestra di dialogo di stampa](#)

[Clonare una stampante](#)

[Diagnosticare un processo di stampa problematico](#)

[Scoprire se è possibile eseguire un processo di stampa a quest'ora del giorno](#)

[Enumerare un sottoinsieme di code di stampa](#)

[Ottenere le proprietà dell'oggetto del sistema di stampa senza reflection](#)

[Stampare file XPS a livello di codice](#)

[Sorvegliare da remoto lo stato delle stampanti](#)

[Convalidare e unire PrintTicket](#)

[Globalizzazione e localizzazione](#)

[Panoramica della globalizzazione e localizzazione WPF](#)

[Globalizzazione per WPF](#)

[Cenni preliminari sull'utilizzo del layout automatico](#)

[Attributi e commenti di localizzazione](#)

[Cenni preliminari sulle funzionalità bidirezionali di WPF](#)

[Procedure relative](#)

[Localizzare un'applicazione](#)

[Usare il layout automatico per creare un pulsante](#)

[Usare una griglia per il layout automatico](#)

[Usare un oggetto ResourceDictionary per gestire le risorse di tipo stringa localizzabili](#)

[Usare risorse in applicazioni localizzabili](#)

Layout

Migrazione e interoperabilità

Interoperatività di WPF e Windows Form

Architettura di input per l'interoperabilità tra Windows Form e WPF

Considerazioni sul layout per l'elemento WindowsFormsHost

Controlli Windows Form e controlli WPF equivalenti

Mapping di proprietà di Windows Form e WPF

Risoluzione dei problemi relativi ad applicazioni ibride

Procedura dettagliata: Hosting di un controllo Windows Form in WPF

Procedura dettagliata: Hosting di un controllo Windows Form in WPF tramite XAML

Procedura dettagliata: Hosting di un controllo composito Windows Form in WPF

Procedura dettagliata: Hosting di un controllo ActiveX in WPF

Procedura: Abilitare stili di visualizzazione in un'applicazione ibrida

Procedura dettagliata: Disposizione di controlli Windows Form in WPF

Procedura dettagliata: Data binding in applicazioni ibride

Procedura dettagliata: Hosting di un controllo composito WPF 3D in Windows Form

Procedura dettagliata: Hosting di un controllo composito WPF in Windows Form

Procedura dettagliata: Mapping di proprietà tramite il controllo ElementHost

Procedura dettagliata: Mapping di proprietà tramite l'elemento WindowsFormsHost

Procedura dettagliata: Localizzazione di un'applicazione ibrida

Interoperatività di WPF e Win32

Cenni preliminari sulle aree di tecnologia

Condivisione dei cicli di messaggi tra Win32 e WPF

Hosting di contenuto Win32 in WPF

Procedura dettagliata: Hosting di un controllo Win32 in WPF

Procedura dettagliata: Hosting di contenuto WPF in Win32

Procedura dettagliata: Hosting di un orologio WPF in Win32

Interoperatività di WPF e Direct3D9

Considerazioni sulle prestazioni per l'interoperabilità fra Direct3D9 e WPF

Procedura dettagliata: Creazione di contenuto Direct3D9 per l'hosting in WPF

Procedura dettagliata: Hosting di contenuto Direct3D9 in WPF

Prestazioni

Livelli di rendering della grafica

Ottimizzazione delle prestazioni di applicazioni WPF

Pianificazione delle prestazioni dell'applicazione

Sfruttare appieno l'hardware

Layout e progettazione

Grafica 2D e creazione di immagini

Comportamento degli oggetti

Risorse dell'applicazione

Testo

Data binding

Controlli

Altri suggerimenti relativi alle prestazioni

Tempo di avvio delle applicazioni

Procedura dettagliata: Memorizzazione dei dati di un'applicazione nella cache di un'applicazione WPF

Modello di threading

Riferimenti alle API non gestite WPF

Funzione Activate

Funzione CreateIDispatchSTAForwarder

Funzione Deactivate

Funzione ForwardTranslateAccelerator

Funzione LoadFromHistory

Funzione ProcessUnhandledException

Funzione SaveToHistory

Funzione SetFakeActiveWindow

Aree avanzate (Windows Presentation Foundation)

23/10/2019 • 2 minutes to read • [Edit Online](#)

Questa sezione descrive alcune delle aree avanzate in WPF.

In questa sezione

[Architettura WPF](#)

[XAML in WPF](#)

[Classi degli elementi di base](#)

[Albero degli elementi e serializzazione](#)

[Sistema di proprietà WPF](#)

[Eventi in WPF](#)

[Input](#)

[Trascinamento della selezione](#)

[Risorse](#)

[Documenti](#)

[Globalizzazione e localizzazione](#)

[Layout](#)

[Migrazione e interoperabilità](#)

[Prestazioni](#)

[Modello di threading](#)

[Informazioni di riferimento sulle API WPF non gestite](#)

Architettura WPF

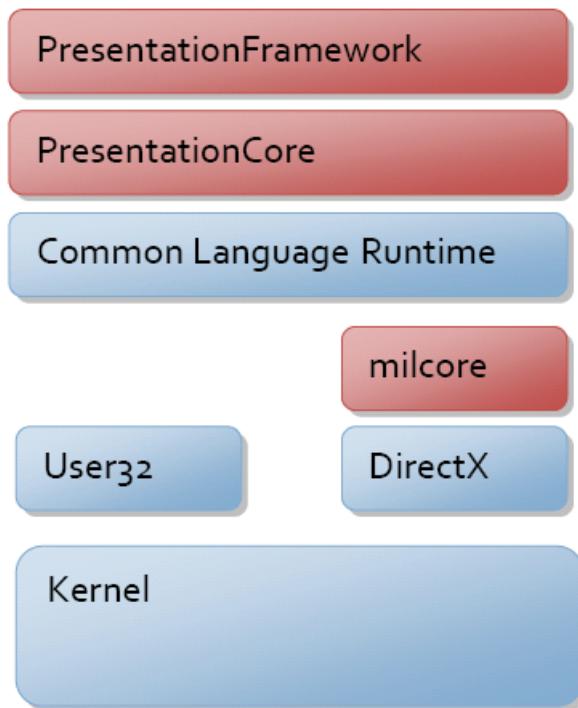
31/01/2020 • 31 minutes to read • [Edit Online](#)

In questo argomento viene fornita una presentazione guidata della gerarchia di classi Windows Presentation Foundation (WPF). Viene illustrata la maggior parte dei principali sottosistemi di WPF e viene descritta la modalità di interazione. Vengono inoltre illustrate alcune delle scelte effettuate dagli architetti di WPF.

System.Object

Il modello di programmazione WPF primario viene esposto tramite codice gestito. Nelle fasi iniziali della fase di progettazione di WPF sono state illustrate alcune discussioni sulla posizione in cui deve essere disegnata la linea tra i componenti gestiti del sistema e quelli non gestiti. CLR fornisce una serie di funzionalità che rendono lo sviluppo più produttivo e affidabile (inclusa la gestione della memoria, la gestione degli errori, Common Type System e così via), ma hanno un costo.

I componenti principali di WPF sono illustrati nella figura seguente. Le sezioni rosse del diagramma (PresentationFramework, PresentationCore e milcore) sono le parti principali del codice di WPF. Di queste, solo milcore è un componente non gestito. Milcore è scritto in codice non gestito per consentire una stretta integrazione con DirectX. Tutte le visualizzazioni in WPF vengono eseguite tramite il motore DirectX, consentendo un efficiente rendering dell'hardware e del software. WPF necessitava anche di un controllo accurato sulla memoria e sull'esecuzione. Il motore di composizione in milcore è estremamente sensibile alle prestazioni ed è necessario rinunciare a molti vantaggi di CLR per ottenere prestazioni ottimali.



La comunicazione tra le parti gestite e non gestite di WPF viene discussa più avanti in questo argomento. Il resto del modello di programmazione gestito viene descritto di seguito.

System.Threading.DispatcherObject

La maggior parte degli oggetti in WPF deriva da [DispatcherObject](#), che fornisce i costrutti di base per la gestione della concorrenza e del threading. WPF si basa su un sistema di messaggistica implementato dal dispatcher.

Funziona in modo molto simile a quello del messaggio Win32 familiare; in realtà, il dispatcher WPF USA messaggi User32 per l'esecuzione di chiamate cross-thread.

Esistono due concetti principali da comprendere quando si discute la concorrenza in WPF, ovvero il dispatcher e l'affinità di thread.

Durante la fase di progettazione di WPF, lo scopo era quello di passare a un singolo thread di esecuzione, ma a un modello "creata un'affinità" non thread. L'affinità di thread si verifica quando un componente utilizza l'identità del thread in esecuzione per archiviare un tipo di stato. Nella forma più comune, per archiviare lo stato viene utilizzata l'archiviazione locale di thread (TLS). L'affinità di thread richiede che ciascun thread di esecuzione logico appartenga a un solo thread fisico del sistema operativo, che può richiedere un elevato consumo di memoria. Alla fine, il modello di threading di WPF è stato tenuto in sincronia con il modello di threading User32 esistente dell'esecuzione a thread singolo con affinità di thread. Il motivo principale di questa interoperabilità è che i sistemi, ad esempio OLE 2,0, gli appunti e Internet Explorer, richiedono l'esecuzione di affinità a thread singolo (STA).

Disponendo di oggetti con threading STA, è necessaria una modalità di comunicazione tra thread e di conferma di utilizzo del thread corretto. Qui si colloca il ruolo del dispatcher. Il dispatcher è un sistema di base di invio di messaggi con più code in ordine di priorità. Gli esempi di messaggi includono le notifiche di input non elaborato (spostamenti del mouse), le funzionalità del framework (layout) o i comandi utente (esecuzione di metodi).

Derivando da [DispatcherObject](#), viene creato un oggetto CLR con un comportamento STA e viene assegnato un puntatore a un dispatcher in fase di creazione.

System.Windows.DependencyObject

Una delle principali filosofie dell'architettura utilizzate nella creazione di WPF era una preferenza per le proprietà di metodi o eventi. Le proprietà sono dichiarative e consentono di specificare più facilmente lo scopo anziché l'azione. In tal modo veniva anche supportato un sistema basato sui modelli o sui dati per la visualizzazione dei contenuti dell'interfaccia utente. Lo scopo di tale approccio era di creare un maggior numero di proprietà con cui stabilire l'associazione per poter controllare meglio il comportamento di un'applicazione.

Per avere un maggior numero di sistemi basati sulle proprietà, è necessario un sistema di proprietà più completo rispetto a quello fornito da CLR. Un semplice esempio è rappresentato dalle notifiche di modifica. Per abilitare l'associazione bidirezionale, è necessario che entrambi i lati supportino la notifica di modifica. Per associare il comportamento a valori di proprietà, è necessario ricevere una notifica quando tale valore viene modificato. Il Framework Microsoft .NET dispone di un'interfaccia, **INotifyPropertyChanged**, che consente a un oggetto di pubblicare notifiche di modifica, ma è facoltativo.

WPF fornisce un sistema di proprietà più completo, derivato dal tipo di [DependencyObject](#). Il sistema di proprietà è effettivamente un sistema di proprietà delle "dipendenze" nel senso che tiene traccia delle dipendenze tra espressioni di proprietà e riconvalida automaticamente i valori delle proprietà quando tali dipendenze vengono modificate. Se, ad esempio, si dispone di una proprietà che eredita (come [FontSize](#)), il sistema viene aggiornato automaticamente se la proprietà viene modificata in un elemento padre di un elemento che eredita il valore.

La base del sistema di proprietà WPF è il concetto di espressione di proprietà. Nella prima versione di WPF, il sistema di espressioni di proprietà viene chiuso e tutte le espressioni vengono fornite come parte del Framework. Le espressioni rappresentano il motivo per cui il sistema di proprietà non dispone di associazione dati, stili o ereditarietà hardcoded forniti invece dai livelli successivi all'interno del framework.

Il sistema di proprietà, inoltre, presenta possibilità limitate di archiviazione per i valori di proprietà. Poiché gli oggetti hanno decine, se non centinaia, di proprietà e la maggior parte dei valori si trova nel relativo stato predefinito (ereditato, impostato per stile e così via), non tutte le istanze di un oggetto devono avere il peso di ciascuna proprietà in esso definita.

La nuova funzionalità finale del sistema di proprietà è la nozione di proprietà associate. Gli elementi WPF sono basati sul principio della composizione e sul riutilizzo dei componenti. Spesso è necessario che un elemento

contenitore, ad esempio un elemento di layout [Grid](#), richieda dati aggiuntivi sugli elementi figlio per controllarne il comportamento, ad esempio le informazioni sulla riga o sulla colonna. Anziché associare tutte queste proprietà a ciascun elemento, un oggetto può fornire le definizioni delle proprietà per qualsiasi altro oggetto, come avviene per le funzionalità "expando" di JavaScript.

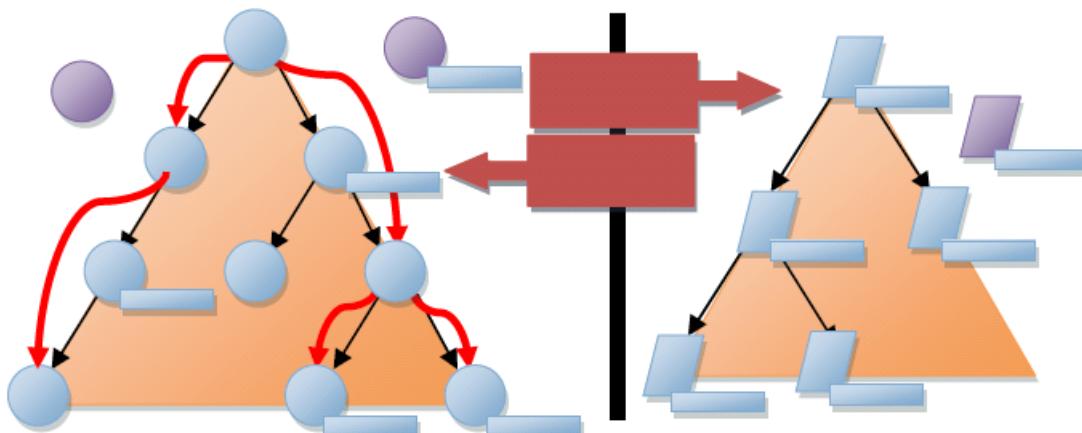
System.Windows.Media.Visual

Dopo avere definito un sistema, è necessario ottenere pixel disegnati sullo schermo. La classe [Visual](#) fornisce per la compilazione di una struttura ad albero di oggetti visivi, ognuno dei quali contiene facoltativamente istruzioni di disegno e metadati su come eseguire il rendering di tali istruzioni (ritaglio, trasformazione e così via). [Visual](#) è progettato per essere estremamente leggero e flessibile, la maggior parte delle funzionalità non hanno un'esposizione API pubblica e si basano molto sulle funzioni di callback protette.

[Visual](#) è effettivamente il punto di ingresso del sistema di composizione WPF. [Visual](#) è il punto di connessione tra questi due sottosistemi, l'API gestita e il milcore non gestito.

WPF Visualizza i dati attraversando le strutture di dati non gestite gestite da milcore. Queste strutture, denominate nodi di composizione, rappresentano una struttura di visualizzazione gerarchica ad albero con istruzioni di rendering in ciascun nodo. La struttura ad albero, illustrata a destra nella figura riportata di seguito, è accessibile solo mediante un protocollo di messaggistica.

Quando si programma WPF, si creano [Visual](#) elementi e tipi derivati, che comunicano internamente all'albero della composizione tramite questo protocollo di messaggistica. Ogni [Visual](#) in WPF può creare uno, nessuno o più nodi di composizione.



In questo caso va notato un dettaglio molto importante relativo all'architettura: l'intera struttura ad albero degli elementi visivi e delle istruzioni di disegno è memorizzata nella cache. Nei termini grafici, WPF usa un sistema di rendering mantenuto. Ciò fornisce al sistema elevate frequenze di aggiornamento senza che il sistema di composizione si blocchi sui callback al codice utente e consente di impedire la visualizzazione di applicazioni che non rispondono.

Un altro dettaglio importante, non facilmente individuabile nel diagramma, è il modo in cui il sistema esegue effettivamente la composizione.

In User32 e GDI il sistema funziona in un sistema di ritaglio in modalità immediata. Quando è necessario eseguire il rendering di un componente, il sistema stabilisce un limite di ritaglio al di fuori del quale il componente non può toccare i pixel; successivamente, viene chiesto al componente di disegnare i pixel in tale casella. Questo sistema funziona molto bene nei sistemi con memoria limitata poiché quando vengono apportate modifiche basta intervenire solo sul componente interessato. Non sono mai richiesti due componenti per il colore di un singolo pixel.

WPF usa un modello di disegno "algoritmo del pittore". Ciò significa che anziché ritagliare ogni componente,

viene richiesto il rendering di ciascun componente dallo sfondo al primo piano della visualizzazione. In tal modo ciascun componente viene disegnato sulla visualizzazione del componente precedente. Il vantaggio di questo modello è la possibilità di avere forme complesse, parzialmente trasparenti. Con l'attuale hardware grafico moderno, questo modello è relativamente veloce, che non era il caso in cui User32/GDI fosse stato creato.

Come indicato in precedenza, una filosofia di base di WPF consiste nel passare a un modello di programmazione più dichiarativo "incentrato sulle proprietà". Nel sistema visuale questo è evidente in un paio di punti interessanti.

Innanzitutto, se si pensa al sistema grafico in modalità memorizzata, si nota un deciso allontanamento da un modello imperativo di tipo DrawLine/DrawLine verso un modello orientato ai dati di tipo new Line()/new Line(). Questo spostamento verso il rendering basato sui dati consente operazioni complesse sulle istruzioni di disegno da esprimere utilizzando le proprietà. I tipi che derivano da [Drawing](#) sono in effetti il modello a oggetti per il rendering.

In secondo luogo, se si esamina il sistema di animazione appare evidente che è quasi completamente dichiarativo. Anziché richiedere a uno sviluppatore di calcolare la posizione o il colore successivo, è possibile esprimere le animazioni come un set di proprietà su un oggetto di animazione. Le animazioni possono quindi esprimere l'intenzione dello sviluppatore o del progettista (spostare questo pulsante da questa a quella posizione in 5 secondi) e il sistema può stabilire il modo più efficace per raggiungere tale scopo.

System.Windows.UIElement

[UIElement](#) definisce sottosistemi di base, tra cui layout, input ed eventi.

Il layout è un concetto di base di WPF. In molti sistemi è presente un insieme fisso di modelli di layout (HTML supporta tre modelli di layout: flusso, assoluto e tavole) oppure non è presente alcun modello di layout (User32 supporta effettivamente solo il posizionamento assoluto). WPF ha iniziato con il presupposto che gli sviluppatori e i progettisti volevano un modello di layout flessibile ed estendibile, che poteva essere basato su valori di proprietà anziché su una logica imperativa. A livello di [UIElement](#), viene introdotto il contratto di base per il layout, ovvero un modello a due fasi con [Measure](#) e [Arrange](#).

[Measure](#) consente a un componente di determinare la quantità di dimensioni che si desidera eseguire. Si tratta di una fase separata da [Arrange](#) perché in molte situazioni un elemento padre chiede a un figlio di misurare più volte per determinare la posizione e le dimensioni ottimali. Il fatto che gli elementi padre chiedano agli elementi figlio di misurare dimostrano un'altra filosofia chiave di WPF, ovvero le dimensioni del contenuto. Tutti i controlli in WPF supportano la possibilità di ridimensionare le dimensioni naturali del contenuto. Tutto ciò semplifica il processo di localizzazione e consente il layout dinamico degli elementi quando gli oggetti vengono ridimensionati. La fase [Arrange](#) consente a un elemento padre di posizionare e determinare le dimensioni finali di ogni elemento figlio.

Spesso si parla del lato output di WPF, [Visual](#) e degli oggetti correlati. Esiste, tuttavia, una straordinaria quantità di innovazione anche sul lato input. Probabilmente la modifica più importante nel modello di input per WPF è il modello coerente in base al quale gli eventi di input vengono instradati attraverso il sistema.

L'input come segnale proviene da un driver di dispositivo in modalità kernel e viene indirizzato verso il processo e il thread corretti mediante un complicato processo che interessa il kernel di Windows e User32. Una volta che il messaggio User32 corrispondente all'input viene indirizzato a WPF, viene convertito in un messaggio di input non elaborato WPF e inviato al dispatcher. WPF consente la conversione di eventi di input non elaborati in più eventi effettivi, consentendo l'implementazione di funzionalità come "MouseEnter" a un livello basso del sistema con recapito garantito.

Ciascun evento di input viene convertito in almeno due eventi: un evento in "anteprima" e l'evento effettivo. Tutti gli eventi in WPF hanno una nozione di routing nell'albero degli elementi. Gli eventi vengono detti "bolle" se passano da una destinazione verso l'alto nell'albero alla radice e vengono detti "tunnel" se iniziano alla radice e passano a una destinazione. Gli eventi di input in anteprima effettuano il tunneling consentendo a qualsiasi elemento della struttura ad albero di filtrare o svolgere un'azione sull'evento. Gli eventi regolari (non in

anteprima) vengono propagati dalla destinazione fino alla radice.

Questa suddivisione tra fase di tunneling e fase di bubbling consente di implementare funzionalità, quali i tasti di scelta rapida, in modo coerente in un contesto composito. In User32 i tasti di scelta rapida vengono implementati con una sola tabella globale che contiene tutti i tasti da supportare (mapping di Ctrl+N su "Nuovo"). Nel dispatcher dell'applicazione viene chiamato **TranslateAccelerator** che rileva la presenza dei messaggi di input in User32 e stabilisce se esiste una corrispondenza con un tasto di scelta rapida registrato. In WPF questa operazione non funzionerebbe perché il sistema è completamente "componibile", qualsiasi elemento può gestire e utilizzare qualsiasi tasto di scelta rapida. Questo modello a due fasi per l'input consente ai componenti di implementare il proprio "TranslateAccelerator".

Per eseguire ulteriormente questo passaggio, [UIElement](#) introduce anche il concetto di CommandBindings. Il sistema di comandi WPF consente agli sviluppatori di definire le funzionalità in termini di un punto finale del comando, ovvero un elemento che implementa [ICommand](#). Le associazioni di comandi consentono a un elemento di definire un mapping tra un movimento di input (Ctrl+N) e un comando (Nuovo). I movimenti di input e le definizioni dei comandi sono estendibili e possono essere collegati al momento dell'utilizzo. In tal modo, ad esempio, è possibile consentire a un utente finale di personalizzare le combinazioni di tasti da utilizzare in un'applicazione.

Fino a questo punto nell'argomento, le funzionalità di base di WPF, ovvero le funzionalità implementate nell'assembly PresentationCore, hanno avuto lo stato attivo. Quando si compila WPF, è stata ottenuta una netta separazione tra le parti fondamentali (ad esempio il contratto per il layout con **misura** e **disposizione**) e le parti del Framework, ad esempio l'implementazione di un layout specifico come [Grid](#). Lo scopo era quello di fornire un punto di estendibilità basso nello stack per consentire agli sviluppatori esterni di creare i propri framework, se necessario.

System.Windows.FrameworkElement

[FrameworkElement](#) possono essere esaminati in due modi diversi. Introduce un set di criteri e personalizzazioni nei sottosistemi introdotti in livelli inferiori di WPF. e dall'altro un insieme di nuovi sottosistemi.

Il criterio principale introdotto da [FrameworkElement](#) riguarda il layout dell'applicazione. [FrameworkElement](#) si basa sul contratto di layout di base introdotto da [UIElement](#) e aggiunge la nozione di "slot" del layout che rende più semplice per gli autori di layout avere un set coerente di semantica di layout basata su Proprietà. Proprietà quali [HorizontalAlignment](#), [VerticalAlignment](#), [MinWidth](#) e [Margin](#) (per citarne alcune) forniscono tutti i componenti derivati da [FrameworkElement](#) comportamento coerente all'interno dei contenitori di layout.

[FrameworkElement](#) offre anche un'esposizione più semplice delle API a molte funzionalità disponibili nei livelli principali di WPF. Ad esempio, [FrameworkElement](#) fornisce accesso diretto all'animazione tramite il metodo [BeginStoryboard](#). Un [Storyboard](#) consente di creare script per più animazioni rispetto a un set di proprietà.

I due aspetti più importanti che [FrameworkElement](#) introduce sono data binding e gli stili.

Il sottosistema data binding in WPF dovrebbe essere relativamente familiare a chiunque abbia utilizzato Windows Forms o ASP.NET per la creazione di un'applicazione interfaccia utente. In ciascuno di questi sistemi è possibile effettuare in modo semplice l'associazione di una o più proprietà di un dato elemento a un blocco di dati. WPF dispone del supporto completo per l'associazione di proprietà, trasformazione ed elenco.

Una delle funzionalità più interessanti di data binding in WPF è l'introduzione dei modelli di dati. che consentono di specificare in modo dichiarativo come deve essere visualizzato un blocco di dati. Anziché creare un'interfaccia utente personalizzata che può essere associata ai dati, è possibile aggirare il problema e fare in modo che siano i dati a stabilire il tipo di visualizzazione da creare.

L'applicazione degli stili è effettivamente un tipo di associazione dati semplice e il suo utilizzo consente di associare un set di proprietà di una definizione condivisa a una o più istanze di un elemento. Gli stili vengono applicati a un elemento in base al riferimento esplicito (impostando la proprietà [Style](#)) o in modo implicito

associando uno stile al tipo CLR dell'elemento.

System.Windows.Controls.Control

La funzionalità più significativa dei controlli è l'applicazione di modelli. Considerando il sistema di composizione di WPF come un sistema di rendering in modalità memorizzata, l'applicazione di modelli consente a un controllo di descriverne il rendering in modo dichiarativo e con parametri. Una [ControlTemplate](#) non è nient'altro che uno script per creare un set di elementi figlio, con binding alle proprietà offerte dal controllo.

[Control](#) fornisce un set di proprietà predefinite, [Foreground](#), [BackgroundPadding](#), per citarne alcuni, che possono quindi essere utilizzati dagli autori di modelli per personalizzare la visualizzazione di un controllo.

L'implementazione di un controllo fornisce un modello dati e un modello di interazione. Il modello di interazione definisce un set di comandi (ad esempio, Chiudi per una finestra) e di associazioni a movimenti di input (ad esempio, fare clic sulla X rossa nell'angolo superiore della finestra). Il modello dati fornisce un set di proprietà per la personalizzazione del modello di interazione o della visualizzazione (stabilità del modello).

Questa suddivisione tra modello dati (proprietà), modello di interazione (comandi ed eventi) e modello di visualizzazione (modelli) consente una personalizzazione completa dell'aspetto e del comportamento di un controllo.

Una caratteristica comune del modello dati dei controlli è il modello di contenuto. Se si osserva un controllo come [Button](#), si noterà che è presente una proprietà denominata "Content" di tipo [Object](#). In Windows Forms e ASP.NET questa proprietà è in genere una stringa, ma limita il tipo di contenuto che è possibile inserire in un pulsante. Il contenuto di un pulsante può essere una stringa semplice, un oggetto dati complesso o l'intera struttura ad albero di un elemento. Nel caso di un oggetto dati, il modello dati viene utilizzato per costruire una visualizzazione.

Riepilogo

WPF è progettato per consentire la creazione di sistemi di presentazione dinamici basati sui dati. Ogni parte del sistema è progettata per la creazione di oggetti utilizzando set di proprietà che ne definiscono il comportamento. L'associazione dati è una parte fondamentale del sistema ed è integrata a ogni livello.

Le applicazioni tradizionali creano una visualizzazione e associano successivamente i dati. In WPF tutte le informazioni sul controllo, ogni aspetto dello schermo, vengono generate da un tipo di data binding. Il testo che si trova nei pulsanti viene visualizzato creando un controllo composto all'interno del pulsante e associando la relativa visualizzazione alla proprietà del contenuto del pulsante.

Quando si inizia a sviluppare applicazioni basate su WPF, il suo aspetto è molto familiare. È possibile impostare proprietà, usare oggetti e associare dati nello stesso modo in cui è possibile usare Windows Forms o ASP.NET. Con un'analisi più approfondita dell'architettura di WPF, si noterà che esiste la possibilità di creare applicazioni molto più ricche che considerano fondamentalmente i dati come driver di base dell'applicazione.

Vedere anche

- [Visual](#)
- [UIElement](#)
- [ICommand](#)
- [FrameworkElement](#)
- [DispatcherObject](#)
- [CommandBinding](#)
- [Control](#)
- [Cenni preliminari sull'associazione dati](#)
- [Layout](#)

- Cenni preliminari sull'animazione

XAML in WPF

03/02/2020 • 2 minutes to read • [Edit Online](#)

Extensible Application Markup Language (XAML) è un linguaggio di markup per la programmazione di applicazioni dichiarative. Windows Presentation Foundation (WPF) implementa un'implementazione del processore XAML e fornisce il supporto del linguaggio XAML. I tipi di WPF vengono implementati in modo che possano fornire il supporto del tipo richiesto per una rappresentazione XAML. In generale, è possibile creare la maggior parte dell'interfaccia utente dell'applicazione WPF nel markup XAML.

Contenuto della sezione

[Panoramica di XAML \(WPF\)](#)

[Descrizione dettagliata della sintassi XAML](#)

[Code-behind e XAML in WPF](#)

[Classi XAML e personalizzate per WPF](#)

[Estensioni di markup e XAML WPF](#)

[Spazi dei nomi XAML e mapping dello spazio dei nomi per XAML WPF](#)

[Ambiti dei nomi XAML WPF](#)

[Stili e modelli inline](#)

[Elaborazione di spazi vuoti in XAML](#)

[TypeConverter e XAML](#)

[Entità carattere XML e XAML](#)

[Funzionalità del linguaggio dello spazio dei nomi XAML \(x:\)](#)

[Estensioni XAML WPF](#)

[Funzionalità del linguaggio per la compatibilità dei markup \(mc:\)](#)

Sezioni correlate

[Architettura WPF](#)

[Elementi di base](#)

[Albero degli elementi e serializzazione](#)

[Proprietà](#)

[Eventi](#)

[Input](#)

[Risorse](#)

[Applicazione di stili e modelli](#)

[Modello di threading](#)

2 minutes to read

Descrizione dettagliata della sintassi XAML

12/02/2020 • 54 minutes to read • [Edit Online](#)

In questo argomento vengono definiti i termini utilizzati per descrivere gli elementi della sintassi XAML. Questi termini vengono usati di frequente nel resto della documentazione, sia per la documentazione di WPF in particolare che per gli altri Framework che usano XAML o i concetti di base di XAML abilitati dal supporto del linguaggio XAML a livello di System. XAML. In questo argomento viene illustrata la terminologia di base introdotta nell'argomento [Cenni preliminari su XAML \(WPF\)](#).

Specifiche del linguaggio XAML

La terminologia di sintassi XAML definita qui viene definita o a cui viene fatto riferimento all'interno della specifica del linguaggio XAML. XAML è un linguaggio basato su XML e segue o si espande in base alle regole strutturali XML. Parte della terminologia è condivisa da o è basata sulla terminologia comunemente utilizzata per descrivere il linguaggio XML o il modello a oggetti documento XML.

Per ulteriori informazioni sulla specifica del linguaggio XAML, scaricare [MS-XAML](#) dall'area download Microsoft.

XAML e CLR

XAML è un linguaggio di markup. Il Common Language Runtime (CLR), come sottinteso dal nome, Abilita l'esecuzione del runtime. XAML non è da solo uno dei linguaggi comuni utilizzati direttamente dal runtime CLR. È invece possibile considerare XAML come supporto per il proprio sistema di tipi. Il particolare sistema di analisi XAML utilizzato da WPF si basa su CLR e sul sistema dei tipi CLR. Viene eseguito il mapping dei tipi XAML ai tipi CLR per creare un'istanza di una rappresentazione in fase di esecuzione quando XAML per WPF viene analizzato. Per questo motivo, il resto della discussione sulla sintassi di questo documento includerà riferimenti al sistema di tipi CLR, anche se le discussioni sulla sintassi equivalenti nella specifica del linguaggio XAML non lo sono. (Per il livello di specifica del linguaggio XAML, è possibile eseguire il mapping dei tipi XAML a qualsiasi altro sistema di tipi, che non deve essere CLR, ma che richiederebbe la creazione e l'uso di un parser XAML diverso).

Membri di tipi ed ereditarietà delle classi

Le proprietà e gli eventi come appaiono come membri XAML di un tipo di WPF vengono spesso ereditati dai tipi di base. Si consideri, ad esempio, questo esempio: `<Button Background="Blue" .../>`. La proprietà [Background](#) non è una proprietà dichiarata immediatamente nella classe [Button](#), se si desidera esaminare la definizione della classe, i risultati della reflection o la documentazione. Al contrario, [Background](#) viene ereditato dalla classe [Control](#) di base.

Il comportamento di ereditarietà delle classi di WPF elementi XAML rappresenta una partenza significativa da un'interpretazione applicata dallo schema del markup XML. L'ereditarietà delle classi può diventare complessa, in particolare quando le classi di base intermedie sono astratte o quando sono incluse le interfacce. Questo è un motivo per cui il set di elementi XAML e i relativi attributi consentiti sono difficili da rappresentare in modo accurato e completo utilizzando i tipi di schema utilizzati generalmente per la programmazione XML, ad esempio il formato DTD o XSD. Un altro motivo è che le funzionalità di estendibilità e mapping dei tipi del linguaggio XAML escludono la completezza di qualsiasi rappresentazione fissa dei tipi e dei membri consentiti.

Sintassi degli elementi oggetto

La *sintassi dell'elemento oggetto* è la sintassi di markup XAML che crea un'istanza di una classe o struttura CLR

dichiarando un elemento XML. Questa sintassi è simile alla sintassi degli elementi di altri linguaggi di markup, ad esempio HTML. La sintassi dell'elemento oggetto inizia con una parentesi uncinata aperta (<), seguita immediatamente dal nome del tipo della classe o della struttura di cui viene creata un'istanza. Zero o più spazi possono seguire il nome del tipo e zero o più attributi possono anche essere dichiarati nell'elemento oggetto, con uno o più spazi che separano ogni coppia nome attributo = "valore". Infine, è necessario che venga soddisfatta una delle condizioni seguenti:

- L'elemento e il tag devono essere chiusi da una barra (/) seguita immediatamente da una parentesi uncinata chiusa (>).
- Il tag di apertura deve essere completato da una parentesi uncinata chiusa (>). Altri elementi oggetto, elementi proprietà o testo interno possono seguire il tag di apertura. Il contenuto che può essere contenuto in questo contesto è in genere vincolato dal modello a oggetti dell'elemento. È necessario che esista anche il tag di chiusura equivalente per l'elemento oggetto, nell'annidamento e nel saldo appropriati con altre coppie di tag di apertura e chiusura.

XAML implementato da .NET include un set di regole che mappano gli elementi oggetto in tipi, attributi in proprietà o eventi e spazi dei nomi XAML agli spazi dei nomi CLR e ad assembly. Per WPF e .NET, gli elementi oggetto XAML vengono mappati ai tipi .NET come definito negli assembly a cui si fa riferimento e gli attributi vengono mappati ai membri di tali tipi. Quando si fa riferimento a un tipo CLR in XAML, è possibile accedere anche ai membri ereditati di tale tipo.

Ad esempio, l'esempio seguente è la sintassi dell'elemento oggetto che crea un'istanza di una nuova istanza della classe [Button](#) e specifica anche un attributo [Name](#) e un valore per l'attributo:

```
<Button Name="CheckoutButton"/>
```

Nell'esempio seguente viene illustrata la sintassi dell'elemento oggetto che include anche la sintassi delle proprietà di contenuto XAML. Il testo interno contenuto all'interno di verrà usato per impostare la proprietà di contenuto XAML [TextBox](#), [Text](#).

```
<TextBox>This is a Text Box</TextBox>
```

Modelli di contenuto

Una classe potrebbe supportare un utilizzo come elemento oggetto XAML in termini di sintassi, ma tale elemento funzionerà correttamente solo in un'applicazione o una pagina quando viene posizionata in una posizione prevista di un modello di contenuto o di un albero di elementi generale. Un [MenuItem](#), ad esempio, deve essere in genere inserito solo come elemento figlio di una classe derivata [MenuBase](#), ad esempio [Menu](#). I modelli di contenuto per elementi specifici sono documentati come parte delle osservazioni sulle pagine della classe per i controlli e altre classi WPF che possono essere usate come elementi XAML.

Proprietà degli elementi oggetto

Le proprietà in XAML sono impostate da varie sintassi possibili. La sintassi che può essere usata per una particolare proprietà può variare in base alle caratteristiche del sistema di tipi sottostante della proprietà che si sta impostando.

Impostando i valori delle proprietà, è possibile aggiungere caratteristiche o caratteristiche agli oggetti esistenti nell'oggetto grafico in fase di esecuzione. Lo stato iniziale dell'oggetto creato da un elemento oggetto è basato sul comportamento del costruttore senza parametri. In genere, l'applicazione userà qualcosa di diverso da un'istanza completamente predefinita di un determinato oggetto.

Sintassi per attributi (proprietà)

La sintassi dell'attributo è la sintassi di markup XAML che imposta un valore per una proprietà dichiarando un attributo in un elemento oggetto esistente. Il nome dell'attributo deve corrispondere al nome del membro CLR della proprietà della classe che esegue il backup dell'elemento oggetto pertinente. Il nome dell'attributo è seguito da un operatore di assegnazione (=). Il valore dell'attributo deve essere una stringa racchiusa tra virgolette.

NOTE

È possibile utilizzare le virgolette alternative per inserire una virgoletta letterale all'interno di un attributo. Ad esempio, è possibile usare le virgolette singole come mezzo per dichiarare una stringa che contiene un carattere virgolette doppie al suo interno. Se si utilizzano virgolette singole o doppie, è necessario utilizzare una coppia corrispondente per l'apertura e la chiusura della stringa del valore dell'attributo. Sono disponibili anche sequenze di escape o altre tecniche per aggirare le restrizioni relative ai caratteri imposte da una particolare sintassi XAML. Vedere [entità carattere XML e XAML](#).

Per poter essere impostato tramite la sintassi dell'attributo, una proprietà deve essere pubblica e deve essere scrivibile. Il valore della proprietà nel sistema di tipi di supporto deve essere un tipo di valore o deve essere un tipo di riferimento di cui è possibile creare un'istanza o a cui fa riferimento un processore XAML durante l'accesso al tipo di supporto pertinente.

Per gli eventi XAML WPF, l'evento a cui viene fatto riferimento come nome di attributo deve essere pubblico e avere un delegato pubblico.

La proprietà o l'evento deve essere un membro della classe o della struttura di cui viene creata un'istanza dall'elemento oggetto contenitore.

Elaborazione dei valori di attributo

Il valore stringa contenuto nelle virgolette di apertura e di chiusura viene elaborato da un processore XAML. Per le proprietà, il comportamento di elaborazione predefinito è determinato dal tipo della proprietà CLR sottostante.

Il valore dell'attributo è riempito da uno dei seguenti elementi, usando questo ordine di elaborazione:

1. Se il processore XAML rileva una parentesi graffa o un elemento oggetto che deriva da [MarkupExtension](#), l'estensione di markup a cui viene fatto riferimento viene valutata per prima anziché elaborare il valore come stringa e l'oggetto restituito dall'estensione di markup viene usato come valore. In molti casi l'oggetto restituito da un'estensione di markup sarà un riferimento a un oggetto esistente oppure un'espressione che rinvia la valutazione fino alla fase di esecuzione e non è un nuovo oggetto di cui è stata creata un'istanza.
2. Se la proprietà viene dichiarata con un [TypeConverter](#) con attributi oppure il tipo di valore di tale proprietà viene dichiarato con un [TypeConverter](#) con attributi, il valore stringa dell'attributo viene inviato al convertitore di tipi come input di conversione e il convertitore restituirà una nuova istanza dell'oggetto.
3. Se non è presente alcuna [TypeConverter](#), viene tentata una conversione diretta nel tipo di proprietà. Questo livello finale è una conversione diretta al valore nativo del parser tra i tipi primitivi del linguaggio XAML o un controllo per i nomi delle costanti denominate in unenumerazione (il parser accede quindi ai valori corrispondenti).

Valori degli attributi di enumerazione

Le enumerazioni in XAML vengono elaborate intrinsecamente da parser XAML e i membri di unenumerazione devono essere specificati specificando il nome di stringa di una delle costanti denominate dellenumerazione.

Per i valori di enumerazione non flag, il comportamento nativo consiste nellelaborare la stringa di un valore di attributo e risolverla in uno dei valori di enumerazione. Non si specifica lenumerazione nellenumerazioneformat. *Valore*, come nel codice. Viene invece specificato solo il *valore* e l'enumerazione viene dedotta dal tipo della proprietà che si sta impostando. Se si specifica un attributo nellenumerazione. Il formato del *valore* non verrà risolto correttamente.

Per le enumerazioni flag, il comportamento è basato sul metodo [Enum.Parse](#). È possibile specificare più valori per un'enumerazione flag separando ogni valore con una virgola. Tuttavia, non è possibile combinare i valori di enumerazione che non sono flag. Ad esempio, non è possibile usare la sintassi della virgola per tentare di creare un [Trigger](#) che agisce su più condizioni di un'enumerazione non flag:

```
<!--This will not compile, because Visibility is not a flagwise enumeration.-->
...
<Trigger Property="Visibility" Value="Collapsed,Hidden">
    <Setter ... />
</Trigger>
...
```

Le enumerazioni flag che supportano attributi che possono essere impostati in XAML sono rare in WPF. Tuttavia, una tale enumerazione viene [StyleSimulations](#). È possibile, ad esempio, usare la sintassi dell'attributo flag delimitato da virgolette per modificare l'esempio fornito nei commenti per la classe [Glyphs](#).

`StyleSimulations = "BoldSimulation"` potrebbe diventare `StyleSimulations = "BoldSimulation,ItalicSimulation"`. [KeyBinding.Modifiers](#) è un'altra proprietà in cui è possibile specificare più di un valore di enumerazione. Tuttavia, questa proprietà si verifica come un caso speciale, perché l'enumerazione [ModifierKeys](#) supporta il proprio convertitore di tipi. Il convertitore di tipi per i modificatori utilizza un segno più (+) come delimitatore anziché una virgola (.). Questa conversione supporta la sintassi più tradizionale per rappresentare le combinazioni di tasti nella programmazione di Microsoft Windows, ad esempio "CTRL + ALT".

Riferimenti alle proprietà e al nome del membro evento

Quando si specifica un attributo, è possibile fare riferimento a qualsiasi proprietà o evento esistente come membro del tipo CLR di cui è stata creata un'istanza per l'elemento oggetto contenitore.

In alternativa, è possibile fare riferimento a una proprietà associata o a un evento associato, indipendentemente dall'elemento oggetto contenitore. (Le proprietà associate sono descritte in una sezione imminente).

È anche possibile assegnare un nome a qualsiasi evento da qualsiasi oggetto accessibile tramite lo spazio dei nomi predefinito usando un *typeName*. nome parzialmente qualificato dell' *evento* ; Questa sintassi supporta il collegamento dei gestori per gli eventi indirizzati in cui il gestore è progettato per gestire il routing degli eventi dagli elementi figlio, ma anche l'elemento padre non dispone di tale evento nella tabella dei membri. Questa sintassi è simile alla sintassi di un evento associato, ma l'evento non è un vero evento associato. Viene invece fatto riferimento a un evento con un nome completo. Per ulteriori informazioni, vedere [Cenni preliminari sugli eventi indirizzati](#).

Per alcuni scenari, i nomi delle proprietà vengono talvolta forniti come valore di un attributo, anziché come nome dell'attributo. Il nome della proprietà può includere anche i qualificatori, ad esempio la proprietà specificata nel formato *tipoProprietario.dependencypropertyname*. Questo scenario è comune durante la scrittura di stili o modelli in XAML. Le regole di elaborazione per i nomi di proprietà fornite come valore di attributo sono diverse e sono regolate dal tipo della proprietà impostata o dai comportamenti di particolari sottosistemi WPF. Per informazioni dettagliate, vedere applicazione di [stili e modelli](#).

Un altro utilizzo per i nomi delle proprietà è quando un valore di attributo descrive una relazione tra proprietà della proprietà. Questa funzionalità viene usata per data binding e per le destinazioni storyboard ed è abilitata dalla classe [PropertyPath](#) e dal relativo convertitore di tipi. Per una descrizione più completa della semantica di ricerca, vedere [sintassi XAML di PropertyPath](#).

Sintassi per gli elementi proprietà

La *sintassi dell'elemento Property* è una sintassi che diverge in parte dalle regole di base della sintassi XML per gli elementi. In XML il valore di un attributo è una stringa de facto, con la sola variazione possibile che indica il formato di codifica delle stringhe utilizzato. In XAML è possibile assegnare altri elementi oggetto come valore di una proprietà. Questa funzionalità è abilitata dalla sintassi dell'elemento [Property](#). Anziché la proprietà specificata

come attributo all'interno del tag dell'elemento, la proprietà viene specificata utilizzando un tag di elemento di apertura in *nomeTipoElemento*. *PropertyName* form, il valore della proprietà viene specificato in e quindi l'elemento Property è Closed.

In particolare, la sintassi inizia con una parentesi uncinata aperta (<), seguita immediatamente dal nome del tipo della classe o della struttura in cui è contenuta la sintassi dell'elemento Property. Questa operazione è seguita immediatamente da un singolo punto (.), quindi dal nome di una proprietà, quindi da una parentesi uncinata chiusa (>). Come per la sintassi degli attributi, tale proprietà deve esistere all'interno dei membri pubblici dichiarati del tipo specificato. Il valore da assegnare alla proprietà è contenuto all'interno dell'elemento Property. In genere, il valore viene fornito come uno o più elementi oggetto, perché la specifica di oggetti come valori è lo scenario per cui la sintassi dell'elemento proprietà è destinata all'indirizzo. Infine, un tag di chiusura equivalente che specifica lo stesso *nomeTipoElemento*. È necessario specificare la combinazione *PropertyName*, in una nidificazione e un bilanciamento appropriati con altri tag di elemento.

Ad esempio, di seguito è riportata la sintassi dell'elemento Property per la proprietà [ContextMenu](#) di un [Button](#).

```
<Button>
  <Button.ContextMenu>
    <ContextMenu>
      <MenuItem Header="1">First item</MenuItem>
      <MenuItem Header="2">Second item</MenuItem>
    </ContextMenu>
  </Button.ContextMenu>
  Right-click me!</Button>
```

Il valore all'interno di un elemento Property può essere fornito anche come testo interno, nei casi in cui il tipo di proprietà specificato è un tipo di valore primitivo, ad esempio [String](#), oppure un'enumerazione in cui viene specificato un nome. Questi due utilizzi sono alquanto rari, perché ognuno di questi casi può utilizzare anche una sintassi di attributo più semplice. Uno scenario per la compilazione di un elemento Property con una stringa è per le proprietà che non sono la proprietà di contenuto XAML, ma vengono comunque usate per la rappresentazione del testo dell'interfaccia utente e gli elementi di spazi vuoti specifici, ad esempio avanzamento riga, devono essere visualizzati nel testo dell'interfaccia utente. La sintassi dell'attributo non può mantenere avanzamento riga, ma la sintassi dell'elemento proprietà può, a condizione che la conservazione significativa degli spazi vuoti sia attiva (per informazioni dettagliate, vedere [elaborazione di spazi vuoti in XAML](#)). Un altro scenario è la possibilità di applicare la [direttiva x:UID](#) all'elemento Property, contrassegnando quindi il valore in come valore che deve essere localizzato nel BAML di output WPF o con altre tecniche.

Un elemento Property non è rappresentato nell'albero logico WPF. Un elemento Property è solo una particolare sintassi per l'impostazione di una proprietà e non è un elemento con un'istanza o un oggetto sottostante. Per informazioni dettagliate sul concetto di albero logico, vedere [strutture ad albero in WPF](#).

Per le proprietà in cui sono supportate sia la sintassi degli elementi di attributo che di proprietà, le due sintassi presentano in genere lo stesso risultato, sebbene le sottiliezzze, ad esempio la gestione degli spazi vuoti, possano variare leggermente tra le sintassi.

Sintassi per raccolte

La specifica XAML richiede implementazioni del processore XAML per identificare le proprietà in cui il tipo di valore è una raccolta. L'implementazione generale del processore XAML in .NET è basata sul codice gestito e su CLR e identifica i tipi di raccolta tramite uno dei seguenti elementi:

- Il tipo implementa [IList](#).
- Il tipo implementa [IDictionary](#).
- Il tipo deriva da [Array](#) (per altre informazioni sulle matrici in XAML, vedere [estensione di markup x:Array](#)).

Se il tipo di una proprietà è una raccolta, il tipo di raccolta derivato non deve essere specificato nel markup come elemento oggetto. Gli elementi destinati a diventare gli elementi della raccolta vengono invece specificati come uno o più elementi figlio dell'elemento `Property`. Ogni elemento di questo tipo viene valutato a un oggetto durante il caricamento e aggiunto alla raccolta chiamando il metodo `Add` della raccolta implicita. Ad esempio, la proprietà `Triggers` di `Style` accetta il tipo di raccolta specializzato `TriggerCollection`, che implementa `IList`. Non è necessario creare un'istanza di un `TriggerCollection` elemento oggetto nel markup. Al contrario, è necessario specificare uno o più elementi `Trigger` come elementi all'interno dell'elemento proprietà `Style.Triggers`, in cui `Trigger` (o una classe derivata) è il tipo previsto come tipo di elemento per la `TriggerCollection` fortemente tipizzata e implicita.

```
<Style x:Key="SpecialButton" TargetType="{x:Type Button}">
  <Style.Triggers>
    <Trigger Property="Button.IsMouseOver" Value="true">
      <Setter Property = "Background" Value="Red"/>
    </Trigger>
    <Trigger Property="Button.IsPressed" Value="true">
      <Setter Property = "Foreground" Value="Green"/>
    </Trigger>
  </Style.Triggers>
</Style>
```

Una proprietà può essere sia un tipo di raccolta che la proprietà di contenuto XAML per il tipo e i tipi derivati, descritti nella sezione successiva di questo argomento.

Un elemento della raccolta implicita crea un membro nella rappresentazione ad albero logico, anche se non viene visualizzato nel markup come elemento. In genere, il costruttore del tipo padre esegue la creazione di un'istanza per la raccolta che è una delle relative proprietà e la raccolta inizialmente vuota diventa parte dell'albero degli oggetti.

NOTE

Le interfacce di elenco e dizionario generico (`IList<T>` e `IDictionary< TKey, TValue >`) non sono supportate per il rilevamento della raccolta. Tuttavia, è possibile usare la classe `List<T>` come classe base, perché implementa direttamente `IList` o `Dictionary< TKey, TValue >` come classe di base, perché implementa `IDictionary` direttamente.

Nelle pagine di riferimento .NET per i tipi di raccolta, questa sintassi con l'omissione intenzionale dell'elemento oggetto per una raccolta viene occasionalmente indicata nelle sezioni della sintassi XAML come sintassi di raccolta implicita.

Fatta eccezione per l'elemento radice, ogni elemento oggetto in un file XAML annidato come elemento figlio di un altro elemento è effettivamente un elemento che è uno o entrambi i casi seguenti: un membro di una proprietà di raccolta implicita del relativo elemento padre, o un elemento che specifica il valore della proprietà di contenuto XAML per l'elemento padre (le proprietà del contenuto XAML verranno discusse in una sezione futura). In altre parole, la relazione degli elementi padre e degli elementi figlio in una pagina di markup è effettivamente un singolo oggetto alla radice e ogni elemento oggetto sotto la radice è una singola istanza che fornisce un valore della proprietà dell'elemento padre o uno degli elementi all'interno di un col coltato che è anche un valore della proprietà del tipo di raccolta dell'elemento padre. Questo concetto a radice singola è comune con XML e viene spesso rinforzato nel comportamento delle API che caricano XAML, ad esempio `Load`.

L'esempio seguente è una sintassi con l'elemento oggetto per una raccolta (`GradientStopCollection`) specificata in modo esplicito.

```
<LinearGradientBrush>
  <LinearGradientBrush.GradientStops>
    <GradientStopCollection>
      <GradientStop Offset="0.0" Color="Red" />
      <GradientStop Offset="1.0" Color="Blue" />
    </GradientStopCollection>
  </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

Si noti che non è sempre possibile dichiarare esplicitamente la raccolta. Ad esempio, il tentativo di dichiarare [TriggerCollection](#) in modo esplicito nell'esempio di [Triggers](#) illustrato in precedenza avrebbe esito negativo. Per dichiarare in modo esplicito la raccolta, è necessario che la classe della raccolta supporti un costruttore senza parametri e [TriggerCollection](#) non disponga di un costruttore senza parametri.

Proprietà di contenuto XAML

La sintassi del contenuto XAML è una sintassi abilitata solo nelle classi che specificano il [ContentPropertyAttribute](#) come parte della dichiarazione di classe. Il [ContentPropertyAttribute](#) fa riferimento al nome della proprietà che è la proprietà di contenuto per quel tipo di elemento (incluse le classi derivate). Quando vengono elaborati da un processore XAML, qualsiasi elemento figlio o testo interno trovato tra i tag di apertura e di chiusura dell'elemento oggetto verrà assegnato come valore della proprietà di contenuto XAML per tale oggetto. È possibile specificare elementi della proprietà esplicativi per la proprietà Content, ma questo utilizzo non viene in genere visualizzato nelle sezioni della sintassi XAML di .NET Reference. La tecnica esplicita/dettagliata presenta un valore occasionale per la chiarezza del markup o come una questione di stile di markup, ma in genere lo scopo di una proprietà di contenuto consiste nel semplificare il markup in modo che gli elementi che sono intuitivamente correlati come padre-figlio possano essere annidati direttamente. I tag dell'elemento proprietà per altre proprietà di un elemento non sono assegnati come "contenuto" per una definizione di linguaggio XAML Strict; vengono elaborati in precedenza nell'ordine di elaborazione del parser XAML e non sono considerati "Content".

I valori delle proprietà di contenuto XAML devono essere contigui

Il valore di una proprietà di contenuto XAML deve essere specificato interamente prima o completamente dopo qualsiasi altro elemento di proprietà nell'elemento oggetto. Questo vale se il valore di una proprietà di contenuto XAML viene specificato come stringa o come uno o più oggetti. Il markup seguente, ad esempio, non analizza:

```
<Button>I am a
<Button.Background>Blue</Button.Background>
blue button</Button>
```

Questa operazione non è sostanzialmente valida perché se questa sintassi è stata resa esplicita usando la sintassi dell'elemento Property per la proprietà Content, la proprietà Content verrebbe impostata due volte:

```
<Button>
  <Button.Content>I am a </Button.Content>
  <Button.Background>Blue</Button.Background>
  <Button.Content> blue button</Button.Content>
</Button>
```

Un esempio non valido è analogo a se la proprietà Content è una raccolta e gli elementi figlio vengono sparpagliati con gli elementi Property:

```

<StackPanel>
    <Button>This example</Button>
    <StackPanel.Resources>
        <SolidColorBrush x:Key="BlueBrush" Color="Blue"/>
    </StackPanel.Resources>
    <Button>... is illegal XAML</Button>
</StackPanel>

```

Combinazione di proprietà di contenuto e sintassi per raccolte

Per accettare più di un singolo elemento oggetto come contenuto, il tipo della proprietà di contenuto deve essere specificamente un tipo di raccolta. Analogamente alla sintassi dell'elemento `Property` per i tipi di raccolta, un processore XAML deve identificare i tipi che sono tipi di raccolta. Se un elemento ha una proprietà di contenuto XAML e il tipo della proprietà di contenuto XAML è una raccolta, non è necessario specificare il tipo di raccolta implicito nel markup come elemento oggetto e non è necessario specificare la proprietà di contenuto XAML come proprietà `El autorità`. Il modello di contenuto apparente nel markup ora può quindi avere più di un elemento figlio assegnato come contenuto. Di seguito è riportata la sintassi del contenuto per un `Panel` classe derivata. Tutte `Panel` classi derivate stabiliscono la proprietà di contenuto XAML da `Children`, che richiede un valore di tipo `UIElementCollection`.

```

<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    >
    <StackPanel>
        <Button>Button 1</Button>
        <Button>Button 2</Button>
        <Button>Button 3</Button>
    </StackPanel>
</Page>

```

Si noti che nell'markup non è necessario né l'elemento `Property` per `Children` né l'elemento per l'`UIElementCollection`. Si tratta di una funzionalità di progettazione di XAML in modo che gli elementi contenuti in modo ricorsivo che definiscono un Interfaccia utente siano rappresentati in modo più intuitivo come albero di elementi annidati con relazioni immediate tra gli elementi padre-figlio, senza tag dell'elemento proprietà o oggetti raccolta. In realtà, non è possibile specificare in modo esplicito `UIElementCollection` nel markup come elemento oggetto, da progettazione. Poiché l'unico utilizzo previsto è come una raccolta implicita, `UIElementCollection` non espone un costruttore pubblico senza parametri e pertanto non è possibile creare un'istanza come elemento oggetto.

Combinazione di elementi di proprietà ed elementi oggetto in un oggetto con una proprietà Content

La specifica XAML dichiara che un processore XAML può applicare che gli elementi oggetto usati per riempire la proprietà di contenuto XAML all'interno di un elemento oggetto devono essere contigui e non devono essere combinati. Questa restrizione sulla combinazione di elementi e contenuto della proprietà viene applicata dal WPF processori XAML.

È possibile avere un elemento oggetto figlio come primo markup immediato all'interno di un elemento oggetto. È quindi possibile introdurre elementi della proprietà. In alternativa, è possibile specificare uno o più elementi `Property`, quindi `Content`, quindi più elementi `Property`. Tuttavia, una volta che un elemento `Property` segue il contenuto, non è possibile introdurre altri contenuti, è possibile aggiungere solo elementi `Property`.

Questo requisito dell'ordine degli elementi di contenuto/proprietà non si applica al testo interno usato come contenuto. Tuttavia, è ancora un valido stile di markup per mantenere contiguo il testo interno, perché gli spazi significativi saranno difficili da rilevare visivamente nel markup se gli elementi della proprietà vengono sparpagliati con testo interno.

Spazi dei nomi XAML

Nessuno degli esempi di sintassi precedenti ha specificato uno spazio dei nomi XAML diverso dallo spazio dei nomi XAML predefinito. Nelle applicazioni WPF tipiche lo spazio dei nomi XAML predefinito viene specificato come spazio dei nomi WPF. È possibile specificare gli spazi dei nomi XAML diversi dallo spazio dei nomi XAML predefinito e usare comunque una sintassi simile. Tuttavia, in qualsiasi punto in cui una classe è denominata non accessibile nello spazio dei nomi XAML predefinito, il nome della classe deve essere preceduto dal prefisso dello spazio dei nomi XAML come mappato allo spazio dei nomi CLR corrispondente. Ad esempio, `<custom:Example/>` è la sintassi dell'elemento oggetto per creare un'istanza della classe `Example`, in cui lo spazio dei nomi CLR contenente tale classe (e possibilmente le informazioni sull'assembly esterno che contiene i tipi di supporto) è stato precedentemente mappato al prefisso `custom`.

Per altre informazioni sugli spazi dei nomi XAML, vedere [spazi dei nomi XAML e mapping dello spazio dei nomi per XAML WPF](#).

Estensioni di markup

XAML definisce un'entità di programmazione dell'estensione di markup che consente di eseguire l'escape dalla normale gestione del processore XAML dei valori di attributo stringa o degli elementi oggetto e rinvia l'elaborazione a una classe sottostante. Il carattere che identifica un'estensione di markup a un processore XAML quando si usa la sintassi dell'attributo è la parentesi graffa aperta (`{`}), seguita da qualsiasi carattere diverso da una parentesi graffa chiusa (`}`). La prima stringa che segue la parentesi graffa aperta deve fare riferimento alla classe che fornisce il comportamento di estensione specifico, in cui il riferimento può omettere la sottostringa "extension" se tale sottostringa fa parte del nome della classe true. Successivamente, è possibile che venga visualizzato un solo spazio, quindi ogni carattere successivo viene usato come input dall'implementazione dell'estensione, fino a quando non viene rilevata la parentesi graffa di chiusura.

Nell'implementazione XAML di .NET viene utilizzata la `MarkupExtension` classe astratta come base per tutte le estensioni di markup supportate da WPF, nonché da altri Framework o tecnologie. Le estensioni di markup che WPF implementano in modo specifico sono spesso destinate a fornire un mezzo per fare riferimento ad altri oggetti esistenti o per creare riferimenti posticipati a oggetti che verranno valutati in fase di esecuzione. Ad esempio, una semplice data binding WPF viene eseguita specificando l'estensione di markup `{Binding}` al posto del valore che una particolare proprietà richiederebbe normalmente. Molte estensioni di markup WPF consentono la sintassi degli attributi per le proprietà in cui la sintassi di un attributo non sarebbe altrimenti possibile. Un oggetto `Style`, ad esempio, è un tipo relativamente complesso che contiene una serie annidata di oggetti e proprietà. Gli stili in WPF vengono in genere definiti come una risorsa in una `ResourceDictionary` quindi fanno riferimento a una delle due estensioni di markup WPF che richiedono una risorsa. L'estensione di markup rinvia la valutazione del valore della proprietà a una ricerca di risorse e consente di fornire il valore della proprietà `Style`, accettando il tipo `Style`, nella sintassi dell'attributo come nell'esempio seguente:

```
<Button Style="{StaticResource MyStyle}">My button</Button>
```

`StaticResource` identifica la classe `StaticResourceExtension` che fornisce l'implementazione dell'estensione di markup. La stringa successiva `MyStyle` viene utilizzata come input per il costruttore di `StaticResourceExtension` non predefinito, in cui il parametro come tratto dalla stringa di estensione dichiara il `ResourceKey` richiesto. `MyStyle` deve essere il valore `x:Key` di un `Style` definito come risorsa. L'utilizzo dell'[estensione di markup StaticResource](#) richiede che la risorsa venga utilizzata per fornire il valore della proprietà `Style` tramite la logica di ricerca di risorse statiche in fase di caricamento.

Per altre informazioni sulle estensioni di markup, vedere [Estensioni di markup e WPF XAML](#). Per un riferimento alle estensioni di markup e ad altre funzionalità di programmazione XAML abilitate nell'implementazione generale di XAML .NET, vedere [spazio dei nomi XAML \(x:\) Funzionalità del linguaggio](#). Per le estensioni di markup specifiche di WPF, vedere [estensioni XAML WPF](#).

Proprietà associate

Le proprietà associate sono un concetto di programmazione introdotto in XAML in cui le proprietà possono essere di proprietà e definite da un particolare tipo, ma impostate come attributi o elementi di proprietà su qualsiasi elemento. Lo scenario principale a cui sono destinate le proprietà è quello di consentire agli elementi figlio in una struttura di markup di segnalare le informazioni a un elemento padre senza richiedere un modello a oggetti ampiamente condiviso in tutti gli elementi. Viceversa, le proprietà associate possono essere utilizzate dagli elementi padre per segnalare le informazioni agli elementi figlio. Per altre informazioni sullo scopo delle proprietà associate e su come creare le proprie proprietà associate, vedere [Cenni preliminari sulle proprietà associate](#).

Le proprietà associate utilizzano una sintassi che è simile alla sintassi degli elementi di proprietà, in quanto si specifica anche un *typeName*. combinazione *PropertyName*. Vi sono due differenze importanti:

- È possibile utilizzare il *typeName*. combinazione *PropertyName* anche quando si imposta una proprietà associata tramite la sintassi dell'attributo. Le proprietà associate sono l'unico caso in cui qualificare il nome della proprietà è un requisito in una sintassi di attributo.
- È inoltre possibile utilizzare la sintassi dell'elemento proprietà per le proprietà associate. Tuttavia, per la sintassi tipica degli elementi proprietà, il *typeName* specificato è l'elemento oggetto che contiene l'elemento *Property*. Se si fa riferimento a una proprietà associata, *typeName* è la classe che definisce la proprietà associata, non l'elemento oggetto contenitore.

Eventi associati

Gli eventi associati sono un altro concetto di programmazione introdotto in XAML in cui gli eventi possono essere definiti da un tipo specifico, ma i gestori possono essere collegati a qualsiasi elemento oggetto.

Nell'implementazione di WOF, spesso il tipo che definisce un evento associato è un tipo statico che definisce un servizio e talvolta gli eventi collegati vengono esposti da un alias di evento indirizzato nei tipi che espongono il servizio. I gestori per gli eventi associati vengono specificati tramite la sintassi dell'attributo. Come per gli eventi associati, la sintassi dell'attributo viene espansa per gli eventi associati per consentire un *typeName*. utilizzo di *EventName*, dove *typeName* è la classe che fornisce le funzioni di accesso del gestore eventi `Add` e `Remove` per l'infrastruttura di eventi collegati e *EventName* è il nome dell'evento.

Anatomia di un elemento radice XAML

La tabella seguente illustra un tipico elemento radice XAML suddiviso, che Mostra gli attributi specifici di un elemento radice:

<code><Page</code>	Apertura dell'elemento oggetto dell'elemento radice
<code>xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"</code>	Spazio dei nomi XAML predefinito (WPF)
<code>xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"</code>	Spazio dei nomi XAML del linguaggio XAML
<code>x:Class="ExampleNamespace.ExampleCode"</code>	Dichiarazione di classe parziale che connette il markup a qualsiasi code-behind definito per la classe parziale
<code>></code>	Fine dell'elemento oggetto per la radice. L'oggetto non è ancora chiuso perché l'elemento contiene elementi figlio

Utilizzi XAML facoltativi e non consigliati

Le sezioni seguenti descrivono gli utilizzi XAML tecnicamente supportati dai processori XAML, ma che producono un livello di dettaglio o altri problemi estetici che interferiscono con i file XAML rimanenti leggibili quando si sviluppano applicazioni che contengono origini XAML.

Utilizzi facoltativi degli elementi proprietà

Gli utilizzi facoltativi degli elementi proprietà includono la scrittura esplicita delle proprietà del contenuto dell'elemento che il processore XAML considera implicite. Ad esempio, quando si dichiara il contenuto di una [Menu](#), è possibile scegliere di dichiarare in modo esplicito la raccolta di [Items](#) del [Menu](#) come tag dell'elemento proprietà `<Menu.Items>` e inserire ogni [MenuItem](#) all'interno `<Menu.Items>`, anziché usare il comportamento del processore XAML implicito che tutti gli elementi figlio di un [Menu](#) devono essere un [MenuItem](#) e vengono inseriti nella raccolta di [Items](#). Talvolta gli utilizzi facoltativi possono aiutare a chiarire visivamente la struttura dell'oggetto come rappresentata nel markup. O a volte un utilizzo esplicito dell'elemento proprietà può evitare markup tecnicamente funzionale, ma visivamente confuso, ad esempio estensioni di markup annidate all'interno di un valore di attributo.

Attributi completi `typeName`. `MemberName`

`TypeName`. il formato `memberName` per un attributo funziona in modo più universale rispetto al solo caso di evento indirizzato. In altre situazioni, tuttavia, il form è superfluo ed è consigliabile evitarlo, se solo per motivi di stile di markup e leggibilità. Nell'esempio seguente ognuno dei tre riferimenti all'attributo [Background](#) è completamente equivalente:

```
<Button Background="Blue">Background</Button>
<Button Button.Background="Blue">Button.Background</Button>
<Button Control.Background="Blue">Control.Background</Button>
```

`Button.Background` funziona perché la ricerca qualificata per la proprietà in [Button](#) ha esito positivo ([Background](#) è stata ereditata da [Control](#)) e [Button](#) è la classe dell'elemento oggetto o di una classe di base.

`Control.Background` funziona perché la classe [Control](#) definisce effettivamente [Background](#) e [Control](#) è una classe di base [Button](#).

Tuttavia, il `typeNames` seguente. L'esempio di form `memberName` non funziona e viene quindi visualizzato come commento:

```
<!--<Button Label.Background="Blue">Does not work</Button> -->
```

[Label](#) è un'altra classe derivata di [Control](#) se è stato specificato `Label.Background` all'interno di un elemento [Label](#) oggetto, l'utilizzo avrebbe avuto esito positivo. Tuttavia, poiché [Label](#) non è la classe o la classe base di [Button](#), il comportamento del processore XAML specificato consiste nel elaborare `Label.Background` come proprietà associata. `Label.Background` non è una proprietà associata disponibile e questo utilizzo non riesce.

Elementi della proprietà `nomeTipoBase`. `MemberName`

In modo analogo a come il `typeName`. il form `membrontename` funziona per la sintassi degli attributi, un `nomeTipoBase`. la sintassi `memberName` funziona per la sintassi dell'elemento [Property](#). Ad esempio, la sintassi seguente funziona:

```
<Button>Control.Background PE
<Control.Background>
  <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
    <GradientStop Color="Yellow" Offset="0.0" />
    <GradientStop Color="LimeGreen" Offset="1.0" />
  </LinearGradientBrush>
</Control.Background>
</Button>
```

In questo caso, l'elemento Property è stato fornito come `Control.Background` anche se l'elemento Property era contenuto nell'`Button`.

Ma proprio come *typeName.memberName* form per gli attributi, *nomeTipoBase.memberName* è uno stile insufficiente nel markup ed è consigliabile evitarlo.

Vedere anche

- [Panoramica di XAML \(WPF\)](#)
- [Funzionalità del linguaggio dello spazio dei nomi XAML \(x:\)](#)
- [Estensioni XAML WPF](#)
- [Cenni preliminari sulle proprietà di dipendenza](#)
- [TypeConverter e XAML](#)
- [Classi XAML e personalizzate per WPF](#)

Code-behind e XAML in WPF

03/02/2020 • 7 minutes to read • [Edit Online](#)

Code-behind è un termine usato per descrivere il codice Unito a oggetti definiti dal markup, quando una pagina di XAML viene compilata con markup. In questo argomento vengono descritti i requisiti per il code-behind e un meccanismo alternativo di codice inline per il codice in XAML.

Questo argomento è suddiviso nelle sezioni seguenti:

- [Prerequisiti](#)
- [Code-behind e linguaggio XAML](#)
- [Code-behind, gestore eventi e requisiti della classe parziale in WPF](#)
- [x:Code](#)
- [Limitazioni del codice inline](#)

Prerequisiti

In questo argomento si presuppone che sia stata letta la [Panoramica di XAML \(WPF\)](#) e che si disponga di una conoscenza di base di CLR e della programmazione orientata a oggetti.

Code-behind e linguaggio XAML

Il linguaggio XAML include funzionalità a livello di linguaggio che consentono di associare i file di codice ai file di markup, dal lato del file di markup. In particolare, il linguaggio XAML definisce le funzionalità del linguaggio [X:Class Directive](#), [X:Subclass Directive](#) e [x:ClassModifier Directive](#). Il modo esatto in cui il codice deve essere prodotto e come integrare markup e codice non fa parte del linguaggio XAML specificato. Viene lasciato a Framework come WPF per determinare come integrare il codice, come usare XAML nei modelli di applicazione e di programmazione e le azioni di compilazione o altro supporto richiesto da tutti.

Code-behind, gestore eventi e requisiti della classe parziale in WPF

- La classe parziale deve derivare dal tipo che esegue il backup dell'elemento radice.
- Si noti che, sotto il comportamento predefinito delle azioni di compilazione di compilazione del markup, è possibile lasciare vuota la derivazione nella definizione della classe parziale sul lato code-behind. Il risultato compilato presuppone che il tipo di supporto della radice della pagina sia la base per la classe parziale, anche se non è specificato. Tuttavia, basarsi su questo comportamento non è una procedura consigliata.
- I gestori di eventi scritti nel code-behind devono essere metodi di istanza e non possono essere metodi statici. Questi metodi devono essere definiti dalla classe parziale nello spazio dei nomi CLR identificato da [x:Class](#). Non è possibile qualificare il nome di un gestore eventi per indicare a un XAML processore di cercare un gestore eventi per il cablaggio degli eventi in un ambito di classe diverso.
- Il gestore deve corrispondere al delegato per l'evento appropriato nel sistema di tipi di supporto.
- Per il linguaggio di Visual Basic Microsoft, è possibile usare la parola chiave [Handles](#) specifica del linguaggio per associare i gestori a istanze ed eventi nella dichiarazione del gestore, anziché collegare i gestori con attributi in XAML. Tuttavia, questa tecnica presenta alcune limitazioni perché la parola chiave [Handles](#) non supporta tutte le funzionalità specifiche del sistema di WPF eventi, ad esempio alcuni scenari

di eventi indirizzati o eventi associati. Per informazioni dettagliate, vedere [Visual Basic e gestione degli eventi WPF](#).

x:Code

[x:code](#) è un elemento di direttiva definito in XAML. Un elemento `x:Code` direttiva può contenere codice di programmazione inline. Il codice definito inline può interagire con il XAML nella stessa pagina. Nell'esempio seguente viene illustrato il codice C# inline. Si noti che il codice si trova all'interno dell'elemento `x:Code` e che il codice deve essere racchiuso tra `<![CDATA[...]]>` per eseguire l'escape del contenuto per XML, in modo che un processore di XAML, interpretando lo schema di XAML o lo schema di WPF, non tenti di interpretare il contenuto letteralmente come XML.

```
<Page
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="MyNamespace.MyCanvasCodeInline"
>
  <Button Name="button1" Click="Clicked">Click Me!</Button>
  <x:Code><![CDATA[
    void Clicked(object sender, RoutedEventArgs e)
    {
      button1.Content = "Hello World";
    }
  ]]></x:Code>
</Page>
```

Limitazioni del codice inline

È consigliabile evitare o limitare l'utilizzo del codice inline. In termini di architettura e filosofia di codifica, mantenere una separazione tra markup e code-behind mantiene i ruoli di progettazione e sviluppo molto più distinti. A un livello più tecnico, il codice che si scrive per il codice inline può essere scomodo da scrivere, perché si sta scrivendo sempre in XAML classe parziale generata e può utilizzare solo i mapping degli spazi dei nomi XML predefiniti. Poiché non è possibile aggiungere istruzioni `using`, è necessario qualificare in modo completo molte delle chiamate API effettuate. I mapping WPF predefiniti includono la maggior parte degli spazi dei nomi CLR presenti negli assembly di WPF. sarà necessario qualificare completamente le chiamate ai tipi e ai membri contenuti negli altri spazi dei nomi CLR. Non è inoltre possibile definire alcunché oltre la classe parziale nel codice inline e tutte le entità del codice utente a cui si fa riferimento devono esistere come membro o variabile all'interno della classe parziale generata. Non sono inoltre disponibili altre funzionalità di programmazione specifiche del linguaggio, ad esempio macro o `#ifdef` su variabili globali o variabili di compilazione. Per altre informazioni, vedere [tipo XAML intrinseco x:code](#).

Vedere anche

- [Panoramica di XAML \(WPF\)](#)
- [Tipo XAML intrinseco x:Code](#)
- [Compilazione di un'applicazione WPF](#)
- [Descrizione dettagliata della sintassi XAML](#)

Classi XAML e personalizzate per WPF

03/02/2020 • 23 minutes to read • [Edit Online](#)

XAML come implementato nei Framework Common Language Runtime (CLR) supporta la possibilità di definire una classe o una struttura personalizzata in qualsiasi linguaggio Common Language Runtime (CLR), quindi accedere a tale classe usando il markup XAML. All'interno dello stesso file di markup è possibile usare una combinazione di tipi definiti da Windows Presentation Foundation (WPF) e di tipi personalizzati, in genere tramite mapping dei tipi personalizzati al prefisso di uno spazio dei nomi XAML. Questo argomento descrive i requisiti che una classe personalizzata deve soddisfare perché sia utilizzabile come elemento XAML.

Classi personalizzate in applicazioni o assembly

È possibile definire classi personalizzate da usare in XAML in due modi distinti: all'interno del code-behind o di altro codice che generi l'applicazione Windows Presentation Foundation (WPF) primaria o come classe di un assembly separato, ad esempio un eseguibile o una DLL usata come libreria di classi. Ognuno di questi approcci presenta vantaggi e svantaggi specifici.

- Il vantaggio della creazione di una libreria di classi consiste nella possibilità di condividere tutte queste classi personalizzate tra un gran numero di applicazioni diverse. Una libreria separata, poi, rende più semplice il controllo delle versioni delle applicazioni in caso di problemi e facilita la creazione di classi destinate a fungere da elementi radice di una pagina XAML.
- Il vantaggio di poter definire classi personalizzate all'interno dell'applicazione consiste nel fatto che si tratta di una tecnica relativamente leggera in grado di ridurre al minimo i problemi di distribuzione e test che si riscontrano quando, oltre al file eseguibile dell'applicazione principale, si introducono assembly separati.
- Che le classi personalizzate vengano definite all'interno dello stesso assembly o in un assembly diverso, perché siano utilizzabili come elementi in XAML è necessario effettuarne il mapping tra lo spazio dei nomi CLR e quello XML. Vedere [Spazi dei nomi XAML e mapping dello spazio dei nomi per XAML WPF](#).

Requisiti per una classe personalizzata come elemento XAML

Perché sia possibile creare un'istanza come elemento oggetto, la classe deve soddisfare i requisiti seguenti:

- Deve essere pubblica e supportare un costruttore pubblico senza parametri predefinito. Per alcune note riguardanti le strutture, vedere la sezione seguente.
- La classe personalizzata non deve essere una classe annidata. Le classi annidate e il punto presente nella relativa sintassi di utilizzo CLR generale interferiscono con altre funzionalità di WPF e/o di XAML, ad esempio le proprietà collegate.

Oltre ad abilitare la sintassi degli elementi oggetto, la definizione dell'oggetto abilita la sintassi degli elementi proprietà per tutte le altre proprietà pubbliche che accettano tale oggetto come tipo valore. Ciò è dovuto al fatto che è ora possibile creare un'istanza dell'oggetto come elemento oggetto e inserirla come valore dell'elemento di tale proprietà.

Strutture

Le strutture definite come tipi personalizzati sono sempre in grado di essere costruite in XAML in WPF. Ciò è dovuto al fatto che i compilatori CLR creano implicitamente un costruttore senza parametri per una struttura che inizializza i valori predefiniti di tutti i valori delle proprietà. In alcuni casi il comportamento predefinito relativo alla costruzione e/o all'utilizzo degli elementi oggetto per una struttura non è consigliabile. Ciò può essere dovuto al fatto che la struttura ha lo scopo di inserire valori e, dal punto di vista concettuale, di funzionare come un'unione,

mentre i valori contenuti potrebbero essere interpretabili come reciprocamente esclusivi. Pertanto, non è possibile impostare alcuna delle proprietà della struttura. Viene [GridLength](#) un WPF esempio di tale struttura. In genere, tali strutture devono implementare un convertitore di tipi che consenta di esprimere i valori sotto forma di attributo tramite convenzioni di stringa che creano interpretazioni o modalità diverse dei valori della struttura. La struttura deve anche esporre un comportamento simile per la costruzione del codice tramite un costruttore senza parametri.

Requisiti per le proprietà di classi personalizzate come attributi XAML

Le proprietà devono fare riferimento a un tipo in base al valore (ad esempio una primitiva) o usare una classe per il tipo che ha un costruttore senza parametri o un convertitore di tipi dedicato a cui un processore XAML può accedere. Nell'implementazione XAML di CLR, i processori XAML trovano questi convertitori tramite il supporto nativo per le primitive di linguaggio o tramite l'applicazione di [TypeConverterAttribute](#) a un tipo o a un membro nelle definizioni di tipi di supporto

In alternativa, la proprietà può fare riferimento a un tipo di una classe astratta oppure a un'interfaccia. Per le classi astratte o le interfacce, l'aspettativa per l'analisi XAML è che il valore della proprietà debba essere popolato con istanze di classi pratiche che implementano l'interfaccia oppure con istanze di tipi che derivano dalla classe astratta.

In una classe astratta è possibile dichiarare le proprietà, ma queste possono essere impostate solo per le classi pratiche che derivano dalla classe astratta. Questo perché la creazione dell'elemento oggetto per la classe richiede un costruttore pubblico senza parametri sulla classe.

Sintassi degli attributi abilitata per TypeConverter

Se si fornisce un convertitore di tipi dedicato con attributi a livello di classe, la conversione di un tipo abilita la sintassi degli attributi per tutte le proprietà per le quali è necessario creare un'istanza del tipo in questione. Un convertitore di tipi non Abilita l'utilizzo dell'elemento oggetto del tipo. solo la presenza di un costruttore senza parametri per quel tipo Abilita l'utilizzo dell'elemento oggetto. In genere, pertanto, le proprietà abilitate dal convertitore di tipi possono essere usate nella sintassi delle proprietà solo se il tipo stesso supporta anche la sintassi degli elementi oggetto. Eccezione: è possibile specificare la sintassi di elementi proprietà, ma l'elemento proprietà deve contenere una stringa. Questo utilizzo è effettivamente essenzialmente equivalente all'utilizzo della sintassi degli attributi e tale utilizzo non è comune a meno che non sia necessaria una gestione più efficace dello spazio vuoto del valore dell'attributo. L'utilizzo degli elementi proprietà riportato di seguito, ad esempio, accetta una stringa e l'equivalente dell'utilizzo dell'attributo:

```
<Button>Hallo!
<Button.Language>
  de-DE
</Button.Language>
</Button>
```

```
<Button Language="de-DE">Hallo!</Button>
```

Esempi di proprietà in cui è consentita la sintassi degli attributi ma la sintassi dell'elemento proprietà che contiene un elemento oggetto non è consentita tramite XAML sono varie proprietà che accettano il tipo di [Cursor](#). La classe [Cursor](#) dispone di un convertitore di tipi dedicato [CursorConverter](#), ma non espone un costruttore senza parametri, quindi la proprietà [Cursor](#) può essere impostata solo tramite la sintassi dell'attributo, anche se il tipo di [Cursor](#) effettivo è un tipo riferimento.

Convertitori di tipi per proprietà

In alternativa, la proprietà stessa può dichiarare un convertitore di tipi a livello di proprietà. In questo modo viene abilitato un "linguaggio mini" che crea un'istanza di oggetti del tipo della proprietà inline, elaborando i valori

stringa in ingresso dell'attributo come input per un'operazione di [ConvertFrom](#) basata sul tipo appropriato. In genere questa operazione viene eseguita per fornire una funzione di accesso pratica e non come unico mezzo per consentire l'impostazione di una proprietà in XAML. Tuttavia, è anche possibile usare convertitori di tipi per gli attributi in cui si vogliono usare i tipi CLR esistenti che non forniscono un costruttore senza parametri o un convertitore di tipi con attributi. Gli esempi dell'API WPF sono alcune proprietà che accettano il tipo di [CultureInfo](#). In questo caso, WPF utilizza il tipo di [CultureInfo](#) Framework di Microsoft .NET esistente per migliorare gli scenari di compatibilità e migrazione utilizzati nelle versioni precedenti dei Framework, ma il tipo di [CultureInfo](#) non supportava i costruttori necessari o la conversione del tipo a livello di tipo in modo che sia possibile utilizzarli direttamente come valore di proprietà XAML.

Ogni volta che si espone una proprietà con utilizzo in XAML, in particolare se l'utente è un autore di controlli, è consigliabile supportare tale proprietà con una proprietà di dipendenza. Ciò è particolarmente vero se si usa l'implementazione Windows Presentation Foundation (WPF) esistente del processore XAML, perché è possibile migliorare le prestazioni usando [DependencyProperty](#) il backup. Una proprietà di dipendenza espone le funzionalità del sistema di proprietà relative alla proprietà in questione che gli utenti si aspettano da una proprietà accessibile tramite XAML, ad esempio funzionalità quali l'animazione, il data binding e il supporto degli stili. Per altre informazioni, vedere [Proprietà di dipendenza personalizzate](#) e [Caricamento XAML e proprietà di dipendenza](#).

Scrittura e assegnazione di un convertitore di tipi

Occasionalmente è necessario scrivere un [TypeConverter](#) classe derivata personalizzata per fornire la conversione del tipo per il tipo di proprietà. Per istruzioni su come derivare da e creare un convertitore di tipi in grado di supportare gli utilizzi XAML e come applicare la [TypeConverterAttribute](#), vedere [TypeConverter e XAML](#).

Requisiti per la sintassi degli attributi del gestore eventi XAML per gli eventi di una classe personalizzata

Per essere utilizzabile come evento CLR, l'evento deve essere esposto come evento pubblico in una classe che supporta un costruttore senza parametri o in una classe astratta in cui è possibile accedere all'evento sulle classi derivate. Per essere utilizzato in modo pratico come un evento indirizzato, l'evento CLR deve implementare metodi esplicativi `add` e `remove`, che aggiungono e rimuovono gestori per la firma dell'evento CLR e inoltrano tali gestori ai metodi [AddHandler](#) e [RemoveHandler](#). Questi metodi aggiungono o rimuovono i gestori dall'archivio dei gestori degli eventi indirizzati per l'istanza a cui l'evento è associato.

NOTE

È possibile registrare i gestori direttamente per gli eventi indirizzati usando [AddHandle](#)re non definire intenzionalmente un evento CLR che espone l'evento indirizzato. Questa operazione non è in genere consigliata. In questo caso, infatti, per l'evento non è abilitata la sintassi degli attributi XAML per il collegamento di gestori e nella classe risultante il codice XAML relativo alle caratteristiche del tipo è meno chiaro.

Scrittura delle proprietà delle raccolte

Le proprietà che accettano un tipo di raccolta hanno una sintassi XAML che consente di specificare gli oggetti da aggiungere alla raccolta. Questa sintassi presenta due funzionalità di rilievo.

- Nella sintassi dell'elemento oggetto non è necessario specificare l'oggetto che rappresenta l'oggetto Collection. Ogni volta che in XAML si specifica una proprietà che accetta un tipo di raccolta, tale tipo di raccolta è implicito.
- Gli elementi figlio della proprietà della raccolta nel markup vengono elaborati in modo che diventino membri della raccolta. In genere l'accesso del codice ai membri di una raccolta avviene tramite metodi di elenco o dizionario, ad esempio `Add`, oppure tramite un indicizzatore. La sintassi XAML, tuttavia, non supporta metodi o indicizzatori, ad eccezione di XAML 2009, che supporta i metodi. L'uso di XAML 2009,

però, limita i possibili utilizzi di WPF. Vedere [Funzionalità del linguaggio XAML 2009](#). Le raccolte sono ovviamente un requisito molto comune per la creazione di un albero di elementi. È quindi necessario individuare il modo più adatto per popolarle in XAML dichiarativo. Pertanto, gli elementi figlio di una proprietà della raccolta vengono elaborati aggiungendoli alla raccolta che rappresenta il valore del tipo della proprietà.

La definizione di proprietà di raccolta per l'implementazione dei servizi XAML di .NET Framework e quindi anche per il processore XAML di WPF è la seguente. Il tipo della proprietà deve soddisfare una delle condizioni seguenti:

- Implementa [IList](#).
- Implementa [IDictionary](#) o l'equivalente generico ([IDictionary<TKey,TValue>](#)).
- Deriva da [Array](#) (per altre informazioni sulle matrici in XAML, vedere estensione di [markup x:Array](#)).
- Implementa [IAddChild](#) (un'interfaccia definita da WPF).

In CLR ognuno di questi tipi dispone di un metodo [Add](#), usato dal processore XAML per aggiungere elementi alla raccolta sottostante durante la creazione dell'oggetto grafico.

NOTE

Le interfacce di [List](#) e [Dictionary](#) generiche ([IList<T>](#) e [IDictionary<TKey,TValue>](#)) non sono supportate per il rilevamento della raccolta da parte del processore XAML di WPF. Tuttavia, è possibile usare la classe [List<T>](#) come classe base, perché implementa direttamente [IList](#) o [Dictionary<TKey,TValue>](#) come classe di base, perché implementa [IDictionary](#) direttamente.

Quando si dichiara una proprietà che accetta una raccolta, prestare attenzione alla modalità di inizializzazione del valore di questa proprietà nelle nuove istanze del tipo. Se non si implementa la proprietà come proprietà di dipendenza, è adeguato che la proprietà usi un campo sottostante che chiami il costruttore del tipo della raccolta. Se la proprietà è una proprietà di dipendenza, potrebbe essere necessario inizializzare la proprietà della raccolta come parte del costruttore del tipo predefinito. Ciò è dovuto al fatto che il valore predefinito di una proprietà di dipendenza proviene dai metadati e in genere è necessario evitare che il valore iniziale di una proprietà di una raccolta corrisponda a una raccolta condivisa statica. Deve esistere un'istanza della raccolta per ogni istanza del tipo che la contiene. Per altre informazioni, vedere [Proprietà di dipendenza personalizzate](#).

Per la proprietà della raccolta è possibile implementare un tipo di raccolta personalizzato. A causa del trattamento implicito della proprietà della raccolta, non è necessario che il tipo di raccolta personalizzato fornisca un costruttore senza parametri per poter essere usato in modo隐式 in XAML. Tuttavia, è possibile specificare facoltativamente un costruttore senza parametri per il tipo di raccolta. Questo modo di procedere può risultare utile. A meno che non si fornisca un costruttore senza parametri, non è possibile dichiarare in modo esplicito la raccolta come elemento oggetto. Alcuni autori di markup interpretano la dichiarazione esplicita di una raccolta come una questione di stile di markup. Inoltre, un costruttore senza parametri può semplificare i requisiti di inizializzazione quando si creano nuovi oggetti che usano il tipo di raccolta come valore della proprietà.

Dichiarazione di proprietà di contenuto XAML

Nel linguaggio XAML è definito il concetto di proprietà di contenuto XAML. Ogni classe utilizzabile nella sintassi per gli oggetti può avere una sola proprietà di contenuto XAML. Per dichiarare una proprietà come proprietà di contenuto XAML per la classe, applicare la [ContentPropertyAttribute](#) come parte della definizione della classe. Specificare il nome della proprietà di contenuto XAML desiderata come [Name](#) nell'attributo. La proprietà viene specificata come stringa in base al nome, non come costrutto di reflection, ad esempio [PropertyInfo](#).

È possibile specificare una proprietà di raccolta come proprietà del contenuto XAML. Il risultato è un utilizzo di tale proprietà in cui l'elemento oggetto può avere uno o più elementi figlio, senza alcun tag di elementi oggetto Collection o di elementi proprietà. Questi elementi vengono considerati come valore per la proprietà del

contenuto XAML e vengono aggiunti all'istanza della raccolta sottostante.

Alcune proprietà del contenuto XAML esistenti usano il tipo di proprietà di `object`. Ciò consente a una proprietà di contenuto XAML che può accettare valori primitivi, ad esempio un [String](#), nonché di accettare un singolo valore dell'oggetto di riferimento. Se si segue questo modello, il tipo usato è responsabile della determinazione del tipo, nonché della gestione dei tipi possibili. Il motivo tipico di un tipo di contenuto [Object](#) consiste nel supportare sia un mezzo semplice per aggiungere il contenuto dell'oggetto come stringa (che riceve un trattamento di presentazione predefinito), sia un mezzo avanzato per aggiungere il contenuto dell'oggetto che specifica una presentazione non predefinita o dati aggiuntivi.

Serializzazione di XAML

Per alcuni scenari, ad esempio per gli autori di controlli, è anche necessario assicurarsi che qualsiasi rappresentazione di oggetti per cui può essere creata un'istanza in XAML possa anche essere serializzata di nuovo nel markup XAML equivalente. I requisiti di serializzazione non sono descritti in questo argomento. Vedere [Cenni preliminari sulla modifica di controlli](#) e [Struttura ad albero e serializzazione degli elementi](#).

Vedere anche

- [Panoramica di XAML \(WPF\)](#)
- [Proprietà di dipendenza personalizzate](#)
- [Cenni preliminari sulla modifica di controlli](#)
- [Cenni preliminari sugli elementi di base](#)
- [Caricamento XAML e proprietà di dipendenza](#)

Estensioni di markup e XAML WPF

03/02/2020 • 22 minutes to read • [Edit Online](#)

Questo argomento introduce le estensioni di markup per XAML, con informazioni sulle regole della sintassi, le finalità e il modello a oggetti di classe sottostante. Le estensioni di markup sono una funzionalità generale del linguaggio XAML e dell'implementazione .NET di servizi XAML. Questo argomento descrive in particolare le estensioni di markup per l'uso in XAML WPF.

Processori XAML ed estensioni di markup

In generale, un parser XAML può interpretare un valore di attributo come stringa letterale che può essere convertita in tipo primitivo oppure può convertirlo in oggetto in qualche modo. Uno di questi modi consiste nel fare riferimento a un convertitore di tipi e viene descritto nell'argomento [Convertitori di tipi e XAML](#). In alcuni scenari, tuttavia, è necessario un comportamento diverso. Ad esempio, è possibile indicare a un processore XAML che un valore di un attributo non deve essere restituito come nuovo oggetto nell'oggetto grafico. L'attributo deve invece essere restituito come oggetto grafico che crea un riferimento a un oggetto già costruito in un'altra parte dell'oggetto grafico o a un oggetto statico. In base a un altro scenario è possibile richiedere a un processore XAML di usare una sintassi che fornisca argomenti non predefiniti al costruttore di un oggetto. In scenari di questi tipi un'estensione di markup può essere la soluzione.

Sintassi delle estensioni di markup di base

Un'estensione di markup può essere implementata per fornire valori per le proprietà in una sintassi di attributi, le proprietà in una sintassi di elementi proprietà o entrambe.

Se usata per fornire un valore di attributo, la sintassi che distingue una sequenza di estensioni di markup per un processore XAML è la presenza delle parentesi graffe di apertura e chiusura ({ e }). Il tipo di estensione di markup viene quindi identificato dal token di stringa immediatamente successivo alla parentesi graffa di apertura.

Se usata nella sintassi di elementi proprietà, un'estensione di markup è visivamente identica a qualsiasi altro elemento usato per fornire il valore di un elemento proprietà: un elemento dichiarato XAML che fa riferimento alla classe dell'estensione di markup come elemento, racchiuso tra parentesi angolari (<>).

Estensioni di markup definite da XAML

Esistono diverse estensioni di markup non specifiche dell'implementazione WPF di XAML, ma che sono invece implementazioni di intrinseci o funzionalità di XAML come linguaggio. Queste estensioni di markup vengono implementate nell'assembly System.Xaml come parte dei servizi XAML generali di .NET Framework e sono incluse nello spazio dei nomi XAML del linguaggio XAML. In termini di utilizzo comune del markup, queste estensioni di markup sono in genere identificabili dal prefisso `x:` nella sintassi. La classe di base [MarkupExtension](#) (definita anche in System.Xaml) fornisce il modello che deve essere usato da tutte le estensioni di markup per supportare i reader XAML e i writer XAML, incluso in XAML WPF.

- `x:Type` fornisce l'oggetto [Type](#) per il tipo denominato. Questa funzionalità viene usata più comunemente in stili e modelli. Per informazioni dettagliate, vedere [Estensione di markup x:Type](#).
- `x:Static` produce valori statici. I valori provengono da entità di codice di tipo di valore che non sono direttamente il tipo del valore di una proprietà di destinazione, ma possono restituire tale tipo. Per informazioni dettagliate, vedere [Estensione di markup x:Static](#).

- `x:Null` specifica `null` come valore per una proprietà e può essere usata per attributi o valori di elementi proprietà. Per informazioni dettagliate, vedere [Estensione di markup x:Null](#).
- `x:Array` supporta la creazione di matrici generali nella sintassi XAML nei casi in cui si sceglie intenzionalmente di non usare il supporto delle raccolte fornito dagli elementi di base e dai modelli di controllo di WPF. Per informazioni dettagliate, vedere [Estensione di markup x:Array](#).

NOTE

Il prefisso `x:` viene usato per il mapping dello spazio dei nomi XAML tipico degli intrinseci del linguaggio XAML, nell'elemento radice di un file o una produzione XAML. Ad esempio, i modelli di Visual Studio per le applicazioni WPF avviano un file XAML utilizzando questo mapping `x:`. Nel mapping dello spazio dei nomi XAML personalizzato è possibile scegliere un token di prefisso diverso. Tuttavia, per identificare le entità che rappresentano una parte definita dello spazio dei nomi XAML per il linguaggio XAML, questa documentazione usa il mapping `x:` predefinito anziché lo spazio dei nomi WPF predefinito o altri spazi dei nomi XAML non correlati a un framework specifico.

Estensioni di markup specifiche di WPF

Le estensioni di markup più comuni usate nella programmazione WPF sono quelle che supportano i riferimenti a risorse (`StaticResource` e `DynamicResource`) e quelli che supportano il data binding (`Binding`).

- `StaticResource` fornisce un valore per una proprietà sostituendo il valore di una risorsa già definita. Una valutazione di `StaticResource` viene infine eseguita in fase di caricamento XAML e non ha accesso all'oggetto grafico in fase di esecuzione. Per informazioni dettagliate, vedere [Estensione di markup StaticResource](#).
- `DynamicResource` fornisce un valore per una proprietà rinviando il valore come riferimento in fase di esecuzione a una risorsa. Il riferimento a una risorsa dinamica forza una nuova ricerca ogni volta che si accede a tale risorsa e che la risorsa ha accesso all'oggetto grafico in fase di esecuzione. Per ottenere questo accesso, il concetto di `DynamicResource` è supportato da proprietà di dipendenza nel sistema di proprietà WPF e da espressioni valutate. Di conseguenza, è possibile usare `DynamicResource` solo per la destinazione di una proprietà di dipendenza. Per informazioni dettagliate, vedere [Estensione di markup DynamicResource](#).
- `Binding` fornisce un valore di data binding per una proprietà, usando il contesto dei dati valido per l'oggetto padre in fase di esecuzione. Questa estensione di markup è relativamente complessa, perché permette una sintassi inline sostanziale per specificare un data binding. Per informazioni dettagliate, vedere [Estensione di markup Binding](#).
- `RelativeSource` fornisce informazioni sull'origine per un `Binding` in grado di spostarsi tra più possibili relazioni nell'albero degli oggetti di run-time. In questo modo, si ottengono origini specializzate per i binding creati in modelli multiuso o creati nel codice senza una completa conoscenza dell'albero di oggetti circostante. Per informazioni dettagliate, vedere [Estensione di markup RelativeSource](#).
- `TemplateBinding` permette a un modello di controllo di usare valori per proprietà basate su modelli provenienti da proprietà definite da modelli a oggetti della classe che userà il modello. In altri termini, la proprietà all'interno della definizione del modello può accedere a un contesto che esiste solo dopo l'applicazione del modello. Per informazioni dettagliate, vedere [Estensione di markup TemplateBinding](#). Per altre informazioni sull'uso pratico di `TemplateBinding`, vedere [Applicazione di stili con l'esempio di ControlTemplates](#).
- `ColorConvertedBitmap` supporta uno scenario di creazione di immagini relativamente avanzato. Per informazioni dettagliate, vedere [Estensione di markup ColorConvertedBitmap](#).
- `ComponentResourceKey` e `ThemeDictionary` supportano alcuni aspetti della ricerca di risorse, in

particolare per risorse e temi che contengono controlli personalizzati. Per altre informazioni, vedere [Estensione di markup ComponentResourceKey](#), [Estensione di markup ThemeDictionary](#) o [Panoramica della creazione di controlli](#).

Classi di estensione *

Per il linguaggio XAML generale e per le estensioni di markup specifiche di WPF, il comportamento di ogni estensione di markup viene identificato a un processore XAML tramite una classe `*Extension` che deriva da `MarkupExtension` fornisce un'implementazione del metodo `ProvideValue`. Questo metodo in ogni estensione fornisce l'oggetto restituito quando viene valutata l'estensione di markup. L'oggetto restituito viene in genere valutato in base ai diversi token di stringa passati all'estensione di markup.

Ad esempio, la classe `StaticResourceExtension` fornisce l'implementazione della superficie di ricerca effettiva della risorsa in modo che la relativa implementazione di `ProvideValue` restituisca l'oggetto richiesto, con l'input di quell'implementazione particolare è una stringa utilizzata per cercare la risorsa in base alla relativa `x:Key`. Molti dei dettagli di questa implementazione sono irrilevanti se si usa un'estensione di markup esistente.

Alcune estensioni di markup non usano argomenti dei token di stringa. Il motivo è che restituiscono un valore statico o coerente oppure perché il contesto per il valore da restituire è disponibile tramite uno dei servizi passati tramite il parametro `serviceProvider`.

Il criterio di denominazione `*Extension` ha scopi di praticità e coerenza. Non è necessario perché un processore XAML identifichi la classe come supporto per un'estensione di markup. Finché la codebase include System. XAML e USA .NET Framework implementazioni dei servizi XAML, tutto ciò che è necessario riconoscere come estensione di markup XAML consiste nel derivare da `MarkupExtension` e supportare una sintassi di costruzione. WPF definisce le classi abilitate per l'estensione di markup che non seguono il modello di denominazione `*Extension`, ad esempio `Binding`. In genere il motivo è che la classe supporta scenari che vanno oltre il puro supporto delle estensioni di markup. Nel caso di `Binding`, tale classe supporta l'accesso in fase di esecuzione ai metodi e alle proprietà dell'oggetto per gli scenari che non hanno nulla a che fare con XAML.

Interpretazione del testo di inizializzazione da parte delle classi delle estensioni

I token di stringa che seguono il nome dell'estensione di markup e ancora racchiuse tra parentesi graffe vengono interpretati da un processore XAML in uno dei modi seguenti:

- Una virgola rappresenta sempre il separatore o il delimitatore di singoli token.
- Se i singoli token separati non contengono segni di uguale, ogni token viene considerato un argomento del costruttore. Ogni parametro del costruttore deve essere specificato come tipo previsto dalla firma e nell'ordine corretto previsto dalla firma.

NOTE

Un processore XAML deve chiamare il costruttore corrispondente al conteggio di argomenti del numero di copie. Per questo motivo, se si implementa un'estensione di markup personalizzata, non fornire più costruttori con lo stesso numero di argomenti. Il comportamento di un processore XAML se sono presenti più percorsi del costruttore dell'espressione di markup con lo stesso parametro è indefinito, ma è bene aspettarsi che a un processore XAML è consentito generare un'eccezione sull'utilizzo in presenza di questa situazione nelle definizioni dei tipi di estensione di markup.

- Se i singoli token separati contengono segni di uguale, un processore XAML chiama prima di tutto il costruttore senza parametri per l'estensione di markup. Quindi, ogni coppia nome=valore viene interpretata come nome di una proprietà presente nell'estensione di markup e come valore da assegnare alla proprietà.

- In presenza di un risultato parallelo tra il comportamento del costruttore e il comportamento di impostazione delle proprietà in un'estensione di markup, il comportamento usato non è importante. È prassi più comune usare le coppie *proprietà* = *valore* per le estensioni di markup che includono più di una proprietà impostabile, anche solo perché questo comportamento rende il markup più intenzionale e riduce la probabilità di trasporre accidentalmente i parametri del costruttore. Quando si specificano le coppie proprietà = valore, queste proprietà possono essere in qualsiasi ordine. Inoltre, non vi è alcuna garanzia che un'estensione di markup fornisca un parametro del costruttore che imposta tutte le proprietà impostabili. Ad esempio, [Binding](#) è un'estensione di markup, con molte proprietà che possono essere impostate tramite l'estensione nel formato *= valore della proprietà*, ma [Binding](#) supporta solo due costruttori, ovvero un costruttore senza parametri e uno che imposta un percorso iniziale.
- Una virgola letterale non può essere passata a un'estensione di markup senza sequenza di escape.

Sequenze di escape ed estensioni di markup

La gestione degli attributi in un processore XAML usa le parentesi graffe come indicatori di una sequenza di estensioni di markup. È anche possibile produrre un valore di attributo di un carattere di parentesi graffa letterale se necessario, ma immettendo una sequenza di escape tramite una coppia di parentesi graffe vuote seguita dalla parentesi graffa letterale. Vedere [{} sequenza di escape-estensione di markup](#).

Annidamento di estensioni di markup nella sintassi XAML

L'annidamento di più estensioni di markup è supportato e viene valutata per prima l'estensione di markup a livello più profondo. Ad esempio, si consideri la sintassi seguente:

```
<Setter Property="Background"
    Value="{DynamicResource {x:Static SystemColors.ControlBrushKey}}" />
```

In questa sintassi l'istruzione `x:Static` viene valutata per prima e restituisce una stringa. Questa stringa viene quindi usata come argomento per `DynamicResource`.

Estensioni di markup e sintassi di elementi proprietà

Se usata come elemento oggetto che immette un valore di elemento proprietà, la classe di un'estensione di markup è visivamente indistinguibile da un tipico elemento oggetto supportato da tipi che può essere usato in XAML. La differenza pratica tra un elemento oggetto tipico e un'estensione di markup è che l'estensione di markup restituisce un valore tipizzato o viene rinviata come espressione. Di conseguenza, i meccanismi per tutti i possibili errori di tipo dei valori di proprietà per l'estensione di markup differiranno, analogamente a come viene gestita una proprietà di data binding in altri modelli di programmazione. Un elemento oggetto comune viene valutato per la corrispondenza dei tipi rispetto alla proprietà di destinazione impostata durante l'analisi del codice XAML.

La maggior parte delle estensioni di markup, se usate nella sintassi di elementi oggetto per immettere un elemento proprietà, non avrà contenuto né ulteriore sintassi di elementi proprietà al suo interno. In questo modo, è necessario chiudere il tag dell'elemento oggetto, senza fornire elementi figlio. Ogni volta che un elemento oggetto viene rilevato da un processore XAML, viene chiamato il costruttore per la classe, che crea un'istanza dell'oggetto creato dall'elemento analizzato. Una classe dell'estensione di markup non è diversa: se si desidera che l'estensione di markup sia utilizzabile nella sintassi dell'elemento oggetto, è necessario fornire un costruttore senza parametri. Alcune estensioni di markup esistenti hanno almeno un valore di proprietà obbligatorio che deve essere specificato ai fini dell'inizializzazione corretta. In questo caso, il valore di proprietà viene in genere specificato come attributo della proprietà nell'elemento oggetto. Nelle pagine di riferimento [Funzionalità del linguaggio dello spazio dei nomi XAML \(x:\)](#) e [Estensioni XAML WPF](#) verranno indicate le estensioni di markup che includono proprietà obbligatorie, insieme ai nomi delle proprietà

obbligatorie. Le pagine di riferimento indicano anche se la sintassi degli elementi oggetto o degli attributi non è consentita per una determinata estensione di markup. Un caso interessante riguarda l'estensione di markup [x:Array](#), che non può supportare la sintassi di attributi perché il contenuto della matrice deve essere specificata all'interno dei tag come contenuto. Poiché i contenuti della matrice vengono gestiti come oggetti generali, non è possibile usare alcun convertitore di tipi predefiniti per l'attributo. Inoltre, l'estensione di markup [x:Array](#) richiede un parametro `type`.

Vedere anche

- [Panoramica di XAML \(WPF\)](#)
- [Funzionalità del linguaggio dello spazio dei nomi XAML \(x:\)](#)
- [Estensioni XAML WPF](#)
- [Estensione di markup StaticResource](#)
- [Estensione di markup Binding](#)
- [Estensione del markup DynamicResource](#)
- [Estensione di markup x:Type](#)

Spazi dei nomi XAML e mapping dello spazio dei nomi per XAML WPF

12/02/2020 • 15 minutes to read • [Edit Online](#)

Questo argomento approfondisce la presenza e lo scopo dei due mapping dello spazio dei nomi XAML nel tag radice di ogni file XAML WPF. L'argomento descrive anche come produrre mapping simili per l'uso di elementi definiti nel codice e/o all'interno di assembly separati.

Definizione di spazio dei nomi XAML

Uno spazio dei nomi XAML è in realtà un'estensione del concetto di spazio dei nomi XML. Le tecniche di specifica di uno spazio dei nomi XAML si basano sulla sintassi dello spazio dei nomi XML, sulla convenzione dell'uso di URI come identificatori dello spazio dei nomi, sull'uso di prefissi per fare riferimento a più spazi dei nomi dalla stessa origine del markup e così via. Il concetto principale aggiunto alla definizione XAML rispetto allo spazio dei nomi XML consiste nel fatto che uno spazio dei nomi XAML implica un ambito di univocità per gli utilizzi del markup e influenza il modo in cui le entità del markup possono essere supportate da spazi dei nomi e da assembly di riferimento CLR specifici. Questa seconda considerazione è influenzata anche dal concetto di contesto dello schema XAML. Tuttavia, per quanto riguarda il funzionamento di WPF rispetto agli spazi dei nomi XAML, è possibile considerare gli spazi dei nomi XAML in termini di mapping diretto da parte del markup XAML di uno spazio dei nomi XAML predefinito, dello spazio dei nomi del linguaggio XAML e di qualsiasi altro spazio dei nomi XAML agli spazi dei nomi e agli assembly di riferimento CLR di supporto specifici.

Dichiarazioni dello spazio dei nomi XAML e WPF

All'interno delle dichiarazioni dello spazio dei nomi nel tag radice di molti file XAML sono in genere presenti due dichiarazioni di spazi dei nomi XML. La prima dichiarazione esegue il mapping dello spazio dei nomi del client WPF complessivo o del framework XAML come predefinito:

```
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

La seconda dichiarazione esegue il mapping di uno spazio dei nomi XAML separato, in genere al prefisso `x:`.

```
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

La relazione tra queste dichiarazioni consiste nel fatto che il mapping del prefisso `x:` supporta gli intrinseci che fanno parte della definizione del linguaggio XAML e WPF è un'implementazione che usa XAML come linguaggio e ne definisce un vocabolario degli oggetti per XAML. Poiché l'utilizzo del vocabolario WPF è in genere molto più frequente dell'utilizzo degli intrinseci XAML, il vocabolario WPF viene mappato come predefinito.

La convenzione di prefisso `x:` per il mapping del supporto intrinseco del linguaggio XAML è seguita da modelli di progetto, codice di esempio e dalla documentazione delle funzionalità del linguaggio all'interno di questo SDK. Lo spazio dei nomi XAML definisce molte funzionalità di uso comune, necessarie anche per le applicazioni WPF di base. Ad esempio, per unire code-behind a un file XAML tramite una classe parziale, è necessario denominare la classe come attributo `x:Class` nell'elemento radice del file XAML corrispondente. In alternativa, per qualsiasi elemento definito in una pagina XAML a cui si desidera accedere come risorsa con chiave è necessario che l'attributo `x:Key` sia impostato sull'elemento in questione. Per altre informazioni su questi e altri aspetti di XAML, vedere [Cenni preliminari su XAML \(WPF\)](#) o [Descrizione dettagliata della sintassi XAML](#).

Mapping a classi e assembly personalizzati

È possibile eseguire il mapping di spazi dei nomi XML ad assembly tramite una serie di token all'interno di una dichiarazione di prefisso `xmlns` eseguendo una procedura analoga a quella del mapping degli spazi dei nomi WPF e XAML degli intrinseci XAML standard ai prefissi.

La sintassi accetta i token denominati e i valori riportati di seguito:

`clr-namespace:` lo spazio dei nomi CLR dichiarato nell'assembly che contiene i tipi pubblici da esporre come elementi.

`assembly=` l'assembly che contiene alcuni o tutti gli spazi dei nomi CLR a cui si fa riferimento. Il valore è dato in genere dal nome dell'assembly, non dal percorso, e non include l'estensione (ad esempio, dll o exe). Il percorso dell'assembly deve essere stabilito come riferimento al progetto nel file di progetto che contiene il codice XAML di cui si sta tentando di eseguire il mapping. Per incorporare il controllo delle versioni e la firma con nome sicuro, il valore `assembly` può essere una stringa definita da [AssemblyName](#), anziché il nome di stringa semplice.

Si noti che il carattere che separa il token `clr-namespace` dal valore è il segno di due punti (:), mentre il carattere che separa il token `assembly` dal valore è un segno di uguale (=). Il carattere da usare tra i due token è il segno di punto e virgola. Inoltre, non includere alcuno spazio vuoto in un punto qualsiasi della dichiarazione.

Esempio di base di mapping personalizzato

Il codice seguente definisce una classe personalizzata di esempio:

```
namespace SDKSample {  
    public class ExampleClass : ContentControl {  
        public ExampleClass() {  
            ...  
        }  
    }  
}
```

```
Namespace SDKSample  
    Public Class ExampleClass  
        Inherits ContentControl  
        ...  
        Public Sub New()  
        End Sub  
    End Class  
End Namespace
```

Questa classe personalizzata viene quindi compilata in una libreria che, in base alle impostazioni del progetto (non visualizzate), è denominata `SDKSampleLibrary`.

Per fare riferimento a tale classe personalizzata, è anche necessario includerla come riferimento per il progetto corrente, operazione in genere effettuata tramite l'interfaccia utente di Esplora soluzioni in Visual Studio.

Ora che si dispone di una libreria contenente una classe e di un riferimento a questa nelle impostazioni del progetto, è possibile aggiungere il mapping del prefisso seguente come parte dell'elemento radice in XAML:

```
xmlns:custom="clr-namespace:SDKSample;assembly=SDKSampleLibrary"
```

Riepilogando, il codice XAML seguente include il mapping personalizzato insieme al mapping predefinito e al mapping del prefisso x: consueti nel tag radice, quindi usa un riferimento con prefisso per creare un'istanza di `ExampleClass` in tale interfaccia utente:

```

<Page x:Class="WPFApplication1.MainPage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:custom="clr-namespace:SDKSample;assembly=SDKSampleLibrary">
    ...
    <custom:ExampleClass/>
    ...
</Page>

```

Mapping ad assembly correnti

È possibile omettere `assembly` se `clr-namespace` a cui si fa riferimento è definito all'interno dello stesso assembly del codice dell'applicazione che fa riferimento alle classi personalizzate. In alternativa, una sintassi equivalente appropriata consiste nello specificare `assembly=` senza token di stringa dopo il segno di uguale.

Non è possibile usare le classi personalizzate come elemento radice di una pagina se definite nello stesso assembly. Non è necessario eseguire il mapping delle classi parziali, ma solo di quelle che non costituiscono la classe parziale di una pagina dell'applicazione, se si intende fare riferimento a esse come elementi in XAML.

Mapping di spazi dei nomi CLR a spazi dei nomi XML in un assembly

WPF definisce un attributo CLR che viene usato dai processori XAML per eseguire il mapping di più spazi dei nomi CLR a un unico spazio dei nomi XAML. Questo attributo, [XmlAttribute](#), viene inserito a livello di assembly nel codice sorgente che produce l'assembly. Il codice sorgente dell'assembly WPF usa questo attributo per eseguire il mapping dei vari spazi dei nomi comuni, ad esempio [System.Windows](#) e [System.Windows.Controls](#), allo spazio dei nomi <http://schemas.microsoft.com/winfx/2006/xaml/presentation>.

Il [XmlAttribute](#) accetta due parametri: il nome dello spazio dei nomi XML/XAML e il nome dello spazio dei nomi CLR. È possibile che esistano più [XmlAttribute](#) per eseguire il mapping di più spazi dei nomi CLR allo stesso spazio dei nomi XML. Dopo avere eseguito il mapping, è possibile fare riferimento ai membri degli spazi dei nomi senza nome completo specificando l'istruzione `using` appropriata nella pagina code-behind della classe parziale. Per altri dettagli, vedere [XmlAttribute](#).

Spazi dei nomi della finestra di progettazione e altri prefissi di modelli XAML

Se si lavora con ambienti di sviluppo e/o con strumenti di progettazione per XAML WPF, si può notare che all'interno del markup XAML esistono altri spazi dei nomi XAML definiti o altri prefissi.

WPF Designer per Visual Studio usa uno spazio dei nomi della finestra di progettazione che in genere viene mappato al prefisso `d:`. Modelli di progetto più recenti per WPF potrebbero pre-eseguire il mapping di questo spazio dei nomi XAML per supportare l'interscambio del codice XAML tra WPF Designer per Visual Studio e altri ambienti di progettazione. Questo spazio dei nomi XAML di progettazione è usato per trasmettere lo stato di progettazione durante la sequenza di andata e ritorno dell'interfaccia utente basata su XAML nella finestra di progettazione. Questo spazio dei nomi viene usato anche per funzionalità quali `d:IsDataSource`, che abilitano origini dati di runtime in una finestra di progettazione.

Un altro prefisso che è possibile vedere mappato è `mc:`. `mc:` è finalizzato alla compatibilità dei markup e usa un modello di compatibilità dei markup non necessariamente specifico di XAML. In alcuni casi, è possibile usare le funzionalità di compatibilità dei markup per scambiare XAML tra framework o tra altri limiti di implementazione di supporto, per lavorare tra contesti di schemi XAML, per garantire la compatibilità per modalità limitate nelle finestre di progettazione e così via. Per altre informazioni sui concetti di compatibilità dei markup e sulla relazione con WPF, vedere [Funzionalità del linguaggio per la compatibilità dei markup \(mc:\)](#).

WPF e caricamento di assembly

Il contesto dello schema XAML per WPF si integra con il modello di applicazione WPF, che a sua volta usa il concetto di [AppDomain](#) definito da CLR. Nella sequenza seguente viene descritto il modo in cui il contesto dello schema XAML interpreta come caricare assembly o trovare tipi in fase di esecuzione o di progettazione, in base all'utilizzo di [AppDomain](#) e ad altri fattori.

1. Scorre il [AppDomain](#) cercando un assembly già caricato che corrisponda a tutti gli aspetti del nome, a partire dall'assembly caricato più di recente.
2. Se il nome è qualificato, chiamare [Assembly.Load\(String\)](#) per il nome completo.
3. Se la combinazione nome breve e token di chiave pubblica di un nome qualificato corrisponde all'assembly da cui è stato caricato il markup, restituire tale assembly.
4. Usare il nome breve e il token di chiave pubblica per chiamare [Assembly.Load\(String\)](#).
5. Se il nome non è qualificato, chiamare [Assembly.LoadWithPartialName](#).

Per il codice XAML loose il passaggio 3 non viene eseguito. Non è infatti presente alcun assembly da cui effettuare il caricamento.

Il codice XAML compilato per WPF (generato tramite XamlBuildTask) non usa gli assembly già caricati da [AppDomain](#) (passaggio 1). Dall'output di XamlBuildTask, poi, il nome non può mai essere non qualificato, pertanto il passaggio 5 non viene applicato.

Il codice BAML (generato tramite PresentationBuildTask) usa tutti i passaggi, anche se non deve contenere nomi di assembly non qualificati.

Vedere anche

- [Informazioni sugli spazi dei nomi XML](#)
- [Panoramica di XAML \(WPF\)](#)

Ambiti dei nomi XAML WPF

31/01/2020 • 16 minutes to read • [Edit Online](#)

Un ambito dei nomi XAML è un concetto che identifica gli oggetti definiti in XAML. I nomi in un ambito dei nomi XAML possono essere usati per stabilire relazioni tra i nomi definiti da XAML degli oggetti e i rispettivi equivalenti di istanza in un albero di oggetti. In genere gli ambiti dei nomi nel codice gestito WPF vengono creati durante il caricamento degli elementi radice delle singole pagine XAML per un'applicazione XAML. I NameScope XAML come oggetto di programmazione sono definiti dall'interfaccia [INamespace](#) e vengono implementati anche dalla classe pratica [Namespace](#).

Ambiti dei nomi in applicazioni XAML caricate

In un più ampio contesto di programmazione o informatico i concetti di programmazione includono spesso il principio di identificatore o nome univoco, che può essere usato per accedere a un oggetto. Per i sistemi che usano identificatori o nomi, l'ambito dei nomi definisce i limiti entro i quali un processo o una tecnica individuerà se è necessario un oggetto con tale nome oppure i limiti entro i quali viene applicata l'univocità dei nomi di identificazione. Questi principi generali si applicano agli ambiti dei nomi XAML. In WPF gli ambiti dei nomi XAML vengono creati nell'elemento radice per una pagina XAML quando questa viene caricata. Ogni nome specificato nella pagina XAML che inizia in corrispondenza della radice della pagina viene aggiunto a un ambito dei nomi XAML pertinente.

In XAML WPF gli elementi che sono elementi radice comuni (ad esempio [Page](#) o [Window](#)) controllano sempre un ambito dei nomi XAML. Se un elemento, ad esempio [FrameworkElement](#) o [FrameworkContentElement](#) è l'elemento radice della pagina nel markup, un processore di XAML aggiunge una radice [Page](#) in modo implicito affinché il [Page](#) possa fornire un NameScope XAML funzionante.

NOTE

Le azioni di compilazione di WPF creano un ambito dei nomi XAML per una produzione XAML anche se non è definito alcun attributo `Name` o `x:Name` in alcun elemento nel markup XAML.

Se si prova a usare lo stesso nome due volte in un ambito dei nomi XAML, viene generata un'eccezione. Per codice XAML WPF che include code-behind e che fa parte di un'applicazione compilata, l'eccezione viene generata in fase di compilazione dalle azioni di compilazione di WPF, quando viene creata la classe generata per la pagina durante la compilazione iniziale del markup. Per codice XAML non compilato dal markup tramite alcuna azione di compilazione, potrebbero essere generate eccezioni correlate a problemi dell'ambito dei nomi XAML durante il caricamento di XAML. I progettisti XAML possono anche prevedere i problemi relativi all'ambito dei nomi XAML in fase di progettazione.

Aggiunta di oggetti all'albero di oggetti di runtime

Il momento in cui XAML viene analizzato rappresenta il momento in cui viene creato e definito un ambito dei nomi XAML WPF. Se si aggiunge un oggetto a un albero di oggetti in un momento successivo all'analisi del codice XAML che ha prodotto l'albero, un valore `Name` o `x:Name` nel nuovo oggetto non aggiorna automaticamente le informazioni in un ambito dei nomi XAML. Per aggiungere un nome per un oggetto in un NameScope XAML WPF dopo il caricamento di XAML, è necessario chiamare l'implementazione appropriata di [RegisterName](#) nell'oggetto che definisce l'ambito dei nomi XAML, che in genere è la radice della pagina XAML. Se il nome non è registrato, il nome non può fare riferimento all'oggetto aggiunto tramite metodi come [FindName](#) non è possibile usare tale nome per la destinazione dell'animazione.

Lo scenario più comune per gli sviluppatori di applicazioni è che si utilizzerà [RegisterName](#) per registrare i nomi

nell'ambito dei nomi XAML nella radice corrente della pagina. [RegisterName](#) fa parte di uno scenario importante per gli storyboard destinati a oggetti per le animazioni. Per altre informazioni, vedere [Cenni preliminari sugli storyboard](#).

Se si chiama [RegisterName](#) su un oggetto diverso dall'oggetto che definisce l'ambito dei nomi XAML, il nome viene ancora registrato nell'ambito dei nomi XAML in cui è contenuto l'oggetto chiamante, come se si fosse chiamato [RegisterName](#) sull'oggetto che definisce l'ambito dei nomi XAML.

Ambiti dei nomi XAML nel codice

È possibile creare e quindi usare ambiti dei nomi XAML nel codice. Le API e i concetti interessati dalla creazione di ambiti dei nomi XAML sono gli stessi, anche per l'utilizzo di codice puro, perché il processore XAML per WPF usa questi concetti e API quando elabora il codice XAML stesso. I concetti e le API hanno prevalentemente lo scopo di trovare oggetti in base al nome all'interno di un albero di oggetti definito parzialmente o interamente in XAML.

Per le applicazioni create a livello di codice e non da codice XAML caricato, l'oggetto che definisce un ambito dei nomi XAML deve implementare [INamespaceScope](#), o essere una classe derivata [FrameworkElement](#) o [FrameworkContentElement](#), per supportare la creazione di un ambito dei nomi XAML nelle relative istanze.

Inoltre, per qualsiasi elemento che non viene caricato ed elaborato da un processore XAML, l'ambito dei nomi XAML per l'oggetto non viene creato o inizializzato per impostazione predefinita. È necessario creare un nuovo ambito dei nomi XAML per qualsiasi oggetto in cui si intende successivamente registrare nomi. Per creare un ambito dei nomi XAML, chiamare il metodo statico [SetNameScope](#). Specificare l'oggetto che ne sarà proprietario come parametro di `dependencyObject` e una nuova chiamata al costruttore [NameScope](#) come parametro `value`.

Se l'oggetto fornito come `dependencyObject` per [SetNameScope](#) non è un'implementazione [INamespaceScope](#), [FrameworkElement](#) o [FrameworkContentElement](#), la chiamata di [RegisterName](#) su tutti gli elementi figlio non avrà alcun effetto. Se non è possibile creare in modo esplicito il nuovo ambito dei nomi XAML, le chiamate a [RegisterName](#) genereranno un'eccezione.

Per un esempio dell'uso di API di ambito dei nomi XAML nel codice, vedere [Definire un ambito dei nomi](#).

Ambiti dei nomi XAML in stili e modelli

Gli stili e i modelli in WPF permettono di riutilizzare e riapplicare contenuto in modo diretto. Tuttavia, stili e modelli potrebbero includere anche elementi con nomi XAML definiti a livello del modello. Questo stesso modello potrebbe essere usato più volte in una pagina. Per questo motivo, gli stili e i modelli definiscono entrambi ambiti dei nomi XAML propri, indipendentemente dalla posizione in un albero di oggetti in cui viene applicato lo stile o il modello.

Si consideri l'esempio seguente:

```
<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    >
    <Page.Resources>
        <ControlTemplate x:Key="MyButtonTemplate" TargetType="{x:Type Button}">
            <Border BorderBrush="Red" Name="TheBorder" BorderThickness="2">
                <ContentPresenter/>
            </Border>
        </ControlTemplate>
    </Page.Resources>
    <StackPanel>
        <Button Template="{StaticResource MyButtonTemplate}">My first button</Button>
        <Button Template="{StaticResource MyButtonTemplate}">My second button</Button>
    </StackPanel>
</Page>
```

Qui viene applicato lo stesso modello a due diversi pulsanti. Se i modelli non avessero ambiti dei nomi XAML discreti, il nome `TheBorder` usato nel modello causerebbe un conflitto di nomi nell'ambito dei nomi XAML. Poiché la creazione di ogni istanza del modello ha un ambito dei nomi XAML proprio, in questo esempio l'ambito dei nomi di ogni modello di cui è stata creata un'istanza contiene esattamente un solo nome.

Anche gli stili definiscono un ambito dei nomi XAML proprio, per lo più in modo che alle parti degli storyboard possano essere assegnati nomi specifici. Questi nomi permettono comportamenti specifici dei controlli destinati a elementi con questi nomi, anche se il modello è stato ridefinito come parte della personalizzazione del controllo.

A causa della presenza di ambiti dei nomi XAML separati, la ricerca di elementi denominati in un modello è più complessa rispetto all'individuazione di un elemento denominato senza modello in una pagina. Per prima cosa è necessario determinare il modello applicato, ottenendo il valore della proprietà `Template` del controllo in cui viene applicato il modello. Viene quindi chiamata la versione del modello di `FindName`, passando il controllo in cui è stato applicato il modello come secondo parametro.

Se si è un autore del controllo e si genera una convenzione in cui un particolare elemento denominato in un modello applicato è la destinazione per un comportamento definito dal controllo stesso, è possibile usare il metodo `GetTemplateChild` dal codice di implementazione del controllo. Il metodo `GetTemplateChild` è protetto, quindi solo l'autore del controllo può accedervi.

Se si utilizza un modello ed è necessario ottenere l'ambito dei nomi XAML in cui viene applicato il modello, ottenere il valore di `TemplatedParent`, quindi chiamare `FindName`. Un esempio di questa situazione è quando si scrive l'implementazione del gestore eventi in cui l'evento verrà generato da un elemento in un modello applicato.

Ambiti dei nomi XAML e API correlate ai nomi

`FrameworkElement` dispone di metodi `FindName`, `RegisterName` e `UnregisterName`. Se l'oggetto in cui vengono chiamati questi metodi è proprietario di un ambito dei nomi XAML, i metodi chiamano nei metodi dell'ambito dei nomi XAML pertinente. In caso contrario, l'elemento padre viene controllato per verificare se è proprietario di un ambito dei nomi XAML e questo processo continua in modo ricorsivo fino a quando non viene trovato un ambito dei nomi XAML. A causa del comportamento del processore XAML, esiste sicuramente un ambito dei nomi XAML nella radice. `FrameworkContentElement` ha comportamenti analoghi, ad eccezione del fatto che nessun `FrameworkContentElement` avrà mai un ambito dei nomi XAML. I metodi sono disponibili in `FrameworkContentElement`, in modo che le chiamate possano essere successivamente trasmesse a un elemento padre `FrameworkElement`.

`SetNameScope` viene utilizzato per eseguire il mapping di un nuovo ambito dei nomi XAML a un oggetto esistente. È possibile chiamare `SetNameScope` più di una volta per reimpostare o deselezionare l'ambito dei nomi XAML, ma questo non è un utilizzo comune. Inoltre, `GetNameScope` non viene in genere utilizzato dal codice.

Implementazioni di ambiti dei nomi XAML

Le classi seguenti implementano `INamespaceScope` direttamente:

- `NameScope`
- `Style`
- `ResourceDictionary`
- `FrameworkTemplate`

`ResourceDictionary` non utilizza nomi o ambiti dei nomi XAML; USA invece chiavi, perché si tratta di un'implementazione del dizionario. L'unico motivo per cui `ResourceDictionary` implementa `INamespaceScope` è la possibilità di generare eccezioni al codice utente che consentono di chiarire la distinzione tra un vero e proprio ambito dei nomi XAML e il modo in cui un `ResourceDictionary` gestisce le chiavi e garantisce inoltre che gli ambiti dei nomi XAML non vengano applicati a un `ResourceDictionary` dagli elementi padre.

[FrameworkTemplate](#) e [Style](#) implementano [INameScope](#) tramite definizioni esplicite dell'interfaccia. Le implementazioni esplicite consentono a questi ambiti dei nomi XAML di comportarsi convenzionalmente quando si accede tramite l'interfaccia [INameScope](#), ovvero come gli ambiti dei nomi XAML vengono comunicati da WPF processi interni. Tuttavia, le definizioni esplicite dell'interfaccia non fanno parte della superficie API convenzionale di [FrameworkTemplate](#) e [Style](#), perché raramente è necessario chiamare i metodi di [INameScope](#) su [FrameworkTemplate](#) e [Style](#) direttamente e usare invece altre API come [GetTemplateChild](#).

Le classi seguenti definiscono il proprio ambito dei nomi XAML, usando la classe helper [System.Windows.NameScope](#) e connettendosi all'implementazione del NameScope XAML tramite la [NameScope.NameScope](#) proprietà associata:

- [FrameworkElement](#)
- [FrameworkContentElement](#)

Vedere anche

- [Spazi dei nomi XAML e mapping dello spazio dei nomi per XAML WPF](#)
- [Direttiva x:Name](#)

Stili e modelli inline

04/11/2019 • 3 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) fornisce oggetti di [Style](#) e oggetti modello ([FrameworkTemplate](#) sottoclassi) come metodo per definire l'aspetto visivo di un elemento nelle risorse, in modo che possano essere utilizzate più volte. Per questo motivo, gli attributi in XAML che accettano i tipi [Style](#) e [FrameworkTemplate](#) quasi sempre fanno riferimento a risorse e modelli esistenti anziché definire nuovi modelli inline.

Limitazioni degli stili e dei modelli in linea

In Extensible Application Markup Language (XAML), le proprietà dello stile e del modello possono tecnicamente essere impostate in uno dei due modi. È possibile utilizzare la sintassi degli attributi per fare riferimento a uno stile definito all'interno di una risorsa, ad esempio `<oggetto style="{StaticResource``}" .../>` [ResourceKey](#). In alternativa, è possibile usare la sintassi dell'elemento [Property](#) per definire uno stile inline, ad esempio:

`oggetto < >`

`oggetto < .Style>`

`< Style``.../>`

`oggetto </ .Style>`

`oggetto </ >`

L'utilizzo dell'attributo è molto più comune. Uno stile definito inline e non definito nelle risorse è necessariamente limitato all'elemento contenitore e non può essere riutilizzato con facilità perché non dispone di una chiave di risorsa. In generale, uno stile definito dalle risorse è più versatile e utile ed è più coerente con il principio generale del modello di programmazione Windows Presentation Foundation (WPF) della separazione della logica di programma nel codice dalla progettazione nel markup.

In genere non esiste alcun motivo per impostare uno stile o un modello inline, anche se si intende utilizzare solo lo stile o il modello in tale posizione. La maggior parte degli elementi che possono assumere uno stile o un modello supporta anche una proprietà di contenuto e un modello di contenuto. Se si usa qualsiasi albero logico creato con lo stile o il modello una sola volta, sarebbe ancora più semplice riempire la proprietà di contenuto con gli elementi figlio equivalenti nel markup diretto. In questo modo, il meccanismo dello stile e del modello verrà ignorato.

Sono inoltre possibili altre sintassi abilitate dalle estensioni di markup che restituiscono un oggetto per gli stili e i modelli. Due estensioni di questo tipo con scenari possibili includono [TemplateBinding](#) e [Binding](#).

Vedere anche

- [Applicazione di stili e modelli](#)

TypeConverter e XAML

10/02/2020 • 20 minutes to read • [Edit Online](#)

Questo argomento illustra lo scopo della conversione del tipo string come funzionalità generale del linguaggio XAML. Nella .NET Framework la classe [TypeConverter](#) svolge uno scopo specifico come parte dell'implementazione di una classe personalizzata gestita che può essere usata come valore della proprietà nell'utilizzo degli attributi XAML. Se si scrive una classe personalizzata e si vuole che le istanze della classe siano utilizzabili come valori di attributo impostabili XAML, potrebbe essere necessario applicare una [TypeConverterAttribute](#) alla classe, scrivere una classe [TypeConverter](#) personalizzata o entrambe.

Concetti relativi alla conversione di tipi

Valori XAML e stringa

Quando si imposta un valore di attributo in un file XAML, il tipo iniziale di tale valore è una stringa di solo testo. Anche altre primitive, ad esempio [Double](#), sono inizialmente stringhe di testo per un processore XAML.

Un processore XAML necessita di due informazioni per elaborare un valore di attributo. La prima informazione è il tipo di valore della proprietà che si imposta. Qualsiasi stringa che definisce un valore di attributo e che viene elaborata in XAML deve essere convertita o risolta in un valore di quel tipo. Se il valore è una primitiva riconosciuta dal parser XAML (ad esempio un valore numerico), viene tentata una conversione diretta della stringa. Se il valore è un'enumerazione, la stringa viene usata per controllare la corrispondenza di un nome con una costante denominata in tale enumerazione. Se il valore non è né una primitiva riconosciuta dal parser né un'enumerazione, il tipo in questione deve essere in grado di fornire un'istanza del tipo o un valore basato su una stringa convertita. Ciò è possibile indicando una classe di convertitore di tipi. Il convertitore di tipi è una classe helper che fornisce valori di un'altra classe, sia per gli scenari XAML che, potenzialmente, per le chiamate nel codice .NET.

Uso del comportamento di conversione dei tipi esistente in XAML

A seconda del livello di conoscenza dei concetti XAML sottostanti, è possibile che si usi già il comportamento di conversione dei tipi nel codice XAML dell'applicazione di base senza rendersene conto. WPF, ad esempio, definisce letteralmente centinaia di proprietà che accettano un valore di tipo [Point](#). Un [Point](#) è un valore che descrive una coordinata in uno spazio delle coordinate bidimensionale e ha solo due proprietà importanti: [X](#) e [Y](#). Quando si specifica un punto in XAML, lo si specifica come stringa con un delimitatore (in genere una virgola) tra i valori [X](#) e [Y](#) forniti. Ad esempio: `<LinearGradientBrush StartPoint="0,0" EndPoint="1,1"/>`.

Anche questo tipo semplice di [Point](#) e il relativo utilizzo semplice in XAML coinvolgono un convertitore di tipi. In questo caso è la classe [PointConverter](#).

Il convertitore di tipi per [Point](#) definito a livello di classe semplifica gli utilizzi di markup di tutte le proprietà che accettano [Point](#). Senza un convertitore di tipi, per l'esempio illustrato in precedenza sarebbe necessario ricorrere al seguente markup molto più dettagliato:

```
<LinearGradientBrush>
  <LinearGradientBrush.StartPoint>
    <Point X="0" Y="0"/>
  </LinearGradientBrush.StartPoint>
  <LinearGradientBrush.EndPoint>
    <Point X="1" Y="1"/>
  </LinearGradientBrush.EndPoint>
</LinearGradientBrush>
```

La scelta tra l'uso di una stringa di conversione del tipo o di una sintassi equivalente più dettagliata è, in genere, una questione di stile di codifica. Anche il flusso di lavoro degli strumenti XAML può influire sul modo in cui vengono impostati i valori. Alcuni strumenti XAML tendono a generare la forma più dettagliata del markup perché è più facile eseguire il round trip nelle visualizzazioni delle finestre di progettazione o nel meccanismo di serializzazione.

I convertitori di tipi esistenti possono in genere essere individuati in WPF e .NET Framework tipi controllando una classe (o una proprietà) per la presenza di un [TypeConverterAttribute](#) applicato. Questo attributo assegnerà un nome alla classe che costituisce il convertitore di tipi di supporto per i valori del tipo in questione, sia per gli scenari XAML sia, eventualmente, per altri scopi.

Convertitori di tipi ed estensioni di markup

Le estensioni di markup e i convertitori di tipi rivestono ruoli ortogonali in termini di comportamento del processore XAML e di scenari ai cui sono applicati. Anche se il contesto è disponibile per gli utilizzi di estensione di markup, il comportamento della conversione di tipi di proprietà in cui un'estensione di markup fornisce un valore in genere non è selezionato nelle implementazioni dell'estensione di markup. In altre parole, anche se un'estensione di markup restituisce una stringa di testo come output di `ProvideValue`, il comportamento di conversione dei tipi su tale stringa così come applicato a una proprietà o a un tipo di valore della proprietà specifico non viene richiamato. In genere, lo scopo di un'estensione di markup è quello di elaborare una stringa e restituire un oggetto senza coinvolgere alcun convertitore di tipi.

Una situazione comune che richiede un'estensione di markup invece che un convertitore di tipi è il riferimento a un oggetto già esistente. Nella migliore delle ipotesi, un convertitore di tipi senza stato può solo generare una nuova istanza, che potrebbe non essere il comportamento ottimale. Per altre informazioni sulle estensioni di markup, vedere [Estensioni di markup e XAML WPF](#).

Convertitori di tipi nativi

Nell'implementazione WPF e .NET Framework del parser XAML ci sono determinati tipi che prevedono la gestione nativa della conversione del tipo, anche se non si tratta di tipi convenzionalmente considerati primitive. Un esempio dei tipi in questione è [DateTime](#). Il motivo è basato sul funzionamento dell'architettura .NET Framework: il tipo [DateTime](#) è definito in mscorelib, la libreria più elementare di .NET, non è consentito assegnare a [DateTime](#) un attributo che deriva da un altro assembly che introduce una dipendenza ([TypeConverterAttribute](#) è dal sistema), quindi il consueto meccanismo di individuazione del convertitore di tipi mediante attribuzione non può essere supportato. Il parser XAML ha invece di un elenco di tipi che necessitano di tale elaborazione nativa ed elabora questi tipi in modo analogo all'elaborazione delle primitive effettive. Nel caso di [DateTime](#) ciò comporta una chiamata al [Parse](#).

Implementazione di un convertitore di tipi

TypeConverter

Nell'esempio [Point](#) fornito in precedenza, è stata indicata la classe [PointConverter](#). Per le implementazioni di .NET di XAML, tutti i convertitori di tipi usati per XAML sono classi che derivano dalla classe di base [TypeConverter](#). La classe [TypeConverter](#) esisteva nelle versioni di .NET Framework che precedono l'esistenza di XAML; uno degli usi originali è stato quello di fornire la conversione di stringa per le finestre di dialogo delle proprietà nelle finestre di progettazione visiva. Per XAML, il ruolo di [TypeConverter](#) viene espanso in modo da includere la classe di base per le conversioni da a stringa e da stringa che consentono l'analisi di un valore di attributo stringa ed eventualmente l'elaborazione di un valore di runtime di una particolare proprietà dell'oggetto in una stringa per la serializzazione come attributo.

[TypeConverter](#) definisce quattro membri rilevanti per la conversione da e verso stringhe per finalità di elaborazione XAML:

- [CanConvertTo](#)

- [CanConvertFrom](#)
- [ConvertTo](#)
- [ConvertFrom](#)

Di questi, il metodo più importante è [ConvertFrom](#). Questo metodo converte la stringa di input nel tipo di oggetto richiesto. In modo rigoroso, è possibile implementare il metodo [ConvertFrom](#) per convertire una gamma molto più ampia di tipi nel tipo di destinazione designato del convertitore e, di conseguenza, fornire finalità che si estendono oltre XAML, ad esempio il supporto delle conversioni in fase di esecuzione, ma per finalità XAML è solo il percorso del codice che può elaborare un input di [String](#) che è importante.

Il metodo più importante successivo è [ConvertTo](#). Se un'applicazione viene convertita in una rappresentazione del markup (ad esempio, se viene salvata in XAML come file), [ConvertTo](#) è responsabile della produzione di una rappresentazione di markup. In questo caso, il percorso di codice importante per XAML è quando si passa una `destinationType` di [String](#).

[CanConvertTo](#) e [CanConvertFrom](#) sono metodi di supporto usati quando un servizio esegue una query sulle funzionalità dell'implementazione di [TypeConverter](#). È necessario implementare questi metodi per restituire `true` per i casi specifici del tipo supportati dai metodi di conversione equivalenti del convertitore. Per XAML, si tratta in genere del tipo [String](#).

Informazioni relative alle impostazioni cultura e convertitori di tipi per XAML

Ogni implementazione di [TypeConverter](#) può avere una propria interpretazione di ciò che costituisce una stringa valida per una conversione e può anche usare o ignorare la descrizione del tipo passata come parametri. C'è un'importante considerazione per quanto riguarda le impostazioni cultura e la conversione di tipi in XAML. L'uso di stringhe localizzabili come valori di attributo è completamente supportato in XAML. L'uso di una stringa localizzabile come input del convertitore di tipi con requisiti di impostazioni cultura specifici non è però supportato, perché i convertitori di tipi per i valori di attributi XAML implicano un comportamento di analisi basato su un'unica lingua e sulle impostazioni cultura `en-us`. Per ulteriori informazioni sui motivi di progettazione di questa restrizione, consultare la specifica del linguaggio XAML ([\[MS-XAML\]](#)).

Un caso in cui le impostazioni cultura possono rappresentare un problema è ad esempio l'uso, in alcune impostazioni cultura, della virgola come separatore decimale per i numeri. Tale caratteristica è in conflitto con il comportamento di molti convertitori di tipi XAML di WPF, che usano la virgola come delimitatore (in base a precedenti storici come il formato X,Y comune o gli elenchi delimitati da virgole). Nemmeno il passaggio di impostazioni cultura nel codice XAML adiacente (impostando ad esempio `Language` o `xml:lang` sulle impostazioni cultura `s1-SI` che prevedono l'uso della virgola come separatore decimale) può consentire di risolvere il problema.

Implementazione di ConvertFrom

Per essere utilizzabile come implementazione di [TypeConverter](#) che supporti XAML, il metodo [ConvertFrom](#) per il convertitore deve accettare una stringa come parametro `value`. Se la stringa ha un formato valido e può essere convertita dall'implementazione di [TypeConverter](#), l'oggetto restituito deve supportare un cast nel tipo previsto dalla proprietà. In caso contrario, l'implementazione [ConvertFrom](#) deve restituire `null`.

Ogni implementazione di [TypeConverter](#) può avere una propria interpretazione di ciò che costituisce una stringa valida per una conversione e può anche usare o ignorare la descrizione del tipo o i contesti di impostazioni cultura passati come parametri. L'elaborazione della sintassi XAML di WPF potrebbe tuttavia non passare i valori al contesto della descrizione del tipo in tutti i casi e potrebbe non passare nemmeno le impostazioni cultura basate su `xml:lang`.

NOTE

Non usare i caratteri parentesi graffa, in particolare {}, come elemento del formato stringa. Questi caratteri vengono usati esclusivamente come punti di ingresso e di uscita di una sequenza di estensione del markup.

Implementazione di ConvertTo

[ConvertTo](#) viene potenzialmente usato per il supporto della serializzazione. Il supporto della serializzazione tramite [ConvertTo](#) per il tipo personalizzato e il relativo convertitore di tipi non è un requisito assoluto. Tuttavia, se si implementa un controllo o si usa la serializzazione come parte delle funzionalità o della progettazione della classe, sarà necessario implementare [ConvertTo](#).

Per essere utilizzabile come implementazione di [TypeConverter](#) che supporta XAML, il metodo [ConvertTo](#) per tale convertitore deve accettare un'istanza del tipo (o un valore) supportata come parametro `value`. Quando il `destinationType` parametro è il tipo [String](#), l'oggetto restituito deve essere in grado di eseguire il cast come [String](#). La stringa restituita deve rappresentare un valore serializzato di `value`. Idealmente, il formato di serializzazione scelto deve essere in grado di generare lo stesso valore se tale stringa venisse passata all'implementazione [ConvertFrom](#) dello stesso convertitore, senza perdite significative di informazioni.

Se il valore non può essere serializzato o il convertitore non supporta la serializzazione, l'implementazione del [ConvertTo](#) deve restituire `null` ed è consentito generare un'eccezione in questo caso. Tuttavia, se si generano eccezioni, è necessario segnalare l'impossibilità di usare tale conversione come parte dell'implementazione di [CanConvertTo](#) in modo che la procedura consigliata per verificare con [CanConvertTo](#) prima di evitare le eccezioni sia supportata.

Se `destinationType` parametro non è di tipo [String](#), è possibile scegliere la gestione del convertitore. In genere, viene ripristinata la gestione dell'implementazione di base, che nel [ConvertTo](#) basemost genera un'eccezione specifica.

Implementazione di CanConvertTo

L'implementazione di [CanConvertTo](#) deve restituire `true` per un oggetto `destinationType` di tipo [String](#); in caso contrario, deve rinviare all'implementazione di base.

Implementazione di CanConvertFrom

L'implementazione di [CanConvertFrom](#) deve restituire `true` per un oggetto `sourceType` di tipo [String](#); in caso contrario, deve rinviare all'implementazione di base.

Applicazione di TypeConverterAttribute

Affinché il convertitore di tipi personalizzato venga usato come convertitore di tipi di azione per una classe personalizzata da un processore XAML, è necessario applicare il [TypeConverterAttribute](#) alla definizione della classe. L'oggetto [ConverterTypeName](#) specificato tramite l'attributo deve corrispondere al nome di tipo del convertitore dei tipi personalizzato. Se si applica questo attributo, quando un processore XAML gestisce valori per i quali il tipo di proprietà usa il tipo di classe personalizzato, può usare stringhe di input e restituire istanze di oggetti.

Si può anche fornire un convertitore dei tipi per le singole proprietà. Anziché applicare una [TypeConverterAttribute](#) alla definizione della classe, applicarla a una definizione di proprietà (la definizione principale, non la `get` / le implementazioni `set` al suo interno). Il tipo della proprietà deve corrispondere al tipo elaborato dal convertitore dei tipi personalizzato. Se questo attributo viene applicato, quando un processore XAML gestisce i valori di quella proprietà, può elaborare stringhe di input e restituire istanze di oggetti. La tecnica del convertitore dei tipi per proprietà è particolarmente utile se si sceglie di usare un tipo di proprietà da Microsoft .NET Framework o da un'altra libreria in cui non è possibile controllare la definizione della classe e non è possibile applicare un [TypeConverterAttribute](#).

Vedere anche

- [TypeConverter](#)
- [Panoramica di XAML \(WPF\)](#)
- [Estensioni di markup e XAML WPF](#)
- [Descrizione dettagliata della sintassi XAML](#)

Estensioni XAML WPF

03/02/2020 • 2 minutes to read • [Edit Online](#)

Contenuto della sezione

[Estensione di markup Binding](#)

[Estensione di markup ColorConvertedBitmap](#)

[Estensione di markup ComponentResourceKey](#)

[Estensione del markup DynamicResource](#)

[Estensione di markup RelativeSource](#)

[Estensione di markup StaticResource](#)

[Estensione di markup TemplateBinding](#)

[Estensione di markup ThemeDictionary](#)

[Sintassi XAML di PropertyPath](#)

[Attributo PresentationOptions:Freeze](#)

Associazione dell'estensione di markup

04/11/2019 • 15 minutes to read • [Edit Online](#)

Rinvia un valore di proprietà a un valore associato a dati, creando un oggetto espressione intermedia e interpretando il contesto dei dati che si applica all'elemento e la relativa associazione in fase di esecuzione.

Utilizzo delle espressioni di binding

```
<object property="{Binding}" .../>
-or-
<object property="{Binding bindProp1=value1[, bindPropN=valueN]*}" ...
/>
-or-
<object property="{Binding path}" .../>
-or
<object property="{Binding path[, bindPropN=valueN]*}" .../>
```

Note sulla sintassi

In queste sintassi, il `[]` e il `*` non sono valori letterali. Sono parte di una notazione per indicare che è possibile usare zero o più coppie `= valore bindProp`, con un separatore di `,` tra di essi e le coppie di `valori = bindProp` precedenti.

Una delle proprietà elencate nella sezione "proprietà di binding che è possibile impostare con l'estensione di binding" può invece essere impostata utilizzando gli attributi di un `Binding` elemento oggetto. Tuttavia, non si tratta effettivamente dell'utilizzo dell'estensione di markup di `Binding`, ma è solo l'elaborazione XAML generale degli attributi che impostano le proprietà della classe `Binding` CLR. In altre parole, `<Binding bindProp1 ="value1 "[bindPropN =" valore "]*/>` è una sintassi equivalente per gli attributi di `Binding` utilizzo dell'elemento oggetto invece di un utilizzo di espressione `Binding`. Per informazioni sull'utilizzo degli attributi XAML di proprietà specifiche di `Binding`, vedere la sezione "utilizzo degli attributi XAML" della proprietà pertinente di `Binding` nella libreria di classi di .NET Framework.

Valori XAML

<code>bindProp1, bindPropN</code>	Nome della proprietà <code>Binding</code> o <code>BindingBase</code> da impostare. Non tutte le proprietà di <code>Binding</code> possono essere impostate con l'estensione <code>Binding</code> e alcune proprietà possono essere impostate all'interno di un'espressione <code>Binding</code> solo utilizzando altre estensioni di markup nidificate. Vedere la sezione relativa alle proprietà di binding che possono essere impostate con l'estensione di binding.
<code>value1, valueN</code>	Valore su cui impostare la proprietà. La gestione del valore dell'attributo è in definitiva specifica per il tipo e la logica della proprietà <code>Binding</code> specifica da impostare.
<code>path</code>	Stringa di percorso che imposta la proprietà <code>Binding.Path</code> implicita. Vedere anche sintassi XAML di PropertyPath .

{Binding} non qualificato

Il `{Binding}` utilizzo illustrato in "binding espressione utilizzo" crea un oggetto [Binding](#) con i valori predefiniti, che include un [Binding.Path](#) iniziale di `null`. Questo è ancora utile in molti scenari, perché il [Binding](#) creato potrebbe basarsi sulle proprietà data binding chiave, ad esempio [Binding.Path](#) e [Binding.Source](#) essere impostate nel contesto dei dati di run-time. Per ulteriori informazioni sul concetto di contesto dati, vedere [Data Binding](#).

Percorso implicito

L'estensione di markup `Binding` utilizza [Binding.Path](#) come proprietà predefinita concettuale, in cui non è necessario che `Path=` venga visualizzata nell'espressione. Se si specifica un'espressione di `Binding` con un percorso implicito, il percorso implicito deve essere visualizzato per primo nell'espressione, prima di qualsiasi altra `bindProp = value` coppie in cui la proprietà [Binding](#) viene specificata in base al nome. Ad esempio: `{Binding PathString}`, dove `PathString` è una stringa che viene valutata come il valore di [Binding.Path](#) nel [Binding](#) creato dall'utilizzo dell'estensione di markup. È possibile aggiungere un percorso implicito con altre proprietà denominate dopo il separatore virgola, ad esempio `{Binding LastName, Mode=TwoWay}`.

Proprietà di binding che possono essere impostate con l'estensione di binding

La sintassi illustrata in questo argomento usa l'`bindProp generica = value` approssimazione, perché sono disponibili molte proprietà di lettura/scrittura di [BindingBase](#) o [Binding](#) che possono essere impostate tramite la sintassi dell'espressione o dell'estensione di markup `Binding`. Possono essere impostate in qualsiasi ordine, ad eccezione di un [Binding.Path](#) implicito. È possibile specificare in modo esplicito `Path=`, nel qual caso è possibile impostarlo in qualsiasi ordine. Fondamentalmente, è possibile impostare zero o più proprietà nell'elenco seguente, usando `bindProp =coppie di value` separate da virgolette.

Molti di questi valori di proprietà richiedono tipi di oggetto che non supportano la conversione di tipi nativi da una sintassi di testo in XAML e pertanto richiedono le estensioni di markup per essere impostate come valore di attributo. Per ulteriori informazioni, controllare la sezione Utilizzo degli attributi XAML nella libreria di classi .NET Framework per ogni proprietà. La stringa usata per la sintassi dell'attributo XAML con o senza ulteriore utilizzo dell'estensione di markup è fondamentalmente uguale al valore specificato in un'espressione di `Binding`, con l'eccezione che non si inseriscono le virgolette intorno a ogni `bindProp = value` espressione `Binding`.

- [BindingGroupName](#): stringa che identifica un possibile gruppo di associazioni. Si tratta di un concetto di associazione relativamente avanzato. Vedere la pagina di riferimento per [BindingGroupName](#).
- [BindsDirectlyToSource](#): booleano, può essere `true` o `false`. Il valore predefinito è `false`.
- [Converter](#): può essere impostato come `bindProp =stringa value` nell'espressione, ma a tale scopo è necessario un riferimento a un oggetto per il valore, ad esempio un' [estensione di markup StaticResource](#). Il valore in questo caso è un'istanza di una classe Converter personalizzata.
- [ConverterCulture](#): impostabile nell'espressione come identificatore basato su standard; per [ConverterCulture](#), vedere l'argomento di riferimento.
- [ConverterParameter](#): può essere impostato come `bindProp =stringa value` nell'espressione, ma dipende dal tipo di parametro passato. Se si passa un tipo riferimento per il valore, questo utilizzo richiede un riferimento a un oggetto, ad esempio un' [estensione di markup StaticResource](#) annidata.
- [ElementName](#): si escludono a vicenda rispetto a [RelativeSource](#) e [Source](#); ognuna di queste proprietà di associazione rappresenta una particolare metodologia di associazione. Vedere [Cenni preliminari sul data binding](#).
- [FallbackValue](#): può essere impostato come `bindProp =stringa value` nell'espressione, ma dipende dal tipo

del valore passato. Se si passa un tipo riferimento, richiede un riferimento a un oggetto, ad esempio un' estensione di markup StaticResource annidata.

- **IsAsync**: booleano, può essere `true` o `false`. Il valore predefinito è `false`.
- **Mode**: *value* è un nome di costante dell'enumerazione [BindingMode](#). Ad esempio `{Binding Mode=OneWay}`.
- **NotifyOnSourceUpdated**: booleano, può essere `true` o `false`. Il valore predefinito è `false`.
- **NotifyOnTargetUpdated**: booleano, può essere `true` o `false`. Il valore predefinito è `false`.
- **NotifyOnValidationError**: booleano, può essere `true` o `false`. Il valore predefinito è `false`.
- **Path**: stringa che descrive un percorso in un oggetto dati o in un modello a oggetti generale. Il formato fornisce diverse convenzioni per l'attraversamento di un modello a oggetti che non può essere descritto in modo adeguato in questo argomento. Vedere [sintassi XAML di PropertyPath](#).
- **RelativeSource**: si escludono a vicenda rispetto a [ElementName](#) e [Source](#); ognuna di queste proprietà di associazione rappresenta una particolare metodologia di associazione. Vedere [Cenni preliminari sul data binding](#). Richiede un utilizzo [MarkupExtension](#) di [RelativeSource](#) annidato per specificare il valore.
- **Source**: si escludono a vicenda rispetto a [RelativeSource](#) e [ElementName](#); ognuna di queste proprietà di associazione rappresenta una particolare metodologia di associazione. Vedere [Cenni preliminari sul data binding](#). Richiede un utilizzo dell'estensione annidato, in genere un' estensione di markup StaticResource che fa riferimento a un'origine dati oggetto da un dizionario risorse con chiave.
- **StringFormat**: stringa che descrive una convenzione di formato stringa per i dati associati. Si tratta di un concetto di associazione relativamente avanzato. vedere la pagina di riferimento per [StringFormat](#).
- **TargetNullValue**: può essere impostato come `bindProp =stringa value` nell'espressione, ma dipende dal tipo di parametro passato. Se si passa un tipo riferimento per il valore, richiede un riferimento a un oggetto, ad esempio un' estensione di markup StaticResource annidata.
- **UpdateSourceTrigger**: *value* è un nome di costante dell'enumerazione [UpdateSourceTrigger](#). Ad esempio `{Binding UpdateSourceTrigger=LostFocus}`. I controlli specifici hanno potenzialmente valori predefiniti diversi per questa proprietà di associazione. Vedere [UpdateSourceTrigger](#).
- **ValidatesOnDataErrors**: booleano, può essere `true` o `false`. Il valore predefinito è `false`. Vedere la sezione Osservazioni.
- **ValidatesOnExceptions**: booleano, può essere `true` o `false`. Il valore predefinito è `false`. Vedere la sezione Osservazioni.
- **XPath**: stringa che descrive un percorso in XMLDOM di un'origine dati XML. Vedere [eseguire l'associazione a dati XML tramite un oggetto XmlDataProvider e query XPath](#).

Di seguito sono riportate le proprietà di [Binding](#) che non è possibile impostare utilizzando il form `Binding` Markup Extension/`{Binding}` Expression.

- **UpdateSourceExceptionFilter**: questa proprietà prevede un riferimento a un'implementazione di callback. Non è possibile fare riferimento a callback/metodi diversi dai gestori eventi nella sintassi XAML.
- **ValidationRules**: la proprietà accetta una raccolta generica di oggetti [ValidationRule](#). Questo può essere espresso come elemento proprietà in un elemento [Binding](#) oggetto, ma non dispone di una tecnica di analisi degli attributi prontamente disponibile per l'utilizzo in un'espressione `Binding`. Per [ValidationRules](#), vedere l'argomento di riferimento.
- **XmlNamespaceManager**

Note

IMPORTANT

In termini di precedenza delle proprietà di dipendenza, un'espressione `Binding` è equivalente a un valore impostato localmente. Se si imposta un valore locale per una proprietà che in precedenza aveva un'espressione `Binding`, il `Binding` viene completamente rimosso. Per altri dettagli, vedere [Precedenza del valore della proprietà di dipendenza](#).

La descrizione di data binding a un livello di base non è descritta in questo argomento. Vedere [Cenni preliminari sul data binding](#).

NOTE

`MultiBinding` e `PriorityBinding` non supportano una sintassi dell'estensione di XAML. Si utilizzeranno invece gli elementi `Property`. Per `MultiBinding` e `PriorityBinding`, vedere gli argomenti di riferimento.

I valori booleani per XAML non fanno distinzione tra maiuscole e minuscole. È ad esempio possibile specificare `{Binding NotifyOnValidationError=true}` o `{Binding NotifyOnValidationError=True}`.

Le associazioni che coinvolgono la convalida dei dati vengono in genere specificate da un elemento `Binding` esplicito anziché come espressione `{Binding ...}` e l'impostazione di `ValidatesOnDataErrors` o `ValidatesOnExceptions` in un'espressione non è comune. Ciò è dovuto al fatto che la proprietà complementare `ValidationRules` non può essere impostata immediatamente nel formato di espressione. Per altre informazioni, vedere [implementare la convalida dell'associazione](#).

`Binding` è un'estensione di markup. Le estensioni di markup vengono in genere implementate quando è necessario eseguire l'escape dei valori di attributo in modo che non siano valori letterali o nomi di gestori e il requisito è più globale rispetto ai convertitori di tipi attribuiti a determinati tipi o proprietà. Tutte le estensioni di markup in XAML usano i caratteri `{` e `}` nella sintassi degli attributi, ovvero la convenzione con cui un processore XAML riconosce che un'estensione di markup deve elaborare il contenuto della stringa. Per altre informazioni, vedere [Estensioni di markup e XAML WPF](#).

`Binding` è un'estensione di markup atipica in quanto la classe `Binding` che implementa la funzionalità di estensione per l'implementazione XAML di WPF implementa anche diversi altri metodi e proprietà non correlati a XAML. Gli altri membri hanno lo scopo di creare `Binding` una classe più versatile e autonoma in grado di risolvere molti scenari di data binding, oltre a funzionare come estensione di markup XAML.

Vedere anche

- [Binding](#)
- [Panoramica sul data binding](#)
- [Cenni preliminari su XAML \(WPF\)](#)
- [Estensioni di markup e XAML WPF](#)

Estensione del markup ColorConvertedBitmap

23/10/2019 • 2 minutes to read • [Edit Online](#)

Fornisce un modo per specificare un'origine bitmap che non dispone di un profilo incorporato. I contesti di colore/profilo sono specificati dall'URI, così come l'URI di origine dell'immagine.

Uso della sintassi XAML per gli attributi

```
<object property="{ColorConvertedBitmap imageSource sourceIIC destinationIIC}" .../>
```

Valori XAML

<code>imageSource</code>	URI della bitmap non profilata.
<code>sourceIIC</code>	URI della configurazione del profilo di origine.
<code>destinationIIC</code>	URI della configurazione del profilo di destinazione

Note

Questa estensione di markup è destinata a compilare un set correlato di valori di proprietà di origine dell'immagine, ad esempio [UriSource](#).

La sintassi per gli attributi è quella più comunemente utilizzata con questa estensione di markup. non è possibile usare `ColorConvertedBitmap` (o `ColorConvertedBitmapExtension`) nella sintassi dell'elemento Property perché i valori possono essere impostati solo come valori nel costruttore iniziale, che è la stringa che segue l'identificatore dell'estensione.

`ColorConvertedBitmap` è un'estensione di markup. Le estensioni di markup in genere vengono implementate quando per i valori dell'attributo devono essere utilizzati caratteri escape in modo che non vengano considerati come valori letterali o nomi di gestori e il requisito è più globale del semplice utilizzo di convertitori dei tipi su alcuni tipi o proprietà. Tutte le estensioni di markup in XAML usano i caratteri { e } nella sintassi degli attributi, vale a dire la convenzione in base a cui il processore XAML riconosce che l'attributo deve essere elaborato da un'estensione di markup. Per altre informazioni, vedere [Estensioni di markup e XAML WPF](#).

Vedere anche

- [UriSource](#)
- [Estensioni di markup e XAML WPF](#)
- [Cenni preliminari sulla creazione dell'immagine](#)

Estensione del markup ComponentResourceKey

08/01/2020 • 6 minutes to read • [Edit Online](#)

Definisce e fa riferimento alle chiavi per le risorse caricate da assembly esterni. Ciò consente a una ricerca di risorse di specificare un tipo di destinazione in un assembly, anziché un dizionario risorse esplicito in un assembly o in una classe.

Utilizzo degli attributi XAML (impostazione chiave, compatta)

```
<object x:Key="{ComponentResourceKey {x:Type targetTypeName}, targetID}" .../>
```

Utilizzo degli attributi XAML (impostazione della chiave, verbose)

```
<object x:Key="{ComponentResourceKey TypeInTargetAssembly={x:Type targetTypeName}, ResourceID=targetID}" .../>
```

Utilizzo degli attributi XAML (risorsa richiesta, compatta)

```
<object property="{DynamicResource {ComponentResourceKey {x:Type targetTypeName}, targetID}}" .../>
```

Utilizzo degli attributi XAML (richiesta di risorse, verbose)

```
<object property="{DynamicResource {ComponentResourceKey TypeInTargetAssembly={x:Type targetTypeName}, ResourceID=targetID}}" .../>
```

Valor XAML

<code>targetTypeName</code>	Nome del tipo di Common Language Runtime pubblico (CLR) definito nell'assembly di risorse.
<code>targetID</code>	Chiave per la risorsa. Quando le risorse vengono cercate, <code>targetID</code> sarà analogo alla direttiva x:Key della risorsa.

Note

Come illustrato negli usi precedenti, un utilizzo dell'estensione di markup `{ ComponentResourceKey }` si trova in due posizioni:

- Definizione di una chiave all'interno di un dizionario risorse del tema, fornita da un autore del controllo.
- Accesso a una risorsa del tema dall'assembly, quando si crea il modello del controllo, ma si desidera utilizzare i valori delle proprietà provenienti dalle risorse fornite dai temi del controllo.

Per fare riferimento alle risorse dei componenti che provengono dai temi, è in genere consigliabile usare

`{DynamicResource}` anziché `{StaticResource}`. Questa operazione viene illustrata negli utilizzi. È consigliabile `{DynamicResource}` perché il tema stesso può essere modificato dall'utente. Se si vuole che la risorsa componente che corrisponda maggiormente alla finalità dell'autore del controllo per il supporto di un tema, è necessario abilitare anche il riferimento a una risorsa di componente dinamica.

Il `TypeInTargetAssembly` identifica un tipo esistente nell'assembly di destinazione in cui la risorsa è effettivamente definita. Un `ComponentResourceKey` può essere definito e utilizzato indipendentemente dal modo in cui viene definita l'`TypeInTargetAssembly`, ma è necessario risolvere il tipo tramite assembly a cui si fa riferimento.

Un utilizzo comune per `ComponentResourceKey` consiste nel definire le chiavi che vengono quindi esposte come membri di una classe. Per questo utilizzo si usa il costruttore della classe `ComponentResourceKey`, non l'estensione di markup. Per ulteriori informazioni, vedere `ComponentResourceKey` o la sezione "definizione e riferimento alle chiavi per le risorse del tema" dell'argomento [Cenni preliminari sulla creazione di controlli](#).

Per la creazione di chiavi e per il riferimento a risorse con chiave, la sintassi degli attributi viene comunemente utilizzata per l'estensione di markup `ComponentResourceKey`.

La sintassi Compact illustrata si basa sulla firma del costruttore `ComponentResourceKey.ComponentResourceKey` e sull'utilizzo dei parametri posizionali di un'estensione di markup. L'ordine in cui vengono forniti i `targetTypeName` e `targetID` è importante. La sintassi dettagliata si basa sul `ComponentResourceKey.ComponentResourceKey` costruttore senza parametri, quindi imposta le `TypeInTargetAssembly` e `ResourceID` in modo analogo a una sintassi di attributo true su un elemento oggetto. Nella sintassi Verbose, l'ordine in cui le proprietà sono impostate non è importante. La relazione e i meccanismi di queste due alternative (Compact e Verbose) vengono descritti in modo più dettagliato nell'argomento [estensioni di markup e XAML WPF](#).

Tecnicamente, il valore per `targetID` può essere qualsiasi oggetto, non è necessario che sia una stringa. Tuttavia, l'utilizzo più comune in WPF consiste nell'allineare il valore `targetID` con i nomi che sono stringhe e in cui tali stringhe sono valide nella [Grammatica XamlName](#).

`ComponentResourceKey` possibile utilizzare la sintassi dell'elemento oggetto. In questo caso, è necessario specificare il valore delle proprietà `TypeInTargetAssembly` e `ResourceID` per inizializzare correttamente l'estensione.

Nell'implementazione WPF XAML Reader, la gestione di questa estensione di markup viene definita dalla classe `ComponentResourceKey`.

`ComponentResourceKey` è un'estensione di markup. Le estensioni di markup in genere vengono implementate quando per i valori dell'attributo devono essere utilizzati caratteri escape in modo che non vengano considerati come valori letterali o nomi di gestori e il requisito è più globale del semplice utilizzo di convertitori dei tipi su alcuni tipi o proprietà. Tutte le estensioni di markup in XAML usano i caratteri { e } nella sintassi degli attributi, vale a dire la convenzione in base a cui il processore XAML riconosce che l'attributo deve essere elaborato da un'estensione di markup. Per altre informazioni, vedere [Estensioni di markup e XAML WPF](#).

Vedere anche

- [ComponentResourceKey](#)
- [ControlTemplate](#)
- [Cenni preliminari sulla modifica di controlli](#)
- [Cenni preliminari su XAML \(WPF\)](#)
- [Estensioni di markup e XAML WPF](#)

Sintassi DateTime XAML

04/11/2019 • 7 minutes to read • [Edit Online](#)

Alcuni controlli, ad esempio [Calendar](#) e [DatePicker](#), hanno proprietà che usano il tipo di [DateTime](#). Anche se una data o un'ora iniziale per questi controlli viene in genere specificata nel code-behind in fase di esecuzione, è possibile specificare una data o un'ora iniziale in XAML. Il parser XAML WPF gestisce l'analisi dei valori [DateTime](#) usando una sintassi del testo XAML incorporata. In questo argomento vengono descritte le specifiche della sintassi del testo XAML [DateTime](#).

Quando usare la sintassi DateTime XAML

Impostare le date in XAML non è sempre necessario e può anche non essere appropriato. Ad esempio, è possibile usare la proprietà [DateTime.Now](#) per inizializzare una data in fase di esecuzione oppure è possibile eseguire tutte le regolazioni della data per un calendario nel code-behind in base all'input dell'utente. Esistono tuttavia scenari in cui è possibile che si desideri impostare come hardcoded le date in una [Calendar](#) e [DatePicker](#) in un modello di controllo. Per questi scenari è necessario usare la sintassi XAML [DateTime](#).

La sintassi DateTime di XAML è un comportamento nativo

[DateTime](#) è una classe definita nelle librerie di classi di base di CLR. A causa del modo in cui le librerie di classi di base sono correlate al resto di CLR, non è possibile applicare [TypeConverterAttribute](#) alla classe e usare un convertitore di tipi per elaborare le stringhe da XAML e convertirle in [DateTime](#) nel modello a oggetti di run-time. Non esiste una classe [DateTimeConverter](#) che fornisce il comportamento di conversione. Il comportamento di conversione descritto in questo argomento è nativo del parser XAML WPF.

Stringhe di formato per la sintassi DateTime XAML

È possibile specificare il formato di un [DateTime](#) con una stringa di formato. Le stringhe di formato formalizzano la sintassi del testo che può essere usata per creare un valore. [DateTime](#) valori per i controlli WPF esistenti in genere utilizzano solo i componenti Data di [DateTime](#) e non i componenti dell'ora.

Quando si specifica un [DateTime](#) in XAML, è possibile usare qualsiasi stringa di formato in modo interscambiabile.

È anche possibile usare formati e stringhe di formato non illustrati in maniera specifica in questo argomento. Tecnicamente, il codice XAML per qualsiasi valore [DateTime](#) specificato e quindi analizzato dal parser XAML WPF usa una chiamata interna a [DateTime.Parse](#), quindi è possibile usare qualsiasi stringa accettata da [DateTime.Parse](#) per l'input XAML. Per ulteriori informazioni, vedere [DateTime.Parse](#).

IMPORTANT

La sintassi XAML [DateTime](#) usa sempre [en-us](#) come [CultureInfo](#) per la conversione nativa. Questo non è influenzato dal valore [Language](#) o dal valore [xml:lang](#) in XAML, perché la conversione del tipo a livello di attributo XAML agisce senza tale contesto. Non tentare di interpolare le stringhe di formato mostrate qui a causa di variazioni relative alla lingua, come l'ordine di visualizzazione di giorno e mese. Le stringhe di formato mostrate qui sono le stringhe di formato esatte usate durante l'analisi del codice XAML indipendentemente dalle altre impostazioni cultura.

Le sezioni seguenti descrivono alcune delle stringhe di formato [DateTime](#) comune.

Schema di data breve ("d")

Di seguito viene illustrato il formato di data breve per un [DateTime](#) in XAML:

M/d/YYYY

Questo è il formato più semplice che specifica tutte le informazioni necessarie per gli utilizzi tipici per i controlli WPF e non è influenzato dalle differenze di fuso orario rispetto a un componente relativo all'ora, quindi è consigliato rispetto altri formati.

Ad esempio, per specificare la data del 1 giugno 2010, usare la stringa seguente:

```
3/1/2010
```

Per ulteriori informazioni, vedere [DateTimeFormatInfo.ShortDatePattern](#).

Schema di data/ora ordinabile ("s")

Di seguito viene illustrato il modello di [DateTime](#) ordinabile in XAML:

```
yyyy'-'MM'-'dd'T'HH':'mm':'ss
```

Ad esempio, per specificare la data del 1 giugno 2010, usare la stringa seguente (tutti i componenti relativi all'ora vengono immessi come 0):

```
2010-06-01T000:00:00
```

Schema RFC1123 ("r")

Il modello RFC1123 è utile perché potrebbe essere un input di stringa da altri generatori di data che usano il modello RFC1123 per via delle impostazioni cultura invariabili. Di seguito viene illustrato il modello di [DateTime](#) RFC1123 in XAML:

```
ddd, dd MMM yyyy HH':'mm':'ss 'UTC'
```

Ad esempio, per specificare la data del 1 giugno 2010, usare la stringa seguente (tutti i componenti relativi all'ora vengono immessi come 0):

```
Mon, 01 Jun 2010 00:00:00 UTC
```

Altri formati e modelli

Come indicato in precedenza, è possibile specificare un [DateTime](#) in XAML come qualsiasi stringa accettabile come input per [DateTime.Parse](#). Sono inclusi altri formati formalizzati (ad esempio [UniversalSortableDateTimePattern](#)) e formati che non sono formalizzati come un particolare [DateTimeFormatInfo](#) formato. Il modulo `yyyy/mm/dd`, ad esempio, è accettabile come input per [DateTime.Parse](#). Questo argomento non tenta di descrivere tutti i possibili formati che funzionano e invece consiglia il formato di data breve come pratica standard.

Vedere anche

- [Cenni preliminari su XAML \(WPF\)](#)

Estensione del markup DynamicResource

08/01/2020 • 6 minutes to read • [Edit Online](#)

Fornisce un valore per qualsiasi attributo di proprietà XAML rinviando tale valore in modo che sia un riferimento a una risorsa definita. Il comportamento di ricerca per tale risorsa è analogo alla ricerca in fase di esecuzione.

Utilizzo della sintassi XAML per attributi

```
<object property="{DynamicResource key}" .../>
```

Utilizzo della sintassi XAML per elementi proprietà

```
<object>
  <object.property>
    <DynamicResource ResourceKey="key" .../>
  </object.property>
</object>
```

Valor XAML

key

Chiave per la risorsa richiesta. Questa chiave è stata inizialmente assegnata dalla [direttiva x:Key](#) se una risorsa è stata creata nel markup oppure è stata fornita come parametro di `key` quando viene chiamata [ResourceDictionary.Add](#) se la risorsa è stata creata nel codice.

Note

Un `dynamicResource` creerà un'espressione temporanea durante la compilazione iniziale e quindi rinvia la ricerca delle risorse finché il valore della risorsa richiesta non è effettivamente necessario per costruire un oggetto.

Questo problema potrebbe essere dovuto al caricamento della pagina XAML. Il valore della risorsa verrà trovato in base alla ricerca chiave eseguita su tutti i dizionari risorse attivi a partire dall'ambito della pagina corrente e sostituisce l'espressione segnaposto dalla compilazione.

IMPORTANT

In termini di precedenza delle proprietà di dipendenza, un'espressione `DynamicResource` equivale alla posizione in cui viene applicato il riferimento a una risorsa dinamica. Se si imposta un valore locale per una proprietà che in precedenza aveva un'espressione `DynamicResource` come valore locale, il `DynamicResource` verrà completamente rimosso. Per altri dettagli, vedere [Precedenza del valore della proprietà di dipendenza](#).

Alcuni scenari di accesso alle risorse sono particolarmente appropriati per `DynamicResource` anziché un'[estensione di markup StaticResource](#). Vedere [risorse XAML](#) per una discussione sui meriti relativi e sulle implicazioni relative alle prestazioni di `DynamicResource` e `StaticResource`.

Il `ResourceKey` specificato deve corrispondere a una risorsa esistente determinata dalla [direttiva x:Key](#) a un certo

livello nella pagina, applicazione, i temi di controllo disponibili e le risorse esterne, o risorse di sistema, e la ricerca delle risorse verrà eseguita in tale ordine. Per altre informazioni sulla ricerca di risorse per le risorse statiche e dinamiche, vedere [risorse XAML](#).

Una chiave di risorsa può essere qualsiasi stringa definita nella [Grammatica XamlName](#). Una chiave di risorsa può essere anche altri tipi di oggetto, ad esempio un [Type](#). Una chiave [Type](#) è fondamentale per il modo in cui i controlli possono essere disegnati in base ai temi. Per altre informazioni, vedere [Cenni preliminari sulla modifica di controlli](#).

Le API per la ricerca dei valori delle risorse, ad esempio [FindResource](#), seguono la stessa logica di ricerca delle risorse usata da [DynamicResource](#).

Il mezzo dichiarativo alternativo per fare riferimento a una risorsa è come [estensione di markup StaticResource](#).

La sintassi per gli attributi è quella più comunemente utilizzata con questa estensione di markup. Il token di stringa fornito dopo la stringa dell'identificatore [DynamicResource](#) viene assegnato come valore [ResourceKey](#) della classe dell'estensione [DynamicResourceExtension](#) sottostante.

[DynamicResource](#) possibile utilizzare la sintassi dell'elemento oggetto. In questo caso, è necessario specificare il valore della proprietà [ResourceKey](#).

L'oggetto [DynamicResource](#) può anche essere utilizzato per un utilizzo dettagliato degli attributi che consente di specificare la proprietà [ResourceKey](#) come coppia proprietà=valore:

```
<object property="{DynamicResource ResourceKey=key}" .../>
```

L'utilizzo dettagliato spesso è utile per le estensioni con più proprietà da impostare o nel caso in cui alcune proprietà siano facoltative. Poiché [DynamicResource](#) presenta una sola proprietà da impostare, obbligatoria, l'utilizzo dettagliato non è tipico.

Nell'implementazione del processore WPF XAML, la gestione di questa estensione di markup viene definita dalla classe [DynamicResourceExtension](#).

[DynamicResource](#) è un'estensione di markup. Le estensioni di markup in genere vengono implementate quando per i valori dell'attributo devono essere utilizzati caratteri escape in modo che non vengano considerati come valori letterali o nomi di gestori e il requisito è più globale del semplice utilizzo di convertitori dei tipi su alcuni tipi o proprietà. Tutte le estensioni di markup in XAML usano i caratteri { e } nella sintassi degli attributi, vale a dire la convenzione in base a cui il processore XAML riconosce che l'attributo deve essere elaborato da un'estensione di markup. Per altre informazioni, vedere [Estensioni di markup e XAML WPF](#).

Vedere anche

- [Risorse XAML](#)
- [Risorse e codice](#)
- [Direttiva x:Key](#)
- [Cenni preliminari su XAML \(WPF\)](#)
- [Estensioni di markup e XAML WPF](#)
- [Estensione di markup StaticResource](#)
- [Estensioni di markup e XAML WPF](#)

Estensione del markup RelativeSource

10/01/2020 • 8 minutes to read • [Edit Online](#)

Specifica le proprietà di un'origine di associazione [RelativeSource](#), da utilizzare in un' [estensione di markup dell'associazione](#) quando si imposta la proprietà [RelativeSource](#) di un elemento [Binding](#) definito in XAML.

Utilizzo della sintassi XAML per attributi

```
<Binding RelativeSource="{RelativeSource modeEnumValue}" .../>
```

Utilizzo della sintassi XAML per gli attributi (annidati nell'estensione Binding)

```
<object property="{Binding RelativeSource={RelativeSource modeEnumValue} ...}" .../>
```

Utilizzo della sintassi XAML per elementi oggetto

```
<Binding>
  <Binding.RelativeSource>
    <RelativeSource Mode="modeEnumValue"/>
  </Binding.RelativeSource>
</Binding>
```

oppure

```
<Binding>
  <Binding.RelativeSource>
    <RelativeSource
      Mode="FindAncestor"
      AncestorType="{x:Type typeName}"
      AncestorLevel="intLevel"
    />
  </Binding.RelativeSource>
</Binding>
```

Valor XAML

<p><code>modeEnumValue</code></p>	<p>Uno dei seguenti:</p> <ul style="list-style-type: none"> : Token di stringa <code>Self</code> ; corrisponde a un RelativeSource creato con la relativa proprietà Mode impostata su <code>Self</code>. : Token di stringa <code>TemplatedParent</code> ; corrisponde a un RelativeSource creato con la relativa proprietà Mode impostata su <code>TemplatedParent</code>. : Token di stringa <code>PreviousData</code> ; corrisponde a un RelativeSource creato con la relativa proprietà Mode impostata su <code>PreviousData</code>. <p>Per informazioni sulla modalità <code>FindAncestor</code>, vedere di seguito.</p>
<p><code>FindAncestor</code></p>	<p>Token stringa <code>FindAncestor</code> . L'utilizzo di questo token consente di accedere a una modalità in cui RelativeSource specifica un tipo predecessore e facoltativamente un livello predecessore. Corrisponde a un oggetto RelativeSource creato con la proprietà Mode impostata su <code>FindAncestor</code>.</p>
<p><code>typeName</code></p>	<p>Obbligatorio per la modalità <code>FindAncestor</code> . Nome di un tipo che riempie la proprietà AncestorType.</p>
<p><code>intLevel</code></p>	<p>Facoltativo per la modalità <code>FindAncestor</code> . Livello predecessore (valutato nella direzione padre dell'albero logico).</p>

Note

`{RelativeSource TemplatedParent}` gli utilizzi di binding sono una tecnica essenziale che risolve un concetto più ampio della separazione tra l'interfaccia utente di un controllo e la logica di un controllo. Ciò consente l'associazione dall'interno della definizione del modello al padre basato su modelli (istanza dell'oggetto in fase di esecuzione alla quale è applicato il modello). In questo caso, l' [estensione di markup TemplateBinding](#) è in realtà una sintassi abbreviata per la seguente espressione di associazione:

`{Binding RelativeSource={RelativeSource TemplatedParent}}` gli utilizzi di [TemplateBinding](#) o `{RelativeSource TemplatedParent}` sono entrambi rilevanti solo all'interno del codice XAML che definisce un modello. Per ulteriori informazioni, vedere [TemplateBinding Markup Extension](#).

`{RelativeSource FindAncestor}` viene utilizzata principalmente nei modelli di controllo o nelle composizioni dell'interfaccia utente autonomia prevedibile, nei casi in cui un controllo deve essere sempre incluso in una struttura ad albero visuale di un determinato tipo di predecessore. Ad esempio, gli elementi di un controllo di elementi potrebbero utilizzare `FindAncestor` per associarsi alle proprietà del relativo predecessore padre del controllo di elementi. In alternativa, gli elementi che fanno parte della composizione del controllo in un modello possono utilizzare le associazioni `FindAncestor` agli elementi padre nella stessa struttura della composizione.

Nella sintassi dell'elemento oggetto per la modalità `FindAncestor` illustrata nelle sezioni sulla sintassi XAML, la sintassi del secondo elemento oggetto viene utilizzata specificamente per la modalità `FindAncestor` . La modalità `FindAncestor` richiede un valore [AncestorType](#). È necessario impostare [AncestorType](#) come attributo utilizzando un riferimento all' [estensione di markup x:Type](#) al tipo di predecessore da cercare. Il valore [AncestorType](#) viene utilizzato quando la richiesta di associazione viene elaborata in fase di esecuzione.

Per la modalità `FindAncestor` , la proprietà facoltativa [AncestorLevel](#) consente di rendere meno ambigua la ricerca del predecessore laddove vi sono più predecessori di questo tipo nella struttura ad albero dell'elemento.

Per ulteriori informazioni su come utilizzare la modalità `FindAncestor` , vedere [RelativeSource](#).

`{RelativeSource Self}` è utile per gli scenari in cui una proprietà di un'istanza deve dipendere dal valore di un'altra proprietà della stessa istanza e non esiste già una relazione di proprietà di dipendenza generale, ad esempio la coercizione, tra queste due proprietà. Sebbene sia raro che esistano due proprietà su un oggetto in modo che i valori siano letteralmente identici (e siano tipizzati in modo identico), è anche possibile applicare un `Converter` parametro a un'associazione con `{RelativeSource Self}` e usare il convertitore per eseguire la conversione tra i tipi di origine e di destinazione. Un altro scenario per `{RelativeSource Self}` è come parte di un [MultiDataTrigger](#).

Ad esempio, nel codice XAML seguente viene definito un elemento `Rectangle` in modo che, indipendentemente dal valore inserito per `Width`, `Rectangle` sia sempre un quadrato:

```
<Rectangle Width="200" Height="{Binding RelativeSource={RelativeSource Self}, Path=Width}" .../>
```

`{RelativeSource PreviousData}` è utile nei modelli di dati o nei casi in cui le associazioni utilizzano una raccolta come origine dati. È possibile utilizzare `{RelativeSource PreviousData}` per evidenziare le relazioni tra gli elementi di dati adiacenti nella raccolta. Una tecnica correlata consiste nel porre un oggetto [MultiBinding](#) tra l'elemento corrente e quello precedente nell'origine dati e utilizzare un convertitore su quell'associazione per determinare la differenza tra i due elementi e le relative proprietà.

Nell'esempio riportato di seguito, il primo `TextBlock` in un modello di elementi visualizza il numero corrente. Il secondo binding `TextBlock` è un [MultiBinding](#) che ha nominalmente due componenti `Binding`: il record corrente e un'associazione che usa intenzionalmente il record di dati precedente tramite `{RelativeSource PreviousData}`. Quindi, un convertitore applicato a [MultiBinding](#) calcola la differenza e la restituisce all'associazione.

```
<ListBox Name="fibolist">
  <ListBox.ItemTemplate>
    <DataTemplate>
      <StackPanel Orientation="Horizontal">
        <TextBlock Text="{Binding}" />
        <TextBlock>, difference = </TextBlock>
        <TextBlock>
          <TextBlock.Text>
            <MultiBinding Converter="{StaticResource DiffConverter}">
              <Binding/>
              <Binding RelativeSource="{RelativeSource PreviousData}" />
            </MultiBinding>
          </TextBlock.Text>
        </TextBlock>
      </StackPanel>
    </DataTemplate>
  </ListBox.ItemTemplate>
```

La descrizione di data binding come concetto non è illustrata in questo articolo, vedere [Cenni preliminari sul data binding](#).

Nell'implementazione del processore XAML WPF la gestione di questa estensione di markup viene definita dalla classe `RelativeSource`.

`RelativeSource` è un'estensione di markup. Le estensioni di markup in genere vengono implementate quando per i valori dell'attributo devono essere utilizzati caratteri escape in modo che non vengano considerati come valori letterali o nomi di gestori e il requisito è più globale del semplice utilizzo di convertitori dei tipi su alcuni tipi o proprietà. Tutte le estensioni di markup in XAML usano i caratteri `{` e `}` nella sintassi dell'attributo, ovvero la convenzione con cui un processore XAML riconosce che un'estensione di markup deve elaborare l'attributo. Per altre informazioni, vedere [Estensioni di markup e XAML WPF](#).

Vedere anche

- [Binding](#)

- [Applicazione di stili e modelli](#)
- [Cenni preliminari su XAML \(WPF\)](#)
- [Estensioni di markup e XAML WPF](#)
- [Cenni preliminari sull'associazione dati](#)
- [Panoramica sulle dichiarazioni di associazione](#)
- [Estensione di markup x:Type](#)

Estensione del markup StaticResource

08/01/2020 • 5 minutes to read • [Edit Online](#)

Fornisce un valore per qualsiasi attributo di proprietà XAML cercando un riferimento a una risorsa già definita. Il comportamento di ricerca per tale risorsa è analogo alla ricerca in fase di caricamento, che cercherà le risorse che sono state caricate in precedenza dal markup della pagina XAML corrente, nonché altre origini applicazione, e genereranno il valore della risorsa come valore della proprietà negli oggetti Runtime.

Utilizzo della sintassi XAML per attributi

```
<object property="{StaticResource key}" .../>
```

Utilizzo della sintassi XAML per elementi oggetto

```
<object>
  <object.property>
<StaticResource ResourceKey="key" .../>
  </object.property>
</object>
```

Valor XAML

key

Chiave per la risorsa richiesta. Questa chiave è stata inizialmente assegnata dalla [direttiva x:Key](#) se una risorsa è stata creata nel markup oppure è stata fornita come parametro di `key` quando viene chiamata [ResourceDictionary.Add](#) se la risorsa è stata creata nel codice.

Note

IMPORTANT

Un `StaticResource` non deve tentare di creare un riferimento in avanti a una risorsa definita in modo lessicale all'interno del file di XAML. Il tentativo di eseguire questa operazione non è supportato e anche se un riferimento di questo tipo non ha esito negativo, il tentativo di eseguire il riferimento in diretta comporterà un calo delle prestazioni del tempo di caricamento quando vengono cercate le tabelle hash interne che rappresentano un [ResourceDictionary](#). Per ottenere risultati ottimali, modificare la composizione dei dizionari risorse in modo che i riferimenti in diretta possano essere evitati. Se non è possibile evitare un riferimento in diretta, usare invece l' [estensione di markup DynamicResource](#).

Il [ResourceKey](#) specificato deve corrispondere a una risorsa esistente, identificata con una [direttiva x:Key](#) a un certo livello nella pagina, applicazione, temi di controllo disponibili e risorse esterne o risorse di sistema. La ricerca di risorse viene eseguita nell'ordine specificato. Per altre informazioni sul comportamento di ricerca delle risorse per le risorse statiche e dinamiche, vedere [risorse XAML](#).

Una chiave di risorsa può essere qualsiasi stringa definita nella [Grammatica XamlName](#). Una chiave di risorsa può essere anche altri tipi di oggetto, ad esempio un [Type](#). Una chiave [Type](#) è fondamentale per il modo in cui i controlli possono essere disegnati in base ai temi, tramite una chiave di stile implicita. Per altre informazioni, vedere [Cenni preliminari sulla modifica di controlli](#).

Il mezzo dichiarativo alternativo per fare riferimento a una risorsa è come [estensione di markup DynamicResource](#).

La sintassi per gli attributi è quella più comunemente utilizzata con questa estensione di markup. Il token di stringa fornito dopo la stringa dell'identificatore `StaticResource` viene assegnato come valore [ResourceKey](#) della classe dell'estensione [StaticResourceExtension](#) sottostante.

`StaticResource` possibile utilizzare la sintassi dell'elemento oggetto. In questo caso, è necessario specificare il valore della proprietà [ResourceKey](#).

L'oggetto `StaticResource` può anche essere utilizzato per un utilizzo dettagliato degli attributi che consente di specificare la proprietà [ResourceKey](#) come coppia proprietà=valore:

```
<object property="{StaticResource ResourceKey=key}" .../>
```

L'utilizzo dettagliato spesso è utile per le estensioni con più proprietà da impostare o nel caso in cui alcune proprietà siano facoltative. Poiché `StaticResource` presenta una sola proprietà da impostare, obbligatoria, l'utilizzo dettagliato non è tipico.

Nell'implementazione del processore WPF XAML, la gestione di questa estensione di markup viene definita dalla classe [StaticResourceExtension](#).

`StaticResource` è un'estensione di markup. Le estensioni di markup in genere vengono implementate quando per i valori dell'attributo devono essere utilizzati caratteri escape in modo che non vengano considerati come valori letterali o nomi di gestori e il requisito è più globale del semplice utilizzo di convertitori dei tipi su alcuni tipi o proprietà. Tutte le estensioni di markup in XAML usano i caratteri { e } nella sintassi degli attributi, vale a dire la convenzione in base a cui il processore XAML riconosce che l'attributo deve essere elaborato da un'estensione di markup. Per altre informazioni, vedere [Estensioni di markup e XAML WPF](#).

Vedere anche

- [Applicazione di stili e modelli](#)
- [Cenni preliminari su XAML \(WPF\)](#)
- [Estensioni di markup e XAML WPF](#)
- [Risorse XAML](#)
- [Risorse e codice](#)

Estensione del markup TemplateBinding

25/11/2019 • 4 minutes to read • [Edit Online](#)

Collega il valore di una proprietà in un modello di controllo come valore di un'altra proprietà sul controllo basato su modelli.

Uso della sintassi XAML per gli attributi

```
<object property="{TemplateBinding sourceProperty}" .../>
```

Utilizzo dell'attributo XAML (per proprietà Set in modelli o stili)

```
<Setter Property="propertyName" Value="{TemplateBinding sourceProperty}" .../>
```

Valori XAML

propertyName	DependencyProperty.Name della proprietà impostata nella sintassi del setter.
sourceProperty	<p>Altra proprietà di dipendenza esistente nel tipo basato su modelli, specificata da DependencyProperty.Name.</p> <p>-oppure-</p> <p>Nome della proprietà puntato definito da un tipo diverso rispetto al tipo di destinazione basato su modelli. Si tratta in effetti di un oggetto PropertyPath. Vedere sintassi XAML di PropertyPath.</p>

Note

Una `TemplateBinding` è una forma ottimizzata di [associazione](#) per gli scenari di modello, analoga a una `Binding` costruita con `{Binding RelativeSource={RelativeSource TemplatedParent}, Mode=OneWay}`. `TemplateBinding` è sempre un'associazione unidirezionale, anche se le proprietà implicano come impostazione predefinita l'associazione bidirezionale. Entrambe le proprietà in questione devono essere proprietà di dipendenza. Per ottenere un binding bidirezionale a un elemento padre basato su modelli, usare invece la seguente istruzione di associazione

```
{Binding RelativeSource={RelativeSource TemplatedParent}, Mode=TwoWay, Path=MyDependencyProperty} .
```

`RelativeSource` è un'altra estensione di markup che viene talvolta utilizzata insieme a o al posto di `TemplateBinding` per eseguire un'associazione di proprietà relativa all'interno di un modello.

La descrizione di modelli di controllo come concetto non è descritta in questo articolo. per altre informazioni, vedere [stili e modelli di Control](#).

La sintassi per gli attributi è quella più comunemente utilizzata con questa estensione di markup. Il token di stringa fornito dopo la stringa dell'identificatore `TemplateBinding` viene assegnato come valore `Property` della

classe dell'estensione [TemplateBindingExtension](#) sottostante.

La sintassi di elementi oggetto è possibile, ma non viene mostrata poiché non ha applicazioni realistiche.

`TemplateBinding` viene utilizzato per inserire valori all'interno dei setter, tramite espressioni valutate e l'utilizzo della sintassi dell'elemento oggetto per `TemplateBinding` per riempire la sintassi dell'elemento di proprietà `<Setter.Property>` è inutilmente dettagliata.

L'oggetto `TemplateBinding` può anche essere utilizzato per un utilizzo dettagliato degli attributi che consente di specificare la proprietà [Property](#) come coppia proprietà=valore:

```
<object property="{TemplateBinding Property=sourceProperty}" .../>
```

L'utilizzo dettagliato spesso è utile per le estensioni con più proprietà da impostare o nel caso in cui alcune proprietà siano facoltative. Poiché `TemplateBinding` presenta una sola proprietà da impostare, obbligatoria, l'utilizzo dettagliato non è tipico.

Nell'implementazione del processore XAML WPF la gestione di questa estensione di markup viene definita dalla classe [TemplateBindingExtension](#).

`TemplateBinding` è un'estensione di markup. Le estensioni di markup in genere vengono implementate quando per i valori dell'attributo devono essere utilizzati caratteri escape in modo che non vengano considerati come valori letterali o nomi di gestori e il requisito è più globale del semplice utilizzo di convertitori dei tipi su alcuni tipi o proprietà. Tutte le estensioni di markup in XAML usano i caratteri `{` e `}` nella sintassi dell'attributo, ovvero la convenzione con cui un processore XAML riconosce che un'estensione di markup deve elaborare l'attributo. Per altre informazioni, vedere [Estensioni di markup e XAML WPF](#).

Vedere anche

- [Style](#)
- [ControlTemplate](#)
- [Applicazione di stili e modelli](#)
- [Cenni preliminari su XAML \(WPF\)](#)
- [Estensioni di markup e XAML WPF](#)
- [Estensione di markup RelativeSource](#)
- [Estensione di markup Binding](#)

Estensione di markup ThemeDictionary

04/11/2019 • 5 minutes to read • [Edit Online](#)

Offre agli autori di controlli personalizzati o alle applicazioni che integrano controlli di terze parti un modo per caricare dizionari di risorse specifici del tema da usare per l'applicazione di stili al controllo.

Uso della sintassi XAML per gli attributi

```
<object property="{ThemeDictionary assemblyUri}" .../>
```

Utilizzo della sintassi XAML per gli elementi oggetto

```
<object>
  <object.property>
    <ThemeDictionary AssemblyName="assemblyUri"/>
  <object.property>
<object>
```

Valori XAML

assemblyUri

URI (Uniform Resource Identifier) dell'assembly che contiene informazioni sul tema. In genere, si tratta di un URI di tipo pack che fa riferimento a un assembly del pacchetto più grande. Le risorse di assembly e gli URI di tipo pack semplificano i problemi di distribuzione. Per altre informazioni, vedere [URI di tipo pack in WPF](#).

Note

Questa estensione è destinata a compilare solo un valore di proprietà specifico, ovvero un valore per [ResourceDictionary.Source](#).

Utilizzando questa estensione, è possibile specificare un solo assembly di sole risorse contenente alcuni stili da utilizzare solo quando il tema Windows Aero viene applicato al sistema dell'utente, altri stili solo quando il tema Luna è attivo e così via. Con questa estensione, il contenuto di un dizionario risorse specifico del controllo può essere invalidato automaticamente e ricaricato in modo che venga considerato specifico per un altro tema quando necessario.

Il `assemblyUri` stringa (`AssemblyName` valore della proprietà) costituisce la base di una convenzione di denominazione che identifica il dizionario applicato per un particolare tema. La logica `ProvideValue` per `ThemeDictionary` completa la convenzione generando un URI (Uniform Resource Identifier) che punta a una particolare variante del dizionario dei temi, come contenuta in un assembly di risorse precompilato. La descrizione di questa convenzione o delle interazioni dei temi con l'applicazione generale degli stili al controllo e a livello di pagina o di applicazione come concetto, non è illustrata completamente in questa sezione. Lo scenario di base per l'utilizzo di `ThemeDictionary` consiste nel specificare la proprietà `Source` di un `ResourceDictionary` dichiarato a livello di applicazione. Quando si specifica un URI per l'assembly tramite un'estensione `ThemeDictionary` anziché come URI diretto, la logica di estensione fornirà la logica di invalidazione che viene applicata ogni volta che il tema

del sistema cambia.

La sintassi per gli attributi è quella più comunemente utilizzata con questa estensione di markup. Il token di stringa fornito dopo la stringa dell'identificatore `ThemeDictionary` viene assegnato come valore `AssemblyName` della classe dell'estensione `ThemeDictionaryExtension` sottostante.

`ThemeDictionary` può essere usato anche nella sintassi degli elementi oggetto. In questo caso, è necessario specificare il valore della proprietà `AssemblyName`.

L'oggetto `ThemeDictionary` può anche essere utilizzato per un utilizzo dettagliato degli attributi che consente di specificare la proprietà `Member` come coppia proprietà=valore:

```
<object property="{ThemeDictionary AssemblyName=assemblyUri}" .../>
```

L'utilizzo dettagliato spesso è utile per le estensioni con più proprietà da impostare o nel caso in cui alcune proprietà siano facoltative. Poiché `ThemeDictionary` presenta una sola proprietà da impostare, obbligatoria, l'utilizzo dettagliato non è tipico.

Nell'implementazione del processore WPF XAML, la gestione di questa estensione di markup viene definita dalla classe `ThemeDictionaryExtension`.

`ThemeDictionary` è un'estensione di markup. Le estensioni di markup in genere vengono implementate quando per i valori dell'attributo devono essere utilizzati caratteri escape in modo che non vengano considerati come valori letterali o nomi di gestori e il requisito è più globale del semplice utilizzo di convertitori dei tipi su alcuni tipi o proprietà. Tutte le estensioni di markup in XAML usano i caratteri { e } nella sintassi degli attributi, vale a dire la convenzione in base a cui il processore XAML riconosce che l'attributo deve essere elaborato da un'estensione di markup. Per altre informazioni, vedere [Estensioni di markup e XAML WPF](#).

Vedere anche

- [Applicazione di stili e modelli](#)
- [Cenni preliminari su XAML \(WPF\)](#)
- [Estensioni di markup e XAML WPF](#)
- [File di dati e di risorse dell'applicazione WPF](#)

Sintassi XAML di PropertyPath

08/01/2020 • 21 minutes to read • [Edit Online](#)

L'oggetto [PropertyPath](#) supporta una sintassi di XAML inline complessa per l'impostazione di varie proprietà che accettano il tipo di [PropertyPath](#) come valore. In questo argomento viene illustrata la sintassi [PropertyPath](#) applicata alle sintassi di associazione e animazione.

Uso di PropertyPath

[PropertyPath](#) è un oggetto comune usato in diverse funzionalità di Windows Presentation Foundation (WPF). Nonostante l'uso di [PropertyPath](#) comuni per fornire informazioni sul percorso delle proprietà, gli utilizzi per ogni area di funzionalità in cui [PropertyPath](#) viene usato come tipo variano. È quindi più pratico documentare le sintassi per ogni funzionalità.

Principalmente, WPF utilizza [PropertyPath](#) per descrivere i percorsi del modello a oggetti per attraversare le proprietà di un'origine dati dell'oggetto e per descrivere il percorso di destinazione per le animazioni di destinazione.

Alcune proprietà dello stile e del modello, ad esempio [Setter.Property](#) accettano un nome di proprietà completo che è simile a una [PropertyPath](#). Ma non si tratta di un vero [PropertyPath](#); è invece un proprietario qualificato . utilizzo del formato della stringa di proprietà abilitato dal processore WPF XAML in combinazione con il convertitore di tipi per [DependencyProperty](#).

PropertyPath per gli oggetti nel data binding

Il data binding è una funzionalità di WPF che consente di eseguire il binding al valore di destinazione di qualsiasi proprietà di dipendenza. L'origine di tale data binding non deve però essere necessariamente una proprietà di dipendenza, ma può essere qualsiasi tipo di proprietà riconosciuto dal provider di dati applicabile. I percorsi delle proprietà vengono utilizzati in particolare per la [ObjectDataProvider](#), che viene utilizzata per ottenere origini di associazione da oggetti Common Language Runtime (CLR) e dalle relative proprietà.

Si noti che data binding a XML non usa [PropertyPath](#), perché non usa [Path](#) nel [Binding](#). Usare invece [XPath](#) e specificare una sintassi XPath valida nei Document Object Model XML (DOM) dei dati. [XPath](#) viene anche specificato come stringa, ma non è documentato qui. vedere [eseguire l'associazione a dati XML tramite un oggetto XmlDataProvider e query XPath](#).

Per comprendere i percorsi delle proprietà nel data binding bisogna considerare che è possibile scegliere come destinazione del binding un singolo valore di proprietà oppure è possibile eseguire il binding a proprietà di destinazione che accettano elenchi o raccolte. Se si stanno associando raccolte, ad esempio l'associazione di un [ListBox](#) che si espanderà a seconda del numero di elementi di dati presenti nella raccolta, il percorso della proprietà deve fare riferimento all'oggetto raccolta, non a singoli elementi della raccolta. Il motore di data binding corrisponderà automaticamente alla raccolta usata come origine dati per il tipo di destinazione del binding, ottenendo un comportamento come il popolamento di una [ListBox](#) con una matrice di elementi.

Singola proprietà sull'oggetto immediato come contesto dati

```
<Binding Path="propertyName" .../>
```

PropertyName deve essere risolto in modo che sia il nome di una proprietà presente nell'[DataContext](#) corrente per un utilizzo [Path](#). Se il binding aggiorna l'origine, tale proprietà deve essere di lettura/scrittura e l'oggetto di origine deve essere modificabile.

Singolo indicizzatore sull'oggetto immediato come contesto dati

```
<Binding Path="[key]" .../>
```

`key` deve essere l'indice tipizzato per un dizionario o una tabella hash oppure l'indice Integer di una matrice. Il valore della chiave deve inoltre essere un tipo direttamente associabile alla proprietà a cui è applicato. Ad esempio, è possibile usare una tabella hash contenente chiavi di stringa e valori di stringa in questo modo per associare il testo a un [TextBox](#). In alternativa, se la chiave punta a una raccolta o a un indice secondario, è possibile usare questa sintassi per il binding a una proprietà della raccolta di destinazione. In caso contrario, è necessario fare riferimento a una proprietà specifica, tramite una sintassi come

```
<Binding Path="[key].propertyName" .../> .
```

È possibile specificare il tipo dell'indice, se necessario. Per informazioni dettagliate su questo aspetto di un percorso di proprietà indicizzato, vedere [Binding.Path](#).

Più proprietà (impostazione indiretta delle proprietà come destinazioni)

```
<Binding Path="propertyName.propertyName2" .../>
```

`propertyName` deve essere risolto in modo che sia il nome di una proprietà che rappresenta l'[DataContext](#) corrente. Le proprietà del percorso `propertyName` e `propertyName2` possono essere costituite da qualsiasi proprietà presente in una relazione, dove `propertyName2` è una proprietà presente nel tipo che corrisponde al valore di `propertyName`.

Singola proprietà, associata o qualificata in altro modo dal tipo

```
<object property="(ownerType.propertyName)" .../>
```

Le parentesi indicano che questa proprietà in un [PropertyPath](#) deve essere costruita utilizzando una qualifica parziale. Può usare uno spazio dei nomi XML per trovare il tipo con un mapping appropriato. Il `ownerType` Cerca i tipi a cui un processore di XAML ha accesso, tramite le dichiarazioni di [XmlAttributeDefinition](#) in ogni assembly. La maggior parte delle applicazioni ha lo spazio dei nomi XML predefinito mappato allo spazio dei nomi `http://schemas.microsoft.com/winfx/2006/xaml/presentation`, quindi in genere è necessario un prefisso solo per i tipi personalizzati o per i tipi altrimenti esterni allo spazio dei nomi. `propertyName` deve risolversi nel nome di una proprietà presente in `ownerType`. Questa sintassi viene in genere usata per uno dei casi seguenti:

- Il percorso è specificato in XAML, ovvero in uno stile o modello che non ha un tipo di destinazione specificato. Un utilizzo qualificato non è in genere valido in casi diversi da questo, perché in casi senza uno stile o un modello la proprietà è presente in un'istanza, non in un tipo.
- La proprietà è una proprietà associata.
- Si sta eseguendo il binding a una proprietà statica.

Per l'utilizzo come destinazione dello storyboard, la proprietà specificata come `propertyName` deve essere una [DependencyProperty](#).

Attraversamento dell'origine (binding a gerarchie di raccolte)

```
<object Path="propertyName/propertyNameX" .../>
```

Il carattere / in questa sintassi viene usato per spostarsi in un oggetto origine dati gerarchico e sono supportati più passaggi nella gerarchia con caratteri / successivi. L'attraversamento dell'origine tiene conto della posizione del puntatore di record corrente, determinata sincronizzando i dati con l'interfaccia utente della relativa

visualizzazione. Per informazioni dettagliate sul binding con oggetti origine dati gerarchici e sul concetto di puntatore di record corrente nel data binding, vedere [Usare il modello Master-Details con dati gerarchici](#) o [Panoramica sul data binding](#).

NOTE

In apparenza, questa sintassi è simile a XPath. Una vera espressione XPath per l'associazione a un'origine dati XML non viene utilizzata come valore di `Path` e deve invece essere utilizzata per la proprietà `XPath` a esclusione reciproca.

Visualizzazioni di raccolte

Per fare riferimento a una visualizzazione di raccolta denominata, premettere il carattere hash (#) al nome della visualizzazione di raccolta.

Puntatore al record corrente

Per fare riferimento al puntatore al record corrente per una visualizzazione di raccolta o uno scenario di data binding master-dettagli, iniziare la stringa di percorso con una barra (/). Qualsiasi percorso dopo la barra viene attraversato a partire dal puntatore al record corrente.

Indicizzatori multipli

```
<object Path="[index1,index2...]" .../>
```

oppure

```
<object Path="propertyName[index,index2...]" .../>
```

Se un determinato oggetto supporta più indicizzatori, è possibile specificare tali indicizzatori in ordine, analogamente a una sintassi che fa riferimento a una matrice. L'oggetto in questione può essere il contesto corrente o il valore di una proprietà contenente un oggetto a più indici.

Per impostazione predefinita, i valori dell'indicizzatore sono tipizzati usando le caratteristiche dell'oggetto sottostante. È possibile specificare il tipo dell'indice, se necessario. Per informazioni dettagliate sulla digitazione degli indicizzatori, vedere [Binding.Path](#).

Sintassi miste

Tutte le sintassi illustrate in precedenza possono essere combinate. Ad esempio, di seguito è riportato un esempio in cui viene creato un percorso di proprietà al colore in corrispondenza di una particolare x, y di una `ColorGrid` proprietà contenente una matrice di griglia in pixel di oggetti `SolidColorBrush`:

```
<Rectangle Fill="{Binding ColorGrid[20,30].SolidColorBrushResult}" .../>
```

Caratteri di escape per le stringhe di percorso delle proprietà

Per determinati oggetti business è possibile che la stringa di percorso delle proprietà richieda un carattere di escape per poter essere analizzata correttamente. L'esigenza di usare caratteri di escape deve essere rara, perché molti di questi caratteri hanno problemi di interazione-denominazione simili nei linguaggi che in genere vengono usati per definire l'oggetto business.

- All'interno degli indicizzatori ([]), l'accento circonflesso (^) funge da escape per il carattere successivo.
- È necessario usare caratteri di escape (con entità XML) per alcuni caratteri specifici della definizione del linguaggio XML. Usare & come carattere di escape per "&". Usare > come carattere di escape per il tag di fine ">".

- È necessario usare un carattere di escape (la barra rovesciata `\`) per i caratteri specifici del comportamento del parser XAML di WPF per l'elaborazione di un'estensione di markup.
 - La stessa barra rovesciata (`\`) costituisce il carattere di escape.
 - Il segno di uguale (`=`) separa il nome della proprietà dal valore della proprietà.
 - La virgola (`,`) separa le proprietà.
 - La parentesi graffa chiusa (`}`) rappresenta la fine di un'estensione di markup.

NOTE

Tecnicamente, questi caratteri di escape funzionano anche per un percorso di proprietà dello storyboard, ma in genere si attraversano modelli a oggetti per oggetti WPF esistenti e l'uso di caratteri di escape non è necessario.

PropertyPath per le destinazioni delle animazioni

La proprietà di destinazione di un'animazione deve essere una proprietà di dipendenza che accetta un [Freezable](#) o un tipo primitivo. La proprietà di destinazione su un tipo e la proprietà animata finale possono tuttavia trovarsi su oggetti diversi. Per le animazioni, un percorso delle proprietà viene usato per definire la connessione tra la proprietà dell'oggetto di destinazione dell'animazione denominata e la proprietà dell'animazione di destinazione prevista, attraversando le relazioni oggetto-proprietà nei valori di proprietà.

Considerazioni generali sulle relazioni oggetto-proprietà per le animazioni

Per altre informazioni sui concetti di animazione in generale, vedere [Cenni preliminari sugli storyboard](#) e [Cenni preliminari sull'animazione](#).

Il tipo di valore o la proprietà a cui si sta aggiungendo un'animazione deve essere un tipo [Freezable](#) o una primitiva. La proprietà che avvia il percorso deve essere risolta come il nome di una proprietà di dipendenza esistente nel tipo di [TargetName](#) specificato.

Per supportare la clonazione per l'animazione di una [Freezable](#) già bloccata, l'oggetto specificato da [TargetName](#) deve essere un [FrameworkElement](#) o [FrameworkContentElement](#) classe derivata.

Singola proprietà nell'oggetto di destinazione

```
<animation Storyboard.TargetProperty="propertyName" .../>
```

`propertyName` deve essere risolto come il nome di una proprietà di dipendenza esistente nel tipo di [TargetName](#) specificato.

Impostazione indiretta delle proprietà come destinazioni

```
<animation Storyboard.TargetProperty="propertyName.propertyName2" .../>
```

`propertyName` deve essere una proprietà che è un tipo di valore [Freezable](#) o una primitiva, presente nel tipo di [TargetName](#) specificato.

`propertyName2` deve essere il nome di una proprietà di dipendenza presente nell'oggetto che costituisce il valore di `propertyName`. In altre parole, `propertyName2` deve esistere come proprietà di dipendenza nel tipo che corrisponde al `propertyName` [.PropertyType](#).

L'impostazione indiretta delle animazioni come destinazioni è necessaria a causa degli stili e dei modelli applicati. Per fare riferimento a un'animazione, è necessario un [TargetName](#) in un oggetto di destinazione e tale

nome viene stabilito da `x:Name` o `Name`. Sebbene gli elementi di modelli e stili possano avere nomi, tali nomi sono validi solo all'interno dell'ambito dei nomi dello stile e del modello. Se modelli e stili non condividessero gli ambiti dei nomi con il markup dell'applicazione, i nomi non potrebbero essere univoci. Gli stili e i modelli sono letteralmente condivisi tra le istanze e possono perpetuare i nomi duplicati. Pertanto, se le singole proprietà di un elemento che si desidera animare provengono da uno stile o da un modello, è necessario iniziare con un'istanza dell'elemento denominato che non deriva da un modello di stile e quindi eseguire la destinazione nella struttura ad albero visuale di stile o modello per arrivare alla proprietà si desidera aggiungere un'animazione a.

Ad esempio, la proprietà `Background` di un `Panel` è un `Brush` completo (effettivamente un `SolidColorBrush`) proveniente da un modello di tema. Per animare un `Brush` completamente, è necessario che sia presente un `BrushAnimation` (probabilmente uno per ogni tipo di `Brush`) e che tale tipo non sia presente. Per animare un pennello, è invece necessario animare le proprietà di un particolare tipo di `Brush`. È necessario ottenere da `SolidColorBrush` al `Color` per applicare un `ColorAnimation`. Il percorso della proprietà per questo esempio sarebbe `Background.Color`.

Proprietà associate

```
<animation Storyboard.TargetProperty="(ownerType.propertyName)" .../>
```

Le parentesi indicano che questa proprietà in un `PropertyPath` deve essere costruita utilizzando una qualifica parziale. Può usare uno spazio dei nomi XML per trovare il tipo. Il `ownerType` Cerca i tipi a cui un processore di XAML ha accesso, tramite le dichiarazioni di `XmlnsDefinitionAttribute` in ogni assembly. La maggior parte delle applicazioni ha lo spazio dei nomi XML predefinito mappato allo spazio dei nomi

<http://schemas.microsoft.com/winfx/2006/xaml/presentation>, quindi in genere è necessario un prefisso solo per i tipi personalizzati o per i tipi altrimenti esterni allo spazio dei nomi. `propertyName` deve risolversi nel nome di una proprietà presente in `ownerType`. La proprietà specificata come `propertyName` deve essere una `DependencyProperty`. Tutte le proprietà associate di WPF sono implementate come proprietà di dipendenza, quindi questo problema riguarda solo le proprietà associate personalizzate.

Indexers (Indicizzatori)

```
<animation Storyboard.TargetProperty="propertyName.propertyName2[index].propertyName3" .../>
```

La maggior parte delle proprietà di dipendenza o dei tipi di `Freezable` non supporta un indicizzatore. L'unico utilizzo di un indicizzatore in un percorso di animazione è quindi in una posizione intermedia tra la proprietà che inizia la catena nella destinazione denominata e la proprietà animata finale. Nella sintassi fornita si tratta di `propertyName2`. Ad esempio, potrebbe essere necessario un utilizzo dell'indicizzatore se la proprietà intermedia è una raccolta, ad esempio `TransformGroup`, in un percorso di proprietà, ad esempio `RenderTransform.Children[1].Angle`.

PropertyPath nel codice

L'utilizzo del codice per `PropertyPath`, inclusa la modalità di creazione di un `PropertyPath`, è documentato nell'argomento di riferimento per `PropertyPath`.

In generale, `PropertyPath` è progettato per usare due costruttori diversi, uno per gli utilizzi di binding e gli utilizzi di animazione più semplici e uno per gli utilizzi di animazione complessi. Usare la firma `PropertyPath(Object)` per gli utilizzi di binding, in cui l'oggetto è una stringa. Usare la firma `PropertyPath(Object)` per i percorsi di animazione in un unico passaggio, in cui l'oggetto è un `DependencyProperty`. Usare la firma `PropertyPath(String, Object[])` per le animazioni complesse. Quest'ultimo costruttore usa una stringa token per il primo parametro e una matrice di oggetti che riempiono le posizioni nella stringa token per definire una relazione di percorso di proprietà.

Vedere anche

- [PropertyPath](#)
- [Cenni preliminari sull'associazione dati](#)
- [Cenni preliminari sugli storyboard](#)

Attributo PresentationOptions:Freeze

23/10/2019 • 2 minutes to read • [Edit Online](#)

Imposta lo `IsFrozen` `true` stato su sull'elemento contenitore `Freezable`. Il comportamento predefinito per `Freezable` un oggetto `PresentationOptions:Freeze` senza l'attributo specificato `IsFrozen` è `false` che è in fase di caricamento e dipende `Freezable` dal comportamento generale in fase di esecuzione.

Uso della sintassi XAML per gli attributi

```
<object
    xmlns:PresentationOptions="http://schemas.microsoft.com/winfx/2006/xaml/presentation/options"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="PresentationOptions">
    <freezableElement PresentationOptions:Freeze="true"/>
</object>
```

Valori XAML

<code>PresentationOptions</code>	Prefisso dello spazio dei nomi XML, che può essere qualsiasi stringa di prefisso valida, in base alla specifica XML 1.0. Il prefisso <code>PresentationOptions</code> viene usato per scopi di identificazione in questa documentazione.
<code>freezableElement</code>	Elemento che crea un'istanza di qualsiasi classe derivata <code>Freezable</code> di.

Note

L' `Freeze` attributo è l'unico attributo o altro elemento

`http://schemas.microsoft.com/winfx/2006/xaml/presentation/options` di programmazione definito nello spazio dei nomi XML. L' `Freeze` attributo esiste in questo particolare spazio dei nomi specifico, in modo che possa essere designato come ignorabile, usando l' [attributo MC: Ignorable](#) come parte delle dichiarazioni degli elementi radice. Il motivo per `Freeze` cui deve essere possibile ignorare è il fatto che non tutte XAML le implementazioni del processore sono in grado `Freezable` di bloccare un oggetto in fase di caricamento. questa funzionalità XAML non fa parte della specifica.

La possibilità di elaborare l' `Freeze` attributo è incorporata in modo specifico XAML nel processore che XAML elabora le applicazioni compilate. L'attributo non è supportato da alcuna classe e la sintassi dell'attributo non è estendibile o modificabile. Se si implementa un processore personalizzato XAML , è possibile scegliere di eseguire il blocco del comportamento WPF XAML del processore durante l'elaborazione `Freeze` dell'attributo `Freezable` sugli elementi in fase di caricamento.

Qualsiasi valore per l' `Freeze` attributo diverso da `true` (senza distinzione tra maiuscole e minuscole) genera un errore in fase di caricamento. (Specificando `Freeze` l'attributo `false` come non è un errore, ma è già l'impostazione predefinita, pertanto l' `false` impostazione di su non esegue alcuna operazione).

Vedere anche

- [Freezable](#)
- [Cenni preliminari sugli oggetti Freezable](#)
- [Attributo mc:Ignorable](#)

Compatibilità dei markup (mc:) Funzionalità del linguaggio

23/10/2019 • 2 minutes to read • [Edit Online](#)

In questa sezione

[Attributo mc:Ignorable](#)

[Attributo mc:ProcessContent](#)

Attributo mc:Ignorable

08/11/2019 • 4 minutes to read • [Edit Online](#)

Specifica quali prefissi dello spazio dei nomi XML rilevati in un file di markup possono essere ignorati da un processore XAML. L'attributo `mc:Ignorable` supporta la compatibilità del markup sia per il mapping dello spazio dei nomi personalizzato che per il controllo delle versioni XAML.

Utilizzo degli attributi XAML (prefisso singolo)

```
<object
    xmlns:ignorablePrefix="ignorableUri"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="ignorablePrefix"...
        <ignorablePrefix1:ThisElementCanBeIgnored/>
</object>
```

Utilizzo degli attributi XAML (due prefissi)

```
<object
    xmlns:ignorablePrefix1="ignorableUri"
    xmlns:ignorablePrefix2="ignorableUri2"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="ignorablePrefix1 ignorablePrefix2"...
        <ignorablePrefix1:ThisElementCanBeIgnored/>
</object>
```

Valori XAML

<i>ignorablePrefix, ignorablePrefix1 e così via.</i>	qualsiasi stringa di prefisso valida, in base alla specifica XML 1,0.
<i>ignorableUri</i>	qualsiasi URI valido per la designazione di uno spazio dei nomi, in base alla specifica XML 1,0.
<i>ThisElementCanBeIgnored</i>	Elemento che può essere ignorato dalle implementazioni del processore Extensible Application Markup Language (XAML), se non è possibile risolvere il tipo sottostante.

Note

Il prefisso dello spazio dei nomi XML `mc` è la convenzione di prefisso consigliata da usare per il mapping dello spazio dei nomi di compatibilità XAML <http://schemas.openxmlformats.org/markup-compatibility/2006>.

Gli elementi o gli attributi in cui la parte prefisso del nome dell'elemento sono identificati come `mc:Ignorable` non genereranno errori quando vengono elaborati da un processore di XAML. Se tale attributo non può essere risolto in un tipo o un costrutto di programmazione sottostante, l'elemento viene ignorato. Si noti tuttavia che gli elementi ignorati possono comunque generare errori di analisi aggiuntivi per ulteriori requisiti degli elementi che sono effetti collaterali dell'elemento non elaborato. Ad esempio, un particolare modello di contenuto

dell'elemento potrebbe richiedere esattamente un elemento figlio, ma se l'elemento figlio specificato si trovava in un prefisso `mc:Ignorable` e non è stato possibile risolvere l'elemento figlio specificato in un tipo, il processore XAML potrebbe generare un errore.

`mc:Ignorable` si applica solo ai mapping dello spazio dei nomi alle stringhe identificatore. `mc:Ignorable` non si applica ai mapping dello spazio dei nomi negli assembly, che specificano uno spazio dei nomi CLR e un assembly (oppure il valore predefinito per l'eseguibile corrente come assembly).

Se si implementa un processore di XAML, l'implementazione del processore non deve generare errori di analisi o elaborazione sulla risoluzione del tipo per qualsiasi elemento o attributo qualificato da un prefisso identificato come `mc:Ignorable`. Tuttavia, l'implementazione del processore può comunque generare eccezioni che sono un risultato secondario di un errore di caricamento o elaborazione di un elemento, ad esempio un elemento figlio di esempio specificato in precedenza.

Per impostazione predefinita, un processore XAML ignorerà il contenuto all'interno di un elemento ignorato. Tuttavia, è possibile specificare un attributo aggiuntivo, [MC: ProcessContent](#), per richiedere l'elaborazione continua del contenuto all'interno di un elemento ignorato dal successivo elemento padre disponibile.

È possibile specificare più prefissi nell'attributo, usando uno o più caratteri di spazio vuoto come separatore, ad esempio: `mc:Ignorable="ignore1 ignore2"`.

Lo spazio dei nomi <http://schemas.openxmlformats.org/markup-compatibility/2006> definisce altri elementi e attributi non documentati in questa area dell'SDK. Per ulteriori informazioni, vedere [XML Markup Compatibility Specification](#).

Vedere anche

- [XamlReader](#)
- [Attributo PresentationOptions:Freeze](#)
- [Cenni preliminari su XAML \(WPF\)](#)
- [Documenti in WPF](#)

Attributo mc:ProcessContent

10/02/2020 • 2 minutes to read • [Edit Online](#)

Specifica quali elementi di XAML devono ancora avere contenuto elaborato da elementi padre pertinenti, anche se l'elemento padre diretto può essere ignorato da un processore di XAML a causa della specifica dell'[attributo MC:Ignorable](#). L'attributo `mc:ProcessContent` supporta la compatibilità del markup sia per il mapping dello spazio dei nomi personalizzato che per il controllo delle versioni XAML.

Utilizzo della sintassi XAML per attributi

```
<object
    xmlns:ignorablePrefix="ignorableUri"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="ignorablePrefix"...
    mc:ProcessContent="ignorablePrefix:ThisElementCanBeIgnored"
>
    <ignorablePrefix:ThisElementCanBeIgnored>
        [content]
    </ignorablePrefix:ThisElementCanBeIgnored>
</object>
```

Valori XAML

<code>ignorablePrefix</code>	qualsiasi stringa di prefisso valida, in base alla specifica XML 1,0.
<code>ignorableUri</code>	qualsiasi URI valido per la designazione di uno spazio dei nomi, in base alla specifica XML 1,0.
<code>ThisElementCanBeIgnored</code>	Elemento che può essere ignorato dalle implementazioni del processore Extensible Application Markup Language (XAML), se non è possibile risolvere il tipo sottostante.
<code>contenuto</code>	<code>ThisElementCanBeIgnored</code> è contrassegnato come ignorabile. Se il processore ignora tale elemento, <code>[content]</code> viene elaborato dall' <i>oggetto</i> .

Osservazioni

Per impostazione predefinita, un processore XAML ignorerà il contenuto all'interno di un elemento ignorato. È possibile specificare un elemento specifico per `mc:ProcessContent` e un processore di XAML continuerà a elaborare il contenuto all'interno dell'elemento ignorato. Questa operazione viene in genere usata se il contenuto è annidato all'interno di diversi tag, almeno uno dei quali è ignorabile e almeno uno di questi non è ignorabile.

È possibile specificare più prefissi nell'attributo, usando un separatore di spazio, ad esempio:

```
mc:ProcessContent="ignore:Element1 ignore:Element2".
```

Lo spazio dei nomi `http://schemas.openxmlformats.org/markup-compatibility/2006` definisce altri elementi e attributi non documentati in questa area dell'SDK. Per ulteriori informazioni, vedere [XML Markup Compatibility Specification](#).

Vedere anche

- [Attributo mc:Ignorable](#)
- [Panoramica di XAML \(WPF\)](#)

Elementi di base

04/11/2019 • 2 minutes to read • [Edit Online](#)

Quattro classi chiave,[UIElement](#), [ContentElement](#), [FrameworkElement](#) e [FrameworkContentElement](#), implementano una percentuale sostanziale della funzionalità degli elementi comuni disponibile nella programmazione WPF. Queste quattro classi sono denominate in questo SDK come classi di elementi di base.

Contenuto della sezione

[Cenni preliminari sugli elementi di base](#)

[Cenni preliminari sugli oggetti Freezable](#)

[Panoramica su allineamento, margini e spaziatura interna](#)

[Procedure relative alle proprietà](#)

Reference

[UIElement](#)

[ContentElement](#)

[FrameworkElement](#)

[FrameworkContentElement](#)

Sezioni correlate

[Architettura WPF](#)

[XAML in WPF](#)

[Albero degli elementi e serializzazione](#)

[Proprietà](#)

[Eventi](#)

[Input](#)

[Risorse](#)

[Applicazione di stili e modelli](#)

[Modello di threading](#)

Cenni preliminari sugli elementi di base

08/11/2019 • 14 minutes to read • [Edit Online](#)

Una percentuale elevata di classi in Windows Presentation Foundation (WPF) deriva da quattro classi, comunemente definite nella documentazione dell'SDK come classi di elementi di base. Queste classi sono [UIElement](#), [FrameworkElement](#), [ContentElement](#) e [FrameworkContentElement](#). Anche la classe [DependencyObject](#) è correlata, perché è una classe di base comune sia [UIElement](#) che [ContentElement](#).

API degli elementi di base nelle classi WPF

Sia [UIElement](#) che [ContentElement](#) derivano da [DependencyObject](#), tramite percorsi leggermente diversi. La suddivisione a questo livello riguarda il modo in cui un [UIElement](#) o un [ContentElement](#) vengono usati in un'interfaccia utente e lo scopo che svolgono in un'applicazione. [UIElement](#) dispone anche di [Visual](#) nella relativa gerarchia di classi, ovvero una classe che espone il supporto della grafica di livello inferiore sottostante al Windows Presentation Foundation (WPF). [Visual](#) fornisce un Framework di rendering definendo aree della schermata rettangolari indipendenti. In pratica, [UIElement](#) è per gli elementi che supporteranno un modello a oggetti più grande, sono progettati per il rendering e il layout in aree che possono essere descritte come aree della schermata rettangolare e in cui il modello di contenuto è intenzionalmente più aperto, per consentire combinazioni diverse di elementi. [ContentElement](#) non deriva da [Visual](#); il modello è che un [ContentElement](#) verrebbe utilizzato da un altro elemento, ad esempio un Reader o un visualizzatore che interpreterà quindi gli elementi e produrrà la [Visual](#) completa per l'utilizzo da parte di Windows Presentation Foundation (WPF). Alcune classi di [UIElement](#) sono destinate a essere host di contenuto: forniscono l'hosting e il rendering per una o più classi [ContentElement](#) ([DocumentViewer](#) è un esempio di tale classe). [ContentElement](#) viene utilizzata come classe di base per gli elementi con modelli a oggetti più piccoli e che più indirizzano il contenuto del testo, delle informazioni o del documento che potrebbe essere ospitato all'interno di un [UIElement](#).

Livello di framework e livello principale

[UIElement](#) funge da classe base per [FrameworkElement](#) e [ContentElement](#) funge da classe di base per [FrameworkContentElement](#). Il motivo di questo livello di classi successivo è quello di supportare un livello di core WPF separato da un livello di Framework WPF, con questa divisione anche nel modo in cui le API sono divise tra gli assembly [PresentationCore](#) e [PresentationFramework](#). Il livello di framework WPF offre una soluzione più completa per le necessità di base delle applicazioni, inclusa l'implementazione del gestore di layout per le presentazioni. Il livello principale WPF offre una modalità per usare in modo esteso WPF senza l'overhead di assembly aggiuntivo. La distinzione tra questi livelli è molto raramente importante per gli scenari di sviluppo di applicazioni più comuni e in generale è opportuno considerare le API WPF nel suo complesso e non preoccuparsi della differenza tra il livello di Framework WPF e il livello di core WPF. Può essere necessario conoscere le distinzioni tra i livelli se in un progetto di applicazione si decide di sostituire quantità significative di funzionalità del livello di framework WPF, ad esempio se la soluzione complessiva dispone già di implementazioni personalizzate per la composizione e il layout dell'interfaccia utente.

Scelta dell'elemento da cui derivare

Il modo ottimale per creare una classe personalizzata che estende WPF consiste nel derivare da una delle classi WPF mediante cui si ottiene la massima funzionalità desiderata tramite la gerarchia di classi esistente. Questa sezione elenca le funzionalità fornite da tre delle più importanti classi di elementi che consentono di decidere da quale classe ereditare.

Se si implementa un controllo, che è in realtà uno dei motivi più comuni per derivare da una classe di WPF, è probabile che si desideri derivare da una classe che sia un controllo pratico, una classe di base della famiglia di

controlli o almeno dalla classe di base [Control](#). Per istruzioni ed esempi pratici, vedere [Cenni preliminari sulla modifica di controlli](#).

Se non si crea un controllo ed è necessario derivare da una classe più elevata nella gerarchia, nelle sezioni seguenti vengono fornite informazioni sulle caratteristiche definite in ogni classe degli elementi di base.

Se si crea una classe che deriva da [DependencyObject](#), si ereditano le funzionalità seguenti:

- supporto di [GetValue](#) e [SetValue](#) e supporto del sistema di proprietà generale.
- Possibilità di utilizzare le proprietà di dipendenza e le proprietà associate implementate come proprietà di dipendenza.

Se si crea una classe che deriva da [UIElement](#), si ereditano le funzionalità seguenti oltre a quelle fornite da [DependencyObject](#):

- Supporto di base per i valori delle proprietà animate. Per altre informazioni, vedere [Panoramica dell'animazione](#).
- Supporto di eventi di input di base e supporto dei comandi. Per altre informazioni, vedere [Cenni preliminari sull'input](#) e [Cenni preliminari sull'esecuzione di comandi](#).
- Metodi virtuali di cui è possibile eseguire l'override per fornire informazioni a un sistema di layout.

Se si crea una classe che deriva da [FrameworkElement](#), si ereditano le funzionalità seguenti oltre a quelle fornite da [UIElement](#):

- Supporto per l'uso di stili e storyboard. Per ulteriori informazioni, vedere [Cenni preliminari sugli storyboard Style](#).
- Supporto del data binding. Per altre informazioni, vedere la [panoramica del data binding](#).
- Supporto per i riferimenti di risorse dinamiche. Per altre informazioni, vedere [Risorse XAML](#).
- Supporto dell'ereditarietà dei valori di proprietà e altri flag nei metadati che agevolano la segnalazione di condizioni relative alle proprietà ai servizi del framework, ad esempio data binding, stili o implementazione del layout del framework. Per altre informazioni, vedere [Metadati delle proprietà del framework](#).
- Concetto di albero logico. Per altre informazioni, vedere [Strutture ad albero in WPF](#).
- Supporto per l'implementazione pratica a livello di Framework WPF del sistema di layout, incluso un override [OnPropertyChanged](#) in grado di rilevare le modifiche apportate alle proprietà che influenzano il layout.

Se si crea una classe che deriva da [ContentElement](#), si ereditano le funzionalità seguenti oltre a quelle fornite da [DependencyObject](#):

- Supporto per le animazioni. Per altre informazioni, vedere [Panoramica dell'animazione](#).
- Supporto di eventi di input di base e supporto dei comandi. Per altre informazioni, vedere [Cenni preliminari sull'input](#) e [Cenni preliminari sull'esecuzione di comandi](#).

Se si crea una classe che deriva da [FrameworkContentElement](#), si ottengono le funzionalità seguenti, oltre a quelle fornite da [ContentElement](#):

- Supporto per l'uso di stili e storyboard. Per ulteriori informazioni, vedere [Style](#) e [Cenni preliminari sull'animazione](#).
- Supporto del data binding. Per altre informazioni, vedere la [panoramica del data binding](#).

- Supporto per i riferimenti di risorse dinamiche. Per altre informazioni, vedere [Risorse XAML](#).
- Supporto dell'ereditarietà dei valori di proprietà e altri flag nei metadati che agevolano la segnalazione di condizioni relative alle proprietà ai servizi del framework, ad esempio data binding, stili o implementazione del layout del framework. Per altre informazioni, vedere [Metadati delle proprietà del framework](#).
- L'accesso alle modifiche apportate al sistema di layout, ad esempio [ArrangeOverride](#), non viene ereditato. Le implementazioni del sistema di layout sono disponibili solo in [FrameworkElement](#). Tuttavia, è possibile ereditare un override [OnPropertyChanged](#) in grado di rilevare le modifiche apportate alle proprietà che influenzano il layout e segnalarle a tutti gli host di contenuto.

I modelli di contenuto sono documentati per una varietà di classi. Il modello di contenuto per una classe è un fattore possibile da considerare per trovare una classe adatta da cui derivare. Per altre informazioni, vedere [Modello di contenuto WPF](#).

Altre classi di base

DispatcherObject

[DispatcherObject](#) fornisce il supporto per il modello di threading WPF e consente a tutti gli oggetti creati per WPF applicazioni di essere associati a una [Dispatcher](#). Anche se non si esegue la derivazione da [UIElement](#), [DependencyObject](#) [Visual](#), è necessario considerare la derivazione da [DispatcherObject](#) per ottenere il supporto del modello di Threading. Per altre informazioni, vedere [Modello di threading](#).

Visual

[Visual](#) implementa il concetto di oggetto 2D che in genere richiede la presentazione visiva in un'area approssimativamente rettangolare. Il rendering effettivo di un [Visual](#) si verifica in altre classi (non è indipendente), ma la classe [Visual](#) fornisce un tipo noto che viene usato dai processi di rendering a vari livelli. [Visual](#) implementa l'hit testing, ma non espone eventi che segnalano positivi di hit testing (si trovano in [UIElement](#)). Per altre informazioni, vedere [Programmazione a livello visivo](#).

Freezable

[Freezable](#) simula l'immutabilità in un oggetto modificabile fornendo i mezzi per generare copie dell'oggetto quando un oggetto non modificabile è obbligatorio o desiderato per motivi di prestazioni. Il tipo di [Freezable](#) fornisce una base comune per determinati elementi grafici, ad esempio le geometrie e i pennelli, nonché le animazioni. In particolare, un [Freezable](#) non è un [Visual](#); può ospitare proprietà che diventano sottoproprietà quando l'[Freezable](#) viene applicato per riempire un valore di proprietà di un altro oggetto e tali proprietà possono influire sul rendering. Per altre informazioni, vedere [Cenni preliminari sugli oggetti Freezable](#).

Animatable

[Animatable](#) è una classe [Freezable](#) derivata che aggiunge in modo specifico il livello di controllo dell'animazione e alcuni membri dell'utilità in modo che le proprietà attualmente animate possano essere distinte dalle proprietà non animate.

Control

[Control](#) è la classe base prevista per il tipo di oggetto che viene definito da un controllo o un componente, a seconda della tecnologia. In genere, le classi di controlli WPF sono classi che rappresentano direttamente un controllo dell'interfaccia utente o partecipano alla composizione di tale controllo. La funzionalità principale che [Control](#) Abilita è il modello di controllo.

Vedere anche

- [Control](#)
- [Panoramica sulle proprietà di dipendenza](#)

- Cenni preliminari sulla modifica di controlli
- Architettura WPF

Cenni preliminari sugli oggetti Freezable

10/02/2020 • 16 minutes to read • [Edit Online](#)

In questo argomento viene descritto come utilizzare e creare in modo efficace oggetti **Freezable**, che forniscono funzionalità speciali che consentono di migliorare le prestazioni dell'applicazione. Esempi di oggetti **Freezable** includono pennelli, penne, trasformazioni, geometrie e animazioni.

Che cos'è un oggetto **Freezable**?

Un **Freezable** è un tipo speciale di oggetto con due stati: non bloccato e bloccato. Quando viene sbloccato, un **Freezable** sembra comportarsi come qualsiasi altro oggetto. Quando il blocco è bloccato, non è più possibile modificare un **Freezable**.

Un **Freezable** fornisce un evento **Changed** per notificare agli osservatori le eventuali modifiche apportate all'oggetto. Il blocco di un **Freezable** può migliorare le prestazioni, perché non è più necessario spendere risorse per le notifiche di modifica. Un **Freezable** bloccato può anche essere condiviso tra thread, mentre un **Freezable** non bloccato non può.

Anche se la classe **Freezable** dispone di molte applicazioni, la maggior parte degli oggetti **Freezable** in Windows Presentation Foundation (WPF) sono correlati al sottosistema grafico.

La classe **Freezable** rende più semplice l'utilizzo di determinati oggetti di sistema grafici e consente di migliorare le prestazioni dell'applicazione. Esempi di tipi che ereditano da **Freezable** includono le classi **Brush**, **Transform** e **Geometry**. Poiché contengono risorse non gestite, il sistema deve monitorare tali oggetti per le modifiche e quindi aggiornare le corrispondenti risorse non gestite quando viene apportata una modifica all'oggetto originale. Anche se non si modifica effettivamente un oggetto di sistema grafico, il sistema deve comunque dedicare alcune risorse al monitoraggio dell'oggetto, nel caso in cui venga modificato.

Si supponga, ad esempio, di creare un pennello **SolidColorBrush** e di usarlo per disegnare lo sfondo di un pulsante.

```
Button myButton = new Button();
SolidColorBrush myBrush = new SolidColorBrush(Colors.Yellow);
myButton.Background = myBrush;
```

```
Dim myButton As New Button()
Dim myBrush As New SolidColorBrush(Colors.Yellow)
myButton.Background = myBrush
```

Quando viene eseguito il rendering del pulsante, il sottosistema grafico WPF usa le informazioni fornite per disegnare un gruppo di pixel per creare l'aspetto di un pulsante. Sebbene sia stato usato un pennello tinta unita per descrivere il modo in cui deve essere disegnato il pulsante, il pennello tinta unita non esegue effettivamente il disegno. Il sistema grafico genera oggetti veloci e di basso livello per il pulsante e il pennello e sono gli oggetti effettivamente visualizzati sullo schermo.

Per modificare il pennello, è necessario rigenerare tali oggetti di basso livello. La classe **Freezable** fornisce a un pennello la possibilità di trovare gli oggetti di basso livello generati corrispondenti e di aggiornarli quando cambiano. Quando questa funzionalità è abilitata, il pennello viene definito "non bloccato".

Il metodo **Freeze** di un oggetto **Freezable** consente di disabilitare questa funzionalità di aggiornamento automatico. È possibile utilizzare questo metodo per fare in modo che il pennello diventi "bloccato" o non

modificabile.

NOTE

Non tutti gli oggetti **Freezable** possono essere bloccati. Per evitare di generare un [InvalidOperationException](#), controllare il valore della proprietà [CanFreeze](#) dell'oggetto **Freezable** per determinare se può essere bloccato prima di tentare di bloccarlo.

```
if (myBrush.CanFreeze)
{
    // Makes the brush unmodifiable.
    myBrush.Freeze();
}
```

```
If myBrush.CanFreeze Then
    ' Makes the brush unmodifiable.
    myBrush.Freeze()
End If
```

Quando non è più necessario modificare un oggetto **Freezable**, il blocco offre vantaggi in merito alle prestazioni. Se si blocca il pennello in questo esempio, il sistema grafico non dovrà più monitorarlo per le modifiche. Il sistema grafico può anche eseguire altre ottimizzazioni, perché sa che il pennello non cambierà.

NOTE

Per praticità, gli oggetti **Freezable** rimangono bloccati a meno che non vengano bloccati in modo esplicito.

Uso di **Freezable**

L'utilizzo di un oggetto **Freezable** sbloccato è analogo all'utilizzo di qualsiasi altro tipo di oggetto. Nell'esempio seguente, il colore di un [SolidColorBrush](#) viene modificato da giallo a rosso dopo che è stato usato per disegnare lo sfondo di un pulsante. Il sistema grafico funziona dietro le quinte per modificare automaticamente il pulsante da giallo a rosso alla successiva aggiornamento dello schermo.

```
Button myButton = new Button();
SolidColorBrush myBrush = new SolidColorBrush(Colors.Yellow);
myButton.Background = myBrush;

// Changes the button's background to red.
myBrush.Color = Colors.Red;
```

```
Dim myButton As New Button()
Dim myBrush As New SolidColorBrush(Colors.Yellow)
myButton.Background = myBrush

' Changes the button's background to red.
myBrush.Color = Colors.Red
```

Blocco di un oggetto **Freezable**

Per rendere immodificabile un [Freezable](#), chiamare il relativo metodo [Freeze](#). Quando si blocca un oggetto che contiene oggetti **Freezable**, anche questi oggetti sono bloccati. Se, ad esempio, si blocca un [PathGeometry](#), anche le figure e i segmenti in esso contenuti verrebbero bloccati.

Un oggetto **Freezable** **non può** essere bloccato se si verifica una delle condizioni seguenti:

- Dispone di proprietà animate o con associazione a dati.
- Dispone di proprietà impostate da una risorsa dinamica. Per ulteriori informazioni sulle risorse dinamiche, vedere le [risorse XAML](#).
- Contiene **Freezable** oggetti secondari che non possono essere bloccati.

Se queste condizioni sono false e non si intende modificare il **Freezable**, è consigliabile bloccarlo per ottenere i vantaggi in termini di prestazioni descritti in precedenza.

Una volta chiamato il metodo **Freeze** di un oggetto **Freezable**, non è più possibile modificarlo. Il tentativo di modificare un oggetto bloccato causa la generazione di un **InvalidOperationException**. Il codice seguente genera un'eccezione, perché si tenta di modificare il pennello dopo che è stato bloccato.

```
Button myButton = new Button();
SolidColorBrush myBrush = new SolidColorBrush(Colors.Yellow);

if (myBrush.CanFreeze)
{
    // Makes the brush unmodifiable.
    myBrush.Freeze();
}

myButton.Background = myBrush;

try {

    // Throws an InvalidOperationException, because the brush is frozen.
    myBrush.Color = Colors.Red;
} catch(InvalidOperationException ex)
{
    MessageBox.Show("Invalid operation: " + ex.ToString());
}
```

```
Dim myButton As New Button()
Dim myBrush As New SolidColorBrush(Colors.Yellow)

If myBrush.CanFreeze Then
    ' Makes the brush unmodifiable.
    myBrush.Freeze()
End If

myButton.Background = myBrush

Try

    ' Throws an InvalidOperationException, because the brush is frozen.
    myBrush.Color = Colors.Red
Catch ex As InvalidOperationException
    MessageBox.Show("Invalid operation: " & ex.ToString())
End Try
```

Per evitare di generare questa eccezione, è possibile usare il metodo **IsFrozen** per determinare se un **Freezable** è bloccato.

```

Button myButton = new Button();
SolidColorBrush myBrush = new SolidColorBrush(Colors.Yellow);

if (myBrush.CanFreeze)
{
    // Makes the brush unmodifiable.
    myBrush.Freeze();
}

myButton.Background = myBrush;

if (myBrush.IsFrozen) // Evaluates to true.
{
    // If the brush is frozen, create a clone and
    // modify the clone.
    SolidColorBrush myBrushClone = myBrush.Clone();
    myBrushClone.Color = Colors.Red;
    myButton.Background = myBrushClone;
}
else
{
    // If the brush is not frozen,
    // it can be modified directly.
    myBrush.Color = Colors.Red;
}

```

```

Dim myButton As New Button()
Dim myBrush As New SolidColorBrush(Colors.Yellow)

If myBrush.CanFreeze Then
    ' Makes the brush unmodifiable.
    myBrush.Freeze()
End If

myButton.Background = myBrush

If myBrush.IsFrozen Then ' Evaluates to true.
    ' If the brush is frozen, create a clone and
    ' modify the clone.
    Dim myBrushClone As SolidColorBrush = myBrush.Clone()
    myBrushClone.Color = Colors.Red
    myButton.Background = myBrushClone
Else
    ' If the brush is not frozen,
    ' it can be modified directly.
    myBrush.Color = Colors.Red
End If

```

Nell'esempio di codice precedente, è stata eseguita una copia modificabile di un oggetto bloccato usando il metodo [Clone](#). La sezione successiva illustra la clonazione in modo più dettagliato.

NOTE

Poiché non è possibile animare un **Freezable** bloccato, il sistema di animazione creerà automaticamente cloni modificabili di oggetti **Freezable** bloccati quando si tenta di animarli con una **Storyboard**. Per eliminare l'overhead delle prestazioni causato dalla clonazione, lasciare un oggetto non bloccato se si intende animarlo. Per ulteriori informazioni sull'animazione con gli storyboard, vedere [Cenni preliminari sugli storyboard](#).

Blocco dal markup

Per bloccare un oggetto **Freezable** dichiarato nel markup, usare l'attributo `PresentationOptions:Freeze`.

Nell'esempio seguente una **SolidColorBrush** viene dichiarata come risorsa di pagina e bloccata. Viene quindi usato per impostare lo sfondo di un pulsante.

```
<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:PresentationOptions="http://schemas.microsoft.com/winfx/2006/xaml/presentation/options"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="PresentationOptions">

    <Page.Resources>

        <!-- This resource is frozen. -->
        <SolidColorBrush
            x:Key="MyBrush"
            PresentationOptions:Freeze="True"
            Color="Red" />
    </Page.Resources>

    <StackPanel>

        <Button Content="A Button"
            Background="{StaticResource MyBrush}">
        </Button>

    </StackPanel>
</Page>
```

Per utilizzare l'attributo `Freeze`, è necessario eseguire il mapping allo spazio dei nomi delle opzioni di presentazione: `http://schemas.microsoft.com/winfx/2006/xaml/presentation/options`. `PresentationOptions` è il prefisso consigliato per il mapping di questo spazio dei nomi:

```
xmlns:PresentationOptions="http://schemas.microsoft.com/winfx/2006/xaml/presentation/options"
```

Poiché non tutti i lettori XAML riconoscono questo attributo, è consigliabile usare l' [attributo MC: Ignorable](#) per contrassegnare l'attributo `Presentation:Freeze` come ignorabile:

```
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="PresentationOptions"
```

Per ulteriori informazioni, vedere la pagina relativa all' [attributo MC: Ignorable](#).

"Sblocco" di un oggetto **Freezable**

Una volta bloccato, un **Freezable** non può mai essere modificato o sbloccato; Tuttavia, è possibile creare un clone non bloccato usando il metodo **Clone** o **CloneCurrentValue**.

Nell'esempio seguente lo sfondo del pulsante viene impostato con un pennello e il pennello viene quindi

bloccato. Viene eseguita una copia non bloccata del pennello utilizzando il metodo [Clone](#). Il clone viene modificato e usato per modificare lo sfondo del pulsante da giallo a rosso.

```
Button myButton = new Button();
SolidColorBrush myBrush = new SolidColorBrush(Colors.Yellow);

// Freezing a Freezable before it provides
// performance improvements if you don't
// intend on modifying it.
if (myBrush.CanFreeze)
{
    // Makes the brush unmodifiable.
    myBrush.Freeze();
}

myButton.Background = myBrush;

// If you need to modify a frozen brush,
// the Clone method can be used to
// create a modifiable copy.
SolidColorBrush myBrushClone = myBrush.Clone();

// Changing myBrushClone does not change
// the color of myButton, because its
// background is still set by myBrush.
myBrushClone.Color = Colors.Red;

// Replacing myBrush with myBrushClone
// makes the button change to red.
myButton.Background = myBrushClone;
```

```
Dim myButton As New Button()
Dim myBrush As New SolidColorBrush(Colors.Yellow)

' Freezing a Freezable before it provides
' performance improvements if you don't
' intend on modifying it.
If myBrush.CanFreeze Then
    ' Makes the brush unmodifiable.
    myBrush.Freeze()
End If

myButton.Background = myBrush

' If you need to modify a frozen brush,
' the Clone method can be used to
' create a modifiable copy.
Dim myBrushClone As SolidColorBrush = myBrush.Clone()

' Changing myBrushClone does not change
' the color of myButton, because its
' background is still set by myBrush.
myBrushClone.Color = Colors.Red

' Replacing myBrush with myBrushClone
' makes the button change to red.
myButton.Background = myBrushClone
```

NOTE

Indipendentemente dal metodo di clonazione usato, le animazioni non vengono mai copiate nella nuova [Freezable](#).

I metodi [Clone](#) e [CloneCurrentValue](#) producono copie complete dell'oggetto Freezable. Se l'oggetto Freezable contiene altri oggetti Freezable bloccati, questi vengono anche clonati e resi modificabili. Se, ad esempio, si clona un [PathGeometry](#) bloccato per renderlo modificabile, anche le figure e i segmenti in esso contenuti vengono copiati e resi modificabili.

Creazione di una classe Freezable personalizzata

Una classe che deriva da [Freezable](#) acquisisce le funzionalità seguenti.

- Stati speciali: uno stato di sola lettura (bloccato) e scrivibile.
- Thread safety: un [Freezable](#) bloccato può essere condiviso tra thread.
- Notifica dettagliata delle modifiche: diversamente da altre [DependencyObject](#)s, gli oggetti Freezable forniscono notifiche di modifica quando i valori delle sottoproprietà cambiano.
- Clonazione semplice: la classe Freezable ha già implementato diversi metodi che producono cloni profondi.

Un [Freezable](#) è un tipo di [DependencyObject](#)e pertanto utilizza il sistema di proprietà di dipendenza. Le proprietà della classe non devono essere proprietà di dipendenza, ma l'uso delle proprietà di dipendenza ridurrà la quantità di codice da scrivere, perché la classe [Freezable](#) è stata progettata con le proprietà di dipendenza. Per ulteriori informazioni sul sistema di proprietà di dipendenza, vedere [Cenni preliminari sulle proprietà di dipendenza](#).

Ogni sottoclasse [Freezable](#) deve eseguire l'override del metodo [CreateInstanceCore](#). Se la classe USA proprietà di dipendenza per tutti i dati, l'operazione è terminata.

Se la classe contiene membri dati di proprietà non di dipendenza, è necessario anche eseguire l'override dei metodi seguenti:

- [CloneCore](#)
- [CloneCurrentValueCore](#)
- [GetAsFrozenCore](#)
- [GetCurrentValueAsFrozenCore](#)
- [FreezeCore](#)

È inoltre necessario osservare le regole seguenti per l'accesso e la scrittura a membri dati che non sono proprietà di dipendenza:

- All'inizio di qualsiasi API che legge i membri dati della proprietà non di dipendenza, chiamare il metodo [ReadPreamble](#).
- All'inizio di qualsiasi API che scriva membri dati di proprietà non di dipendenza, chiamare il metodo [WritePreamble](#). Dopo aver chiamato [WritePreamble](#) in un'API, non è necessario effettuare una chiamata aggiuntiva a [ReadPreamble](#) se si leggono anche i membri dati della proprietà non di dipendenza.
- Chiamare il metodo [WritePostscript](#) prima di uscire da metodi che scrivono in membri dati di proprietà non di dipendenza.

Se la classe contiene membri dati di proprietà non dipendenza [DependencyObject](#) oggetti, è necessario chiamare anche il metodo [OnFreezablePropertyChanged](#) ogni volta che si modifica uno dei relativi valori, anche se si imposta il membro su `null`.

NOTE

È molto importante iniziare ogni [Freezable](#) metodo di cui si esegue l'override con una chiamata all'implementazione di base.

Per un esempio di classe di [Freezable](#) personalizzata, vedere l'[esempio di animazione personalizzata](#).

Vedere anche

- [Freezable](#)
- [Esempio di animazione personalizzata](#)
- [Cenni preliminari sulle proprietà di dipendenza](#)
- [Proprietà di dipendenza personalizzate](#)

Panoramica su allineamento, margini e spaziatura interna

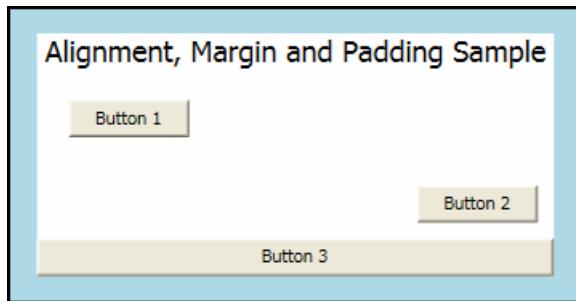
23/10/2019 • 25 minutes to read • [Edit Online](#)

La [FrameworkElement](#) classe espone diverse proprietà che vengono usate per posizionare con precisione gli elementi figlio. In questo argomento vengono illustrate quattro delle proprietà più [HorizontalAlignment](#) importanti [Margin](#), [Padding](#), [VerticalAlignment](#). È fondamentale comprendere gli effetti di queste proprietà, perché forniscono la base per controllare la posizione degli elementi nelle applicazioni Windows Presentation Foundation (WPF).

Introduzione al posizionamento degli elementi

È possibile posizionare gli elementi in molti modi con WPF. Tuttavia, il raggiungimento del layout ideale va oltre la [Panel](#) semplice scelta dell'elemento corretto. Per un controllo accurato del posizionamento è necessario conoscere [Margin](#), [Padding](#), [HorizontalAlignment](#), [VerticalAlignment](#), e.

La figura seguente illustra uno scenario di layout in cui vengono usate diverse proprietà di posizionamento.



A prima vista, gli [Button](#) elementi in questa illustrazione potrebbero sembrare posizionati in modo casuale. Le posizioni sono in realtà controllate con precisione usando una combinazione di margini, allineamento e spaziatura interna.

L'esempio seguente descrive come creare il layout nella figura precedente. Un [Border](#) elemento incapsula un elemento padre [StackPanel](#) con un [Padding](#) valore di 15 pixel indipendenti dal dispositivo. Questo account per la banda [LightBlue](#) stretta che racchiude l'elemento figlio [StackPanel](#). Gli elementi figlio di [StackPanel](#) vengono utilizzati per illustrare ognuna delle varie proprietà di posizionamento descritte in dettaglio in questo argomento. Vengono [Button](#) usati tre elementi per illustrare le [Margin](#) proprietà [HorizontalAlignment](#) e.

```
// Create the application's main Window.  
mainWindow = new Window();  
mainWindow.Title = "Margins, Padding and Alignment Sample";  
  
// Add a Border  
myBorder = new Border();  
myBorder.Background = Brushes.LightBlue;  
myBorder.BorderBrush = Brushes.Black;  
myBorder.Padding = new Thickness(15);  
myBorder.BorderThickness = new Thickness(2);  
  
myStackPanel = new StackPanel();  
myStackPanel.Background = Brushes.White;  
myStackPanel.HorizontalAlignment = HorizontalAlignment.Center;  
myStackPanel.VerticalAlignment = VerticalAlignment.Top;  
  
TextBlock myTextBlock = new TextBlock();  
myTextBlock.Margin = new Thickness(5, 0, 5, 0);  
myTextBlock.FontSize = 18;  
myTextBlock.HorizontalAlignment = HorizontalAlignment.Center;  
myTextBlock.Text = "Alignment, Margin and Padding Sample";  
Button myButton1 = new Button();  
myButton1.HorizontalAlignment = HorizontalAlignment.Left;  
myButton1.Margin = new Thickness(20);  
myButton1.Content = "Button 1";  
Button myButton2 = new Button();  
myButton2.HorizontalAlignment = HorizontalAlignment.Right;  
myButton2.Margin = new Thickness(10);  
myButton2.Content = "Button 2";  
Button myButton3 = new Button();  
myButton3.HorizontalAlignment = HorizontalAlignment.Stretch;  
myButton3.Margin = new Thickness(0);  
myButton3.Content = "Button 3";  
  
// Add child elements to the parent StackPanel.  
myStackPanel.Children.Add(myTextBlock);  
myStackPanel.Children.Add(myButton1);  
myStackPanel.Children.Add(myButton2);  
myStackPanel.Children.Add(myButton3);  
  
// Add the StackPanel as the lone Child of the Border.  
myBorder.Child = myStackPanel;  
  
// Add the Border as the Content of the Parent Window Object.  
mainWindow.Content = myBorder;  
mainWindow.Show();
```

```

WindowTitle = "Margins, Padding and Alignment Sample"

'Add a Border.
Dim myBorder As New Border()
myBorder.Background = Brushes.LightBlue
myBorder.BorderBrush = Brushes.Black
myBorder.Padding = New Thickness(15)
myBorder.BorderThickness = New Thickness(2)

Dim myStackPanel As New StackPanel()
myStackPanel.Background = Brushes.White
myStackPanel.HorizontalAlignment = Windows.HorizontalAlignment.Center
myStackPanel.VerticalAlignment = Windows.VerticalAlignment.Top

Dim myTextBlock As New TextBlock()
myTextBlock.Margin = New Thickness(5, 0, 5, 0)
myTextBlock.FontSize = 18
myTextBlock.HorizontalAlignment = Windows.HorizontalAlignment.Center
myTextBlock.Text = "Alignment, Margin and Padding Sample"
Dim myButton1 As New Button()
myButton1.HorizontalAlignment = Windows.HorizontalAlignment.Left
myButton1.Margin = New Thickness(20)
myButton1.Content = "Button 1"
Dim myButton2 As New Button()
myButton2.HorizontalAlignment = Windows.HorizontalAlignment.Right
myButton2.Margin = New Thickness(10)
myButton2.Content = "Button 2"
Dim myButton3 As New Button()
myButton3.HorizontalAlignment = Windows.HorizontalAlignment.Stretch
myButton3.Margin = New Thickness(0)
myButton3.Content = "Button 3"

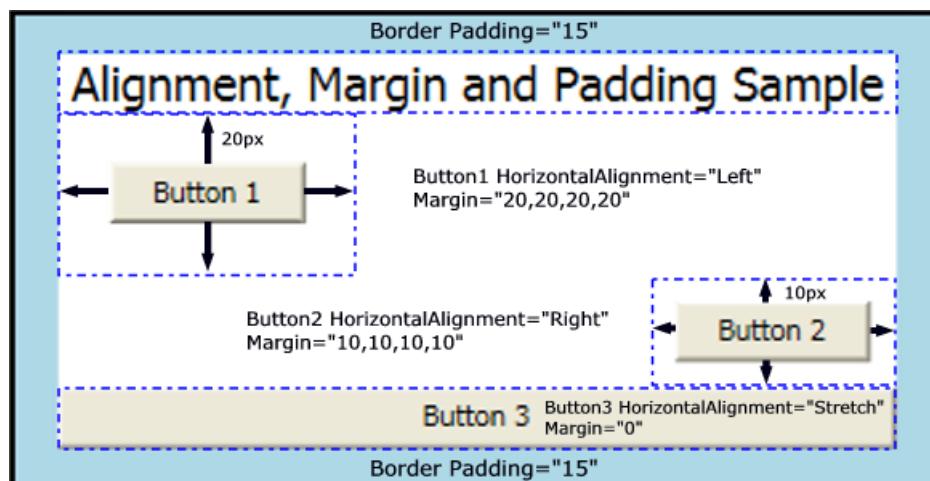
'Add child elements to the parent StackPanel.
myStackPanel.Children.Add(myTextBlock)
myStackPanel.Children.Add(myButton1)
myStackPanel.Children.Add(myButton2)
myStackPanel.Children.Add(myButton3)

'Add the StackPanel as the lone Child of the Border.
myBorder.Child = myStackPanel

' Add the Canvas as the lone Child of the Border
myBorder.Child = myStackPanel
Me.Content = myBorder

```

Il diagramma seguente mostra una visualizzazione dettagliata delle varie proprietà di posizionamento usate nell'esempio precedente. Le sezioni successive di questo argomento descrivono in maggior dettaglio come usare ogni proprietà di posizionamento.



Informazioni sulle proprietà di allineamento

Le [HorizontalAlignment](#) proprietà [VerticalAlignment](#) e descrivono come posizionare un elemento figlio all'interno dello spazio di layout allocato di un elemento padre. Usando insieme queste proprietà, è possibile posizionare con precisione gli elementi figlio. Gli elementi figlio [DockPanel](#) di un oggetto possono ad esempio specificare quattro diversi allineamenti [Left](#)orizzontali, [Center](#) Rightovvero, o [Stretch](#) oppure su per riempire lo spazio disponibile. Per il posizionamento verticale sono disponibili valori analoghi.

NOTE

Le [Height](#) [Width](#) proprietà impostate in modo esplicito su un elemento hanno la precedenza [Stretch](#) sul valore della proprietà. Il [Height](#) [Width](#) [HorizontalAlignment](#) [Stretch](#) tentativo di impostare, e un valore dei risultati nella richiesta viene ignorato. [Stretch](#)

Proprietà HorizontalAlignment

La [HorizontalAlignment](#) proprietà dichiara le caratteristiche di allineamento orizzontale da applicare agli elementi figlio. Nella tabella seguente vengono illustrati i valori possibili della [HorizontalAlignment](#) proprietà.

MEMBER	DESCRIZIONE
Left	Gli elementi figlio sono allineati a sinistra dello spazio di layout allocato dell'elemento padre.
Center	Gli elementi figlio sono allineati al centro dello spazio di layout allocato dell'elemento padre.
Right	Gli elementi figlio sono allineati a destra dello spazio di layout allocato dell'elemento padre.
Stretch Predefinita	Gli elementi figlio vengono estesi fino a riempire lo spazio di layout allocato dell'elemento padre. I Width valori Height e esplicativi hanno la precedenza.

Nell'esempio seguente viene illustrato come applicare la [HorizontalAlignment](#) proprietà agli [Button](#) elementi. Vengono visualizzati tutti i valori di attributo per illustrare i diversi comportamenti di rendering.

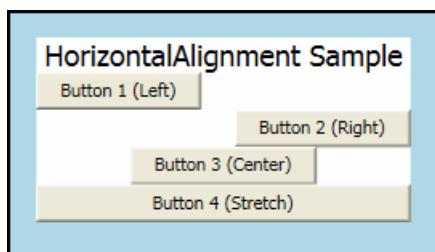
```
Button myButton1 = new Button();
myButton1.HorizontalAlignment = HorizontalAlignment.Left;
myButton1.Content = "Button 1 (Left)";
Button myButton2 = new Button();
myButton2.HorizontalAlignment = HorizontalAlignment.Right;
myButton2.Content = "Button 2 (Right)";
Button myButton3 = new Button();
myButton3.HorizontalAlignment = HorizontalAlignment.Center;
myButton3.Content = "Button 3 (Center)";
Button myButton4 = new Button();
myButton4.HorizontalAlignment = HorizontalAlignment.Stretch;
myButton4.Content = "Button 4 (Stretch);
```

```

Dim myButton1 As New Button()
myButton1.HorizontalAlignment = Windows.HorizontalAlignment.Left
myButton1.Margin = New Thickness(20)
myButton1.Content = "Button 1"
Dim myButton2 As New Button()
myButton2.HorizontalAlignment = Windows.HorizontalAlignment.Right
myButton2.Margin = New Thickness(10)
myButton2.Content = "Button 2"
Dim myButton3 As New Button()
myButton3.HorizontalAlignment = Windows.HorizontalAlignment.Center
myButton3.Margin = New Thickness(0)
myButton3.Content = "Button 3"
Dim myButton4 As New Button()
myButton4.HorizontalAlignment = Windows.HorizontalAlignment.Stretch
myButton4.Content = "Button 4 (Stretch)"

```

Il risultato del codice precedente è simile a quello riportato nell'immagine seguente. Gli effetti del posizionamento di [HorizontalAlignment](#) ogni valore sono visibili nella figura.



Proprietà VerticalAlignment

La [VerticalAlignment](#) proprietà descrive le caratteristiche di allineamento verticale da applicare agli elementi figlio. Nella tabella seguente vengono illustrati tutti i valori possibili per [VerticalAlignment](#) la proprietà.

MEMBER	DESCRIZIONE
Top	Gli elementi figlio sono allineati in alto nello spazio di layout allocato dell'elemento padre.
Center	Gli elementi figlio sono allineati al centro dello spazio di layout allocato dell'elemento padre.
Bottom	Gli elementi figlio sono allineati in basso nello spazio di layout allocato dell'elemento padre.
Stretch	Gli elementi figlio vengono estesi fino a riempire lo spazio di layout allocato dell'elemento padre. I Width valori Height e esplicativi hanno la precedenza.

Nell'esempio seguente viene illustrato come applicare la [VerticalAlignment](#) proprietà agli [Button](#) elementi. Vengono visualizzati tutti i valori di attributo per illustrare i diversi comportamenti di rendering. Ai fini di questo esempio, viene [Grid](#) utilizzato un elemento con linee della griglia visibile come elemento padre, per illustrare meglio il comportamento del layout di ogni valore della proprietà.

```
TextBlock myTextBlock = new TextBlock();
myTextBlock.FontSize = 18;
myTextBlock.HorizontalAlignment = HorizontalAlignment.Center;
myTextBlock.Text = "VerticalAlignment Sample";
Grid.SetRow(myTextBlock, 0);
Button myButton1 = new Button();
myButton1.VerticalAlignment = VerticalAlignment.Top;
myButton1.Content = "Button 1 (Top)";
Grid.SetRow(myButton1, 1);
Button myButton2 = new Button();
myButton2.VerticalAlignment = VerticalAlignment.Bottom;
myButton2.Content = "Button 2 (Bottom)";
Grid.SetRow(myButton2, 2);
Button myButton3 = new Button();
myButton3.VerticalAlignment = VerticalAlignment.Center;
myButton3.Content = "Button 3 (Center)";
Grid.SetRow(myButton3, 3);
Button myButton4 = new Button();
myButton4.VerticalAlignment = VerticalAlignment.Stretch;
myButton4.Content = "Button 4 (Stretch)";
Grid.SetRow(myButton4, 4);
```

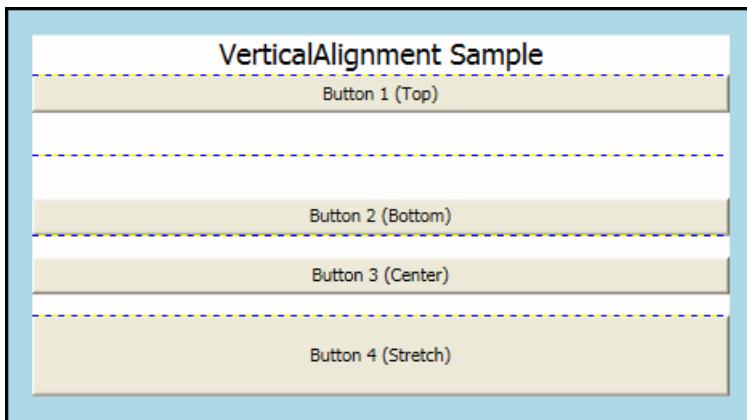
```
Dim myTextBlock As New TextBlock()
myTextBlock.FontSize = 18
myTextBlock.HorizontalAlignment = Windows.HorizontalAlignment.Center
myTextBlock.Text = "VerticalAlignment Sample"
Grid.SetRow(myTextBlock, 0)
Dim myButton1 As New Button()
myButton1.VerticalAlignment = Windows.VerticalAlignment.Top
myButton1.Content = "Button 1 (Top)"
Grid.SetRow(myButton1, 1)
Dim myButton2 As New Button()
myButton2.VerticalAlignment = Windows.VerticalAlignment.Bottom
myButton2.Content = "Button 2 (Bottom)"
Grid.SetRow(myButton2, 2)
Dim myButton3 As New Button()
myButton3.VerticalAlignment = Windows.VerticalAlignment.Center
myButton3.Content = "Button 3 (Center)"
Grid.SetRow(myButton3, 3)
Dim myButton4 As New Button()
myButton4.VerticalAlignment = Windows.VerticalAlignment.Stretch
myButton4.Content = "Button 4 (Stretch)"
Grid.SetRow(myButton4, 4)
```

```

<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      WindowTitle="VerticalAlignment Sample">
    <Border Background="LightBlue" BorderBrush="Black" BorderThickness="2" Padding="15">
        <Grid Background="White" ShowGridLines="True">
            <Grid.RowDefinitions>
                <RowDefinition Height="25"/>
                <RowDefinition Height="50"/>
                <RowDefinition Height="50"/>
                <RowDefinition Height="50"/>
                <RowDefinition Height="50"/>
            </Grid.RowDefinitions>
            <TextBlock Grid.Row="0" Grid.Column="0" FontSize="18"
HorizontalAlignment="Center">VerticalAlignment Sample</TextBlock>
            <Button Grid.Row="1" Grid.Column="0" VerticalAlignment="Top">Button 1 (Top)</Button>
            <Button Grid.Row="2" Grid.Column="0" VerticalAlignment="Bottom">Button 2 (Bottom)</Button>
            <Button Grid.Row="3" Grid.Column="0" VerticalAlignment="Center">Button 3 (Center)</Button>
            <Button Grid.Row="4" Grid.Column="0" VerticalAlignment="Stretch">Button 4 (Stretch)</Button>
        </Grid>
    </Border>
</Page>

```

Il risultato del codice precedente è simile a quello riportato nell'immagine seguente. Gli effetti del posizionamento di **VerticalAlignment** ogni valore sono visibili nella figura.



Informazioni sulla proprietà Margin

La **Margin** proprietà descrive la distanza tra un elemento e i relativi elementi figlio o peer. **Margini** valori possono essere uniformi usando una `Margin="20"` sintassi simile a. Con questa sintassi, all'elemento **Margin** viene applicata un'uniforme di 20 pixel indipendenti dal dispositivo. **Margini** valori possono anche assumere il formato di quattro valori distinti, ognuno dei quali descrive un margine distinto da applicare a sinistra, in alto, a destra e in basso (in questo ordine) `Margin="0,10,5,25"`, ad esempio. L' **Margin** uso corretto della proprietà consente un controllo molto preciso della posizione di rendering di un elemento e della posizione di rendering degli elementi adiacenti e degli elementi figlio.

NOTE

Un margine diverso da zero applica uno spazio all'esterno degli **ActualWidth** elementi **ActualHeight**.

Nell'esempio seguente viene illustrato come applicare margini uniformi intorno a **Button** un gruppo di elementi. Gli **Button** elementi sono spaziati uniformemente con un buffer di margine di dieci pixel in ogni direzione.

```
Button^ myButton7 = gcnew Button();
myButton7->Margin = Thickness(10);
myButton7->Content = "Button 7";
Button^ myButton8 = gcnew Button();
myButton8->Margin = Thickness(10);
myButton8->Content = "Button 8";
Button^ myButton9 = gcnew Button();
myButton9->Margin = Thickness(10);
myButton9->Content = "Button 9";
```

```
Button myButton7 = new Button();
myButton7.Margin = new Thickness(10);
myButton7.Content = "Button 7";
Button myButton8 = new Button();
myButton8.Margin = new Thickness(10);
myButton8.Content = "Button 8";
Button myButton9 = new Button();
myButton9.Margin = new Thickness(10);
myButton9.Content = "Button 9";
```

```
Dim myButton7 As New Button
myButton7.Margin = New Thickness(10)
myButton7.Content = "Button 7"
Dim myButton8 As New Button
myButton8.Margin = New Thickness(10)
myButton8.Content = "Button 8"
Dim myButton9 As New Button
myButton9.Margin = New Thickness(10)
myButton9.Content = "Button 9"
```

```
<Button Margin="10">Button 7</Button>
<Button Margin="10">Button 8</Button>
<Button Margin="10">Button 9</Button>
```

In diverse situazioni un margine uniforme non è appropriato. In questi casi, è possibile applicare una spaziatura non uniforme. L'esempio seguente illustra come applicare una spaziatura dei margini non uniforme agli elementi figlio. I margini sono descritti in quest'ordine: sinistro, superiore, destro, inferiore.

```
Button^ myButton1 = gcnew Button();
myButton1->Margin = Thickness(0, 10, 0, 10);
myButton1->Content = "Button 1";
Button^ myButton2 = gcnew Button();
myButton2->Margin = Thickness(0, 10, 0, 10);
myButton2->Content = "Button 2";
Button^ myButton3 = gcnew Button();
myButton3->Margin = Thickness(0, 10, 0, 10);
```

```
Button myButton1 = new Button();
myButton1.Margin = new Thickness(0, 10, 0, 10);
myButton1.Content = "Button 1";
Button myButton2 = new Button();
myButton2.Margin = new Thickness(0, 10, 0, 10);
myButton2.Content = "Button 2";
Button myButton3 = new Button();
myButton3.Margin = new Thickness(0, 10, 0, 10);
```

```
Dim myButton1 As New Button
myButton1.Margin = New Thickness(0, 10, 0, 10)
myButton1.Content = "Button 1"
Dim myButton2 As New Button
myButton2.Margin = New Thickness(0, 10, 0, 10)
myButton2.Content = "Button 2"
Dim myButton3 As New Button
myButton3.Margin = New Thickness(0, 10, 0, 10)
```

```
<Button Margin="0,10,0,10">Button 1</Button>
<Button Margin="0,10,0,10">Button 2</Button>
<Button Margin="0,10,0,10">Button 3</Button>
```

Informazioni sulla proprietà Padding

La spaziatura interna **Margin** è simile alla maggior parte degli aspetti. La proprietà **Padding** viene esposta solo in alcune classi **Block**, principalmente per praticità: **Control**, **Border**, e **TextBlock** sono esempi di classi che espongono una proprietà **Padding**. La **Padding** proprietà consente di ingrandire le dimensioni effettive di un elemento figlio **Thickness** in base al valore specificato.

Nell'esempio seguente viene illustrato come applicare **Padding** a un elemento **Border** padre.

```
myBorder = gcnew Border();
myBorder->Background = Brushes::LightBlue;
myBorder->BorderBrush = Brushes::Black;
myBorder->BorderThickness = Thickness(2);
myBorder->CornerRadius = CornerRadius(45);
myBorder->Padding = Thickness(25);
```

```
myBorder = new Border();
myBorder.Background = Brushes.LightBlue;
myBorder.BorderBrush = Brushes.Black;
myBorder.BorderThickness = new Thickness(2);
myBorder.CornerRadius = new CornerRadius(45);
myBorder.Padding = new Thickness(25);
```

```
Dim myBorder As New Border
myBorder.Background = Brushes.LightBlue
myBorder.BorderBrush = Brushes.Black
myBorder.BorderThickness = New Thickness(2)
myBorder.CornerRadius = New CornerRadius(45)
myBorder.Padding = New Thickness(25)
```

```
<Border Background="LightBlue"
        BorderBrush="Black"
        BorderThickness="2"
        CornerRadius="45"
        Padding="25">
```

Uso di allineamento, margini e spaziatura interna in un'applicazione

HorizontalAlignment, e forniscono **VerticalAlignment** il controllo di posizionamento necessario per creare un oggetto interfaccia utente complesso. **Margin Padding** È possibile usare gli effetti di ogni proprietà per modificare il posizionamento degli elementi figlio, ottenendo in questo modo flessibilità nella creazione di applicazioni ed

esperienze utente dinamiche.

L'esempio seguente illustra ognuno dei concetti descritti in questo argomento. Basandosi sull'infrastruttura disponibile nel primo esempio di questo argomento, in questo esempio viene aggiunto [Grid](#) un elemento come figlio [Border](#) di nel primo esempio. [Padding](#) viene applicato all'elemento padre [Border](#). Viene usato per partizionare lo spazio tra tre [StackPanel](#) elementi figlio. [Grid](#) [Button](#) gli elementi vengono nuovamente utilizzati per visualizzare i vari effetti [Margin](#) di [HorizontalAlignment](#). [TextBlock](#) gli elementi vengono aggiunti a [ColumnDefinition](#) ognuno per definire meglio le varie proprietà applicate [Button](#) agli elementi in ogni colonna.

```
mainWindow = gcnew Window();

myBorder = gcnew Border();
myBorder->Background = Brushes::LightBlue;
myBorder->BorderBrush = Brushes::Black;
myBorder->BorderThickness = Thickness(2);
myBorder->CornerRadius = CornerRadius(45);
myBorder->Padding = Thickness(25);

// Define the Grid.
myGrid = gcnew Grid();
myGrid->Background = Brushes::White;
myGrid->ShowGridLines = true;

// Define the Columns.
ColumnDefinition^ myColDef1 = gcnew ColumnDefinition();
myColDef1->Width = GridLength(1, GridUnitType::Auto);
ColumnDefinition^ myColDef2 = gcnew ColumnDefinition();
myColDef2->Width = GridLength(1, GridUnitType::Star);
ColumnDefinition^ myColDef3 = gcnew ColumnDefinition();
myColDef3->Width = GridLength(1, GridUnitType::Auto);

// Add the ColumnDefinitions to the Grid.
myGrid->ColumnDefinitions->Add(myColDef1);
myGrid->ColumnDefinitions->Add(myColDef2);
myGrid->ColumnDefinitions->Add(myColDef3);

// Add the first child StackPanel.
StackPanel^ myStackPanel = gcnew StackPanel();
myStackPanel->HorizontalAlignment = HorizontalAlignment::Left;
myStackPanel->VerticalAlignment = VerticalAlignment::Top;
Grid::SetColumn(myStackPanel, 0);
Grid::SetRow(myStackPanel, 0);
TextBlock^ myTextBlock1 = gcnew TextBlock();
myTextBlock1->FontSize = 18;
myTextBlock1->HorizontalAlignment = HorizontalAlignment::Center;
myTextBlock1->Margin = Thickness(0, 0, 0, 15);
myTextBlock1->Text = "StackPanel 1";
Button^ myButton1 = gcnew Button();
myButton1->Margin = Thickness(0, 10, 0, 10);
myButton1->Content = "Button 1";
Button^ myButton2 = gcnew Button();
myButton2->Margin = Thickness(0, 10, 0, 10);
myButton2->Content = "Button 2";
Button^ myButton3 = gcnew Button();
myButton3->Margin = Thickness(0, 10, 0, 10);
TextBlock^ myTextBlock2 = gcnew TextBlock();
myTextBlock2->Text = "ColumnDefinition.Width = \"Auto\"";
TextBlock^ myTextBlock3 = gcnew TextBlock();
myTextBlock3->Text = "StackPanel.HorizontalAlignment = \"Left\"";
TextBlock^ myTextBlock4 = gcnew TextBlock();
myTextBlock4->Text = "StackPanel.VerticalAlignment = \"Top\"";
TextBlock^ myTextBlock5 = gcnew TextBlock();
myTextBlock5->Text = "StackPanel.Orientation = \"Vertical\"";
TextBlock^ myTextBlock6 = gcnew TextBlock();
myTextBlock6->Text = "Button.Margin = \"1,10,0,10\"";
myStackPanel->Children->Add(myTextBlock1);
```

```

myStackPanel->Children->Add(myButton1);
myStackPanel->Children->Add(myButton2);
myStackPanel->Children->Add(myButton3);
myStackPanel->Children->Add(myTextBlock2);
myStackPanel->Children->Add(myTextBlock3);
myStackPanel->Children->Add(myTextBlock4);
myStackPanel->Children->Add(myTextBlock5);
myStackPanel->Children->Add(myTextBlock6);

// Add the second child StackPanel.
StackPanel^ myStackPanel2 = gcnew StackPanel();
myStackPanel2->HorizontalAlignment = HorizontalAlignment::Stretch;
myStackPanel2->VerticalAlignment = VerticalAlignment::Top;
myStackPanel2->Orientation = Orientation::Vertical;
Grid::SetColumn(myStackPanel2, 1);
Grid::SetRow(myStackPanel2, 0);
TextBlock^ myTextBlock7 = gcnew TextBlock();
myTextBlock7->FontSize = 18;
myTextBlock7->HorizontalAlignment = HorizontalAlignment::Center;
myTextBlock7->Margin = Thickness(0, 0, 0, 15);
myTextBlock7->Text = "StackPanel 2";
Button^ myButton4 = gcnew Button();
myButton4->Margin = Thickness(10, 0, 10, 0);
myButton4->Content = "Button 4";
Button^ myButton5 = gcnew Button();
myButton5->Margin = Thickness(10, 0, 10, 0);
myButton5->Content = "Button 5";
Button^ myButton6 = gcnew Button();
myButton6->Margin = Thickness(10, 0, 10, 0);
myButton6->Content = "Button 6";
TextBlock^ myTextBlock8 = gcnew TextBlock();
myTextBlock8->HorizontalAlignment = HorizontalAlignment::Center;
myTextBlock8->Text = "ColumnDefinition.Width = \\"*\\"";
TextBlock^ myTextBlock9 = gcnew TextBlock();
myTextBlock9->HorizontalAlignment = HorizontalAlignment::Center;
myTextBlock9->Text = "StackPanel.HorizontalAlignment = \"Stretch\"";
TextBlock^ myTextBlock10 = gcnew TextBlock();
myTextBlock10->HorizontalAlignment = HorizontalAlignment::Center;
myTextBlock10->Text = "StackPanel.VerticalAlignment = \"Top\"";
TextBlock^ myTextBlock11 = gcnew TextBlock();
myTextBlock11->HorizontalAlignment = HorizontalAlignment::Center;
myTextBlock11->Text = "StackPanel.Orientation = \"Horizontal\"";
TextBlock^ myTextBlock12 = gcnew TextBlock();
myTextBlock12->HorizontalAlignment = HorizontalAlignment::Center;
myTextBlock12->Text = "Button.Margin = \"10,0,10,0\"";
myStackPanel2->Children->Add(myTextBlock7);
myStackPanel2->Children->Add(myButton4);
myStackPanel2->Children->Add(myButton5);
myStackPanel2->Children->Add(myButton6);
myStackPanel2->Children->Add(myTextBlock8);
myStackPanel2->Children->Add(myTextBlock9);
myStackPanel2->Children->Add(myTextBlock10);
myStackPanel2->Children->Add(myTextBlock11);
myStackPanel2->Children->Add(myTextBlock12);

// Add the final child StackPanel.
StackPanel^ myStackPanel3 = gcnew StackPanel();
myStackPanel3->HorizontalAlignment = HorizontalAlignment::Left;
myStackPanel3->VerticalAlignment = VerticalAlignment::Top;
Grid::SetColumn(myStackPanel3, 2);
Grid::SetRow(myStackPanel3, 0);
TextBlock^ myTextBlock13 = gcnew TextBlock();
myTextBlock13->FontSize = 18;
myTextBlock13->HorizontalAlignment = HorizontalAlignment::Center;
myTextBlock13->Margin = Thickness(0, 0, 0, 15);
myTextBlock13->Text = "StackPanel 3";
Button^ myButton7 = gcnew Button();
myButton7->Margin = Thickness(10);
myButton7->Content = "Button 7";

```

```

Button^ myButton8 = gcnew Button();
myButton8->Margin = Thickness(10);
myButton8->Content = "Button 8";
Button^ myButton9 = gcnew Button();
myButton9->Margin = Thickness(10);
myButton9->Content = "Button 9";
TextBlock^ myTextBlock14 = gcnew TextBlock();
myTextBlock14->Text = "ColumnDefinition.Width = \"Auto\"";
TextBlock^ myTextBlock15 = gcnew TextBlock();
myTextBlock15->Text = "StackPanel.HorizontalAlignment = \"Left\"";
TextBlock^ myTextBlock16 = gcnew TextBlock();
myTextBlock16->Text = "StackPanel.VerticalAlignment = \"Top\"";
TextBlock^ myTextBlock17 = gcnew TextBlock();
myTextBlock17->Text = "StackPanel.Orientation = \"Vertical\"";
TextBlock^ myTextBlock18 = gcnew TextBlock();
myTextBlock18->Text = "Button.Margin = \"10\"";
myStackPanel3->Children->Add(myTextBlock13);
myStackPanel3->Children->Add(myButton7);
myStackPanel3->Children->Add(myButton8);
myStackPanel3->Children->Add(myButton9);
myStackPanel3->Children->Add(myTextBlock14);
myStackPanel3->Children->Add(myTextBlock15);
myStackPanel3->Children->Add(myTextBlock16);
myStackPanel3->Children->Add(myTextBlock17);
myStackPanel3->Children->Add(myTextBlock18);

// Add child content to the parent Grid.
myGrid->Children->Add(myStackPanel);
myGrid->Children->Add(myStackPanel2);
myGrid->Children->Add(myStackPanel3);

// Add the Grid as the lone child of the Border.
myBorder->Child = myGrid;

// Add the Border to the Window as Content and show the Window.
mainWindow->Content = myBorder;
mainWindow->Title = "Margin, Padding, and Alignment Sample";
mainWindow->Show();

```

```

mainWindow = new Window();

myBorder = new Border();
myBorder.Background = Brushes.LightBlue;
myBorder.BorderBrush = Brushes.Black;
myBorder.BorderThickness = new Thickness(2);
myBorder.CornerRadius = new CornerRadius(45);
myBorder.Padding = new Thickness(25);

// Define the Grid.
myGrid = new Grid();
myGrid.Background = Brushes.White;
myGrid.ShowGridLines = true;

// Define the Columns.
ColumnDefinition myColDef1 = new ColumnDefinition();
myColDef1.Width = new GridLength(1, GridUnitType.Auto);
ColumnDefinition myColDef2 = new ColumnDefinition();
myColDef2.Width = new GridLength(1, GridUnitType.Star);
ColumnDefinition myColDef3 = new ColumnDefinition();
myColDef3.Width = new GridLength(1, GridUnitType.Auto);

// Add the ColumnDefinitions to the Grid.
myGrid.ColumnDefinitions.Add(myColDef1);
myGrid.ColumnDefinitions.Add(myColDef2);
myGrid.ColumnDefinitions.Add(myColDef3);

// Add the first child StackPanel.

```

```
StackPanel myStackPanel = new StackPanel();
myStackPanel.HorizontalAlignment = HorizontalAlignment.Left;
myStackPanel.VerticalAlignment = VerticalAlignment.Top;
Grid.SetColumn(myStackPanel, 0);
Grid.SetRow(myStackPanel, 0);
TextBlock myTextBlock1 = new TextBlock();
myTextBlock1.FontSize = 18;
myTextBlock1.HorizontalAlignment = HorizontalAlignment.Center;
myTextBlock1.Margin = new Thickness(0, 0, 0, 15);
myTextBlock1.Text = "StackPanel 1";
Button myButton1 = new Button();
myButton1.Margin = new Thickness(0, 10, 0, 10);
myButton1.Content = "Button 1";
Button myButton2 = new Button();
myButton2.Margin = new Thickness(0, 10, 0, 10);
myButton2.Content = "Button 2";
Button myButton3 = new Button();
myButton3.Margin = new Thickness(0, 10, 0, 10);
TextBlock myTextBlock2 = new TextBlock();
myTextBlock2.Text = @"ColumnDefinition.Width = ""Auto""";
TextBlock myTextBlock3 = new TextBlock();
myTextBlock3.Text = @"StackPanel.HorizontalAlignment = ""Left""";
TextBlock myTextBlock4 = new TextBlock();
myTextBlock4.Text = @"StackPanel.VerticalAlignment = ""Top""";
TextBlock myTextBlock5 = new TextBlock();
myTextBlock5.Text = @"StackPanel.Orientation = ""Vertical""";
TextBlock myTextBlock6 = new TextBlock();
myTextBlock6.Text = @"Button.Margin = ""1,10,0,10""";
myStackPanel.Children.Add(myTextBlock1);
myStackPanel.Children.Add(myButton1);
myStackPanel.Children.Add(myButton2);
myStackPanel.Children.Add(myButton3);
myStackPanel.Children.Add(myTextBlock2);
myStackPanel.Children.Add(myTextBlock3);
myStackPanel.Children.Add(myTextBlock4);
myStackPanel.Children.Add(myTextBlock5);
myStackPanel.Children.Add(myTextBlock6);

// Add the second child StackPanel.
StackPanel myStackPanel2 = new StackPanel();
myStackPanel2.HorizontalAlignment = HorizontalAlignment.Stretch;
myStackPanel2.VerticalAlignment = VerticalAlignment.Top;
myStackPanel2.Orientation = Orientation.Vertical;
Grid.SetColumn(myStackPanel2, 1);
Grid.SetRow(myStackPanel2, 0);
TextBlock myTextBlock7 = new TextBlock();
myTextBlock7.FontSize = 18;
myTextBlock7.HorizontalAlignment = HorizontalAlignment.Center;
myTextBlock7.Margin = new Thickness(0, 0, 0, 15);
myTextBlock7.Text = "StackPanel 2";
Button myButton4 = new Button();
myButton4.Margin = new Thickness(10, 0, 10, 0);
myButton4.Content = "Button 4";
Button myButton5 = new Button();
myButton5.Margin = new Thickness(10, 0, 10, 0);
myButton5.Content = "Button 5";
Button myButton6 = new Button();
myButton6.Margin = new Thickness(10, 0, 10, 0);
myButton6.Content = "Button 6";
TextBlock myTextBlock8 = new TextBlock();
myTextBlock8.HorizontalAlignment = HorizontalAlignment.Center;
myTextBlock8.Text = @"ColumnDefinition.Width = ""*""";
TextBlock myTextBlock9 = new TextBlock();
myTextBlock9.HorizontalAlignment = HorizontalAlignment.Center;
myTextBlock9.Text = @"StackPanel.HorizontalAlignment = ""Stretch""";
TextBlock myTextBlock10 = new TextBlock();
myTextBlock10.HorizontalAlignment = HorizontalAlignment.Center;
myTextBlock10.Text = @"StackPanel.VerticalAlignment = ""Top""";
TextBlock myTextBlock11 = new TextBlock();
```

```

myTextBlock11.HorizontalAlignment = HorizontalAlignment.Center;
myTextBlock11.Text = @"StackPanel.Orientation = ""Horizontal""";
TextBlock myTextBlock12 = new TextBlock();
myTextBlock12.HorizontalAlignment = HorizontalAlignment.Center;
myTextBlock12.Text = @"Button.Margin = ""10,0,10,0""";
myStackPanel2.Children.Add(myTextBlock7);
myStackPanel2.Children.Add(myButton4);
myStackPanel2.Children.Add(myButton5);
myStackPanel2.Children.Add(myButton6);
myStackPanel2.Children.Add(myTextBlock8);
myStackPanel2.Children.Add(myTextBlock9);
myStackPanel2.Children.Add(myTextBlock10);
myStackPanel2.Children.Add(myTextBlock11);
myStackPanel2.Children.Add(myTextBlock12);

// Add the final child StackPanel.
StackPanel myStackPanel3 = new StackPanel();
myStackPanel3.HorizontalAlignment = HorizontalAlignment.Left;
myStackPanel3.VerticalAlignment = VerticalAlignment.Top;
Grid.SetColumn(myStackPanel3, 2);
Grid.SetRow(myStackPanel3, 0);
TextBlock myTextBlock13 = new TextBlock();
myTextBlock13.FontSize = 18;
myTextBlock13.HorizontalAlignment = HorizontalAlignment.Center;
myTextBlock13.Margin = new Thickness(0, 0, 0, 15);
myTextBlock13.Text = "StackPanel 3";
Button myButton7 = new Button();
myButton7.Margin = new Thickness(10);
myButton7.Content = "Button 7";
Button myButton8 = new Button();
myButton8.Margin = new Thickness(10);
myButton8.Content = "Button 8";
Button myButton9 = new Button();
myButton9.Margin = new Thickness(10);
myButton9.Content = "Button 9";
TextBlock myTextBlock14 = new TextBlock();
myTextBlock14.Text = @"ColumnDefinition.Width = ""Auto""";
TextBlock myTextBlock15 = new TextBlock();
myTextBlock15.Text = @"StackPanel.HorizontalAlignment = ""Left""";
TextBlock myTextBlock16 = new TextBlock();
myTextBlock16.Text = @"StackPanel.VerticalAlignment = ""Top""";
TextBlock myTextBlock17 = new TextBlock();
myTextBlock17.Text = @"StackPanel.Orientation = ""Vertical""";
TextBlock myTextBlock18 = new TextBlock();
myTextBlock18.Text = @"Button.Margin = ""10""";
myStackPanel3.Children.Add(myTextBlock13);
myStackPanel3.Children.Add(myButton7);
myStackPanel3.Children.Add(myButton8);
myStackPanel3.Children.Add(myButton9);
myStackPanel3.Children.Add(myTextBlock14);
myStackPanel3.Children.Add(myTextBlock15);
myStackPanel3.Children.Add(myTextBlock16);
myStackPanel3.Children.Add(myTextBlock17);
myStackPanel3.Children.Add(myTextBlock18);

// Add child content to the parent Grid.
myGrid.Children.Add(myStackPanel1);
myGrid.Children.Add(myStackPanel2);
myGrid.Children.Add(myStackPanel3);

// Add the Grid as the lone child of the Border.
myBorder.Child = myGrid;

// Add the Border to the Window as Content and show the Window.
mainWindow.Content = myBorder;
mainWindow.Title = "Margin, Padding, and Alignment Sample";
mainWindow.Show();

```

```
Dim myBorder As New Border
myBorder.Background = Brushes.LightBlue
myBorder.BorderBrush = Brushes.Black
myBorder.BorderThickness = New Thickness(2)
myBorder.CornerRadius = New CornerRadius(45)
myBorder.Padding = New Thickness(25)

'Define the Grid.
Dim myGrid As New Grid
myGrid.Background = Brushes.White
myGrid.ShowGridLines = True

'Define the Columns.
Dim myColDef1 As New ColumnDefinition
myColDef1.Width = New GridLength(1, GridUnitType.Auto)
Dim myColDef2 As New ColumnDefinition
myColDef2.Width = New GridLength(1, GridUnitType.Star)
Dim myColDef3 As New ColumnDefinition
myColDef3.Width = New GridLength(1, GridUnitType.Auto)

'Add the ColumnDefinitions to the Grid
myGrid.ColumnDefinitions.Add(myColDef1)
myGrid.ColumnDefinitions.Add(myColDef2)
myGrid.ColumnDefinitions.Add(myColDef3)

'Add the first child StackPanel.
Dim myStackPanel As New StackPanel
myStackPanel.HorizontalAlignment = System.Windows.HorizontalAlignment.Left
myStackPanel.VerticalAlignment = System.Windows.VerticalAlignment.Top
Grid.SetColumn(myStackPanel, 0)
Grid.SetRow(myStackPanel, 0)
Dim myTextBlock1 As New TextBlock
myTextBlock1.FontSize = 18
myTextBlock1.HorizontalAlignment = System.Windows.HorizontalAlignment.Center
myTextBlock1.Margin = New Thickness(0, 0, 0, 15)
myTextBlock1.Text = "StackPanel 1"

Dim myButton1 As New Button
myButton1.Margin = New Thickness(0, 10, 0, 10)
myButton1.Content = "Button 1"
Dim myButton2 As New Button
myButton2.Margin = New Thickness(0, 10, 0, 10)
myButton2.Content = "Button 2"
Dim myButton3 As New Button
myButton3.Margin = New Thickness(0, 10, 0, 10)

Dim myTextBlock2 As New TextBlock
myTextBlock2.Text = "ColumnDefinition.Width = ""Auto"""
Dim myTextBlock3 As New TextBlock
myTextBlock3.Text = "StackPanel.HorizontalAlignment = ""Left"""
Dim myTextBlock4 As New TextBlock
myTextBlock4.Text = "StackPanel.VerticalAlignment = ""Top"""
Dim myTextBlock5 As New TextBlock
myTextBlock5.Text = "StackPanel.Orientation = ""Vertical"""
Dim myTextBlock6 As New TextBlock
myTextBlock6.Text = "Button.Margin = ""1,10,0,10"""
myStackPanel.Children.Add(myTextBlock1)
myStackPanel.Children.Add(myButton1)
myStackPanel.Children.Add(myButton2)
myStackPanel.Children.Add(myButton3)
myStackPanel.Children.Add(myTextBlock2)
myStackPanel.Children.Add(myTextBlock3)
myStackPanel.Children.Add(myTextBlock4)
myStackPanel.Children.Add(myTextBlock5)
myStackPanel.Children.Add(myTextBlock6)

'Add the second child StackPanel.
```

```

Dim myStackPanel2 As New StackPanel
myStackPanel2.HorizontalAlignment = System.Windows.HorizontalAlignment.Stretch
myStackPanel2.VerticalAlignment = System.Windows.VerticalAlignment.Top
myStackPanel2.Orientation = Orientation.Vertical
Grid.SetColumn(myStackPanel2, 1)
Grid.SetRow(myStackPanel2, 0)
Dim myTextBlock7 As New TextBlock
myTextBlock7.FontSize = 18
myTextBlock7.HorizontalAlignment = System.Windows.HorizontalAlignment.Center
myTextBlock7.Margin = New Thickness(0, 0, 0, 15)
myTextBlock7.Text = "StackPanel 2"
Dim myButton4 As New Button
myButton4.Margin = New Thickness(10, 0, 10, 0)
myButton4.Content = "Button 4"
Dim myButton5 As New Button
myButton5.Margin = New Thickness(10, 0, 10, 0)
myButton5.Content = "Button 5"
Dim myButton6 As New Button
myButton6.Margin = New Thickness(10, 0, 10, 0)
myButton6.Content = "Button 6"
Dim myTextBlock8 As New TextBlock
myTextBlock8.HorizontalAlignment = System.Windows.HorizontalAlignment.Center
myTextBlock8.Text = "ColumnDefinition.Width = ""*"""
Dim myTextBlock9 As New TextBlock
myTextBlock9.HorizontalAlignment = System.Windows.HorizontalAlignment.Center
myTextBlock9.Text = "StackPanel.HorizontalAlignment = ""Stretch"""
Dim myTextBlock10 As New TextBlock
myTextBlock10.HorizontalAlignment = System.Windows.HorizontalAlignment.Center
myTextBlock10.Text = "StackPanel.VerticalAlignment = ""Top"""
Dim myTextBlock11 As New TextBlock
myTextBlock11.HorizontalAlignment = System.Windows.HorizontalAlignment.Center
myTextBlock11.Text = "StackPanel.Orientation = ""Horizontal"""
Dim myTextBlock12 As New TextBlock
myTextBlock12.HorizontalAlignment = System.Windows.HorizontalAlignment.Center
myTextBlock12.Text = "Button.Margin = ""10,0,10,0"""
myStackPanel2.Children.Add(myTextBlock7)
myStackPanel2.Children.Add(myButton4)
myStackPanel2.Children.Add(myButton5)
myStackPanel2.Children.Add(myButton6)
myStackPanel2.Children.Add(myTextBlock8)
myStackPanel2.Children.Add(myTextBlock9)
myStackPanel2.Children.Add(myTextBlock10)
myStackPanel2.Children.Add(myTextBlock11)
myStackPanel2.Children.Add(myTextBlock12)

'Add the final child StackPanel.
Dim myStackPanel3 As New StackPanel
myStackPanel3.HorizontalAlignment = System.Windows.HorizontalAlignment.Left
myStackPanel3.VerticalAlignment = System.Windows.VerticalAlignment.Top
Grid.SetColumn(myStackPanel3, 2)
Grid.SetRow(myStackPanel3, 0)
Dim myTextBlock13 As New TextBlock
myTextBlock13.FontSize = 18
myTextBlock13.HorizontalAlignment = System.Windows.HorizontalAlignment.Center
myTextBlock13.Margin = New Thickness(0, 0, 0, 15)
myTextBlock13.Text = "StackPanel 3"

Dim myButton7 As New Button
myButton7.Margin = New Thickness(10)
myButton7.Content = "Button 7"
Dim myButton8 As New Button
myButton8.Margin = New Thickness(10)
myButton8.Content = "Button 8"
Dim myButton9 As New Button
myButton9.Margin = New Thickness(10)
myButton9.Content = "Button 9"
Dim myTextBlock14 As New TextBlock
myTextBlock14.Text = "ColumnDefinition.Width = ""Auto"""
Dim myTextBlock15 As New TextBlock

```

```
myTextBlock15.Text = "StackPanel.HorizontalAlignment = ""Left"""
Dim myTextBlock16 As New TextBlock
myTextBlock16.Text = "StackPanel.VerticalAlignment = ""Top"""
Dim myTextBlock17 As New TextBlock
myTextBlock17.Text = "StackPanel.Orientation = ""Vertical"""
Dim myTextBlock18 As New TextBlock
myTextBlock18.Text = "Button.Margin = ""10"""
myStackPanel3.Children.Add(myTextBlock13)
myStackPanel3.Children.Add(myButton7)
myStackPanel3.Children.Add(myButton8)
myStackPanel3.Children.Add(myButton9)
myStackPanel3.Children.Add(myTextBlock14)
myStackPanel3.Children.Add(myTextBlock15)
myStackPanel3.Children.Add(myTextBlock16)
myStackPanel3.Children.Add(myTextBlock17)
myStackPanel3.Children.Add(myTextBlock18)

'Add child content to the parent Grid.
myGrid.Children.Add(myStackPanel)
myGrid.Children.Add(myStackPanel2)
myGrid.Children.Add(myStackPanel3)

'Add the Grid as the lone child of the Border.
myBorder.Child = myGrid
```

```

<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" WindowTitle="Margins, Padding and
Alignment Sample">
    <Border Background="LightBlue"
        BorderBrush="Black"
        BorderThickness="2"
        CornerRadius="45"
        Padding="25">
        <Grid Background="White" ShowGridLines="True">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto"/>
                <ColumnDefinition Width="*"/>
                <ColumnDefinition Width="Auto"/>
            </Grid.ColumnDefinitions>

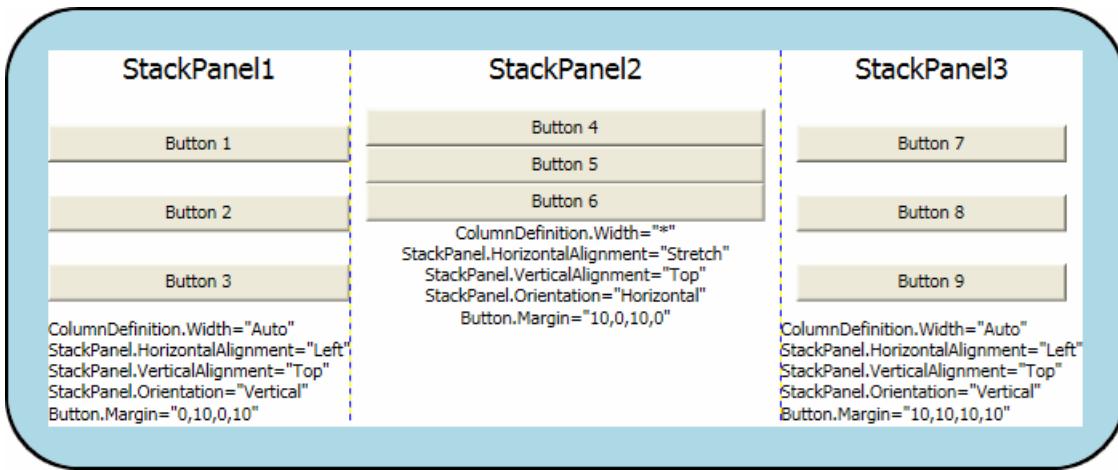
            <StackPanel Grid.Column="0" Grid.Row="0" HorizontalAlignment="Left" Name="StackPanel1"
VerticalAlignment="Top">
                <TextBlock FontSize="18" HorizontalAlignment="Center" Margin="0,0,0,15">StackPanel1</TextBlock>
                <Button Margin="0,10,0,10">Button 1</Button>
                <Button Margin="0,10,0,10">Button 2</Button>
                <Button Margin="0,10,0,10">Button 3</Button>
                <TextBlock>ColumnDefinition.Width="Auto"</TextBlock>
                <TextBlock>StackPanel.HorizontalAlignment="Left"</TextBlock>
                <TextBlock>StackPanel.VerticalAlignment="Top"</TextBlock>
                <TextBlock>StackPanel.Orientation="Vertical"</TextBlock>
                <TextBlock>Button.Margin="0,10,0,10"</TextBlock>
            </StackPanel>

            <StackPanel Grid.Column="1" Grid.Row="0" HorizontalAlignment="Stretch" Name="StackPanel2"
VerticalAlignment="Top" Orientation="Vertical">
                <TextBlock FontSize="18" HorizontalAlignment="Center" Margin="0,0,0,15">StackPanel2</TextBlock>
                <Button Margin="10,0,10,0">Button 4</Button>
                <Button Margin="10,0,10,0">Button 5</Button>
                <Button Margin="10,0,10,0">Button 6</Button>
                <TextBlock HorizontalAlignment="Center">ColumnDefinition.Width="*"/</TextBlock>
                <TextBlock HorizontalAlignment="Center">StackPanel.HorizontalAlignment="Stretch"</TextBlock>
                <TextBlock HorizontalAlignment="Center">StackPanel.VerticalAlignment="Top"</TextBlock>
                <TextBlock HorizontalAlignment="Center">StackPanel.Orientation="Horizontal"</TextBlock>
                <TextBlock HorizontalAlignment="Center">Button.Margin="10,0,10,0"</TextBlock>
            </StackPanel>

            <StackPanel Grid.Column="2" Grid.Row="0" HorizontalAlignment="Left" Name="StackPanel3"
VerticalAlignment="Top">
                <TextBlock FontSize="18" HorizontalAlignment="Center" Margin="0,0,0,15">StackPanel3</TextBlock>
                <Button Margin="10">Button 7</Button>
                <Button Margin="10">Button 8</Button>
                <Button Margin="10">Button 9</Button>
                <TextBlock>ColumnDefinition.Width="Auto"</TextBlock>
                <TextBlock>StackPanel.HorizontalAlignment="Left"</TextBlock>
                <TextBlock>StackPanel.VerticalAlignment="Top"</TextBlock>
                <TextBlock>StackPanel.Orientation="Vertical"</TextBlock>
                <TextBlock>Button.Margin="10"</TextBlock>
            </StackPanel>
        </Grid>
    </Border>
</Page>

```

Dopo la compilazione, l'applicazione precedente genera un'Interfaccia utente di aspetto simile al seguente. Gli effetti dei vari valori delle proprietà sono evidenti nella spaziatura tra gli elementi e i valori di proprietà significativi per gli elementi di ogni **TextBlock** colonna vengono visualizzati all'interno di elementi.



Argomenti successivi

Le proprietà di posizionamento definite [FrameworkElement](#) dalla classe consentono un controllo accurato del WPF posizionamento degli elementi all'interno delle applicazioni. Quelle descritte qui sono dunque le diverse tecniche che permettono di posizionare in modo più efficace gli elementi usando WPF.

Sono inoltre disponibili ulteriori risorse in cui il layout di WPF viene descritto in modo più particolareggiato. Nell'argomento [Panoramica dei pannelli](#) sono disponibili informazioni più dettagliate [Panel](#) sui vari elementi. Argomento [della procedura dettagliata: La prima applicazione desktop WPF](#) introduce tecniche avanzate che usano elementi di layout per posizionare i componenti e associare le azioni alle origini dati.

Vedere anche

- [FrameworkElement](#)
- [HorizontalAlignment](#)
- [VerticalAlignment](#)
- [Margin](#)
- [Cenni preliminari sugli elementi Panel](#)
- [Layout](#)
- [Esempio di raccolte di layout WPF](#)

Procedure relative agli elementi di base

23/10/2019 • 2 minutes to read • [Edit Online](#)

Negli argomenti di questa sezione viene descritto come utilizzare i quattro elementi di base di WPF: [UIElement](#), [ContentElement](#), [FrameworkElement](#), e [FrameworkContentElement](#).

In questa sezione

- [Rendere trasparente o semitrasparente un oggetto UIElement](#)
- [Aggiungere un'animazione alle dimensioni di un oggetto FrameworkElement](#)
- [Determinare se un oggetto Freezable è bloccato](#)
- [Gestire un evento caricato](#)
- [Impostare i margini di elementi e controlli](#)
- [Impostare la proprietà di sola lettura per un oggetto Freezable](#)
- [Ottenere una copia scrivibile di un oggetto Freezable di sola lettura](#)
- [Capovolgere un oggetto UIElement orizzontalmente o verticalmente](#)
- [Usare un oggetto ThicknessConverter](#)
- [Gestire l'evento ContextMenuOpening](#)

Riferimenti

- [UIElement](#)
- [ContentElement](#)
- [FrameworkElement](#)
- [FrameworkContentElement](#)

Sezioni correlate

- [Elementi di base](#)

Procedura: Rendere trasparente o semitrasparente un oggetto UIElement

23/10/2019 • 3 minutes to read • [Edit Online](#)

In questo esempio viene illustrato come effettuare una [UIElement](#) trasparente o semitrasparente. Per rendere un elemento trasparente o semitrasparente, impostare il [Opacity](#) proprietà. Un valore pari a `0.0` l'elemento viene reso completamente trasparente, mentre un valore di `1.0` l'elemento viene reso completamente opaco. Un valore di `0.5` rende l'elemento 50% e così via. Un elemento [Opacity](#) è impostata su `1.0` per impostazione predefinita.

Esempio

L'esempio seguente imposta la [Opacity](#) di un pulsante su `0.25`, rendendo opaco l'oggetto e relativo contenuto (in questo caso, il testo del pulsante) al 25%.

```
<!-- Both the button and its text are made 25% opaque. -->
<Button Opacity="0.25">A Button</Button>
```

```
//
// Both the button and its text are made 25% opaque.
//
Button myTwentyFivePercentOpaqueButton = new Button();
myTwentyFivePercentOpaqueButton.Opacity = new Double();
myTwentyFivePercentOpaqueButton.Opacity = 0.25;
myTwentyFivePercentOpaqueButton.Content = "A Button";
```

Se il contenuto di un elemento dispone di propri [Opacity](#) le impostazioni, tali valori vengono moltiplicate per gli elementi che lo contiene [Opacity](#).

L'esempio seguente imposta un pulsante [Opacity](#) al `0.25` e il [Opacity](#) di un [Image](#) controllo contenuto all'interno del pulsante su `0.5`. Di conseguenza, l'immagine viene visualizzata 12,5% opaco: $0.25 * 0.5 = 0.125$.

```
<!-- The image contained within this button has an effective
     opacity of 0.125 (0.25 * 0.5 = 0.125). -->
<Button Opacity="0.25">
    <StackPanel Orientation="Horizontal">
        <TextBlock VerticalAlignment="Center" Margin="10">A Button</TextBlock>
        <Image Source="sampleImages\berries.jpg" Width="50" Height="50"
              Opacity="0.5"/>
    </StackPanel>
</Button>
```

```

//  

// The image contained within this button has an  

// effective opacity of 0.125 (0.25*0.5 = 0.125);  

//  

Button myImageButton = new Button();  

myImageButton.Opacity = new Double();  

myImageButton.Opacity = 0.25;  

StackPanel myImageStackPanel = new StackPanel();  

myImageStackPanel.Orientation = Orientation.Horizontal;  

TextBlock myTextBlock = new TextBlock();  

myTextBlock.VerticalAlignment = VerticalAlignment.Center;  

myTextBlock.Margin = new Thickness(10);  

myTextBlock.Text = "A Button";  

myImageStackPanel.Children.Add(myTextBlock);  

Image myImage = new Image();  

BitmapImage myBitmapImage = new BitmapImage();  

myBitmapImage.BeginInit();  

myBitmapImage.UriSource = new Uri("sampleImages/berries.jpg", UriKind.Relative);  

myBitmapImage.EndInit();  

myImage.Source = myBitmapImage;  

ImageBrush myImageBrush = new ImageBrush(myBitmapImage);  

myImage.Width = 50;  

myImage.Height = 50;  

myImage.Opacity = 0.5;  

myImageStackPanel.Children.Add(myImage);  

myImageButton.Content = myImageStackPanel;

```

Un altro modo per controllare l'opacità di un elemento consiste nell'impostare l'opacità del [Brush](#) che disegna l'elemento. Questo approccio consente di modificare in modo selettivo l'opacità delle porzioni di un elemento e offre vantaggi nelle prestazioni rispetto all'uso dell'elemento [Opacity](#) proprietà. L'esempio seguente imposta la [Opacity](#) di un [SolidColorBrush](#) utilizzato per disegnare il pulsante [Background](#) è impostata su [0.25](#). Di conseguenza, lo sfondo del pennello è opaca al 25%, ma il relativo contenuto (testo del pulsante) rimane opache al 100%.

```

<!-- This button's background is made 25% opaque, but its  

    text remains 100% opaque. -->  

<Button>  

    <Button.Background>  

        <SolidColorBrush Color="Gray" Opacity="0.25" />  

    </Button.Background>  

    A Button  

</Button>

```

```

//  

// This button's background is made 25% opaque,  

// but its text remains 100% opaque.  

//  

Button myOpaqueTextButton = new Button();  

SolidColorBrush mySolidColorBrush = new SolidColorBrush(Colors.Gray);  

mySolidColorBrush.Opacity = 0.25;  

myOpaqueTextButton.Background = mySolidColorBrush;  

myOpaqueTextButton.Content = "A Button";

```

È inoltre possibile controllare l'opacità di colori singoli all'interno di un pennello. Per altre informazioni sui colori e pennelli, vedere [disegno con colori a tinta unita e sfumature Panoramica](#). Per un esempio che illustra come animare l'opacità di un elemento, vedere [animare l'opacità di un elemento o un pennello](#).

Procedura: Aggiungere un'animazione alle dimensioni di un oggetto FrameworkElement

23/10/2019 • 2 minutes to read • [Edit Online](#)

Per animare la proprietà size di un [FrameworkElement](#), è possibile aggiungere un'animazione relativa [Width](#) e [Height](#) delle proprietà o utilizzare un oggetto animato [ScaleTransform](#).

Nell'esempio seguente anima le dimensioni dei due pulsanti utilizzando questi due approcci. Tramite l'animazione viene ridimensionato un pulsante relativi [Width](#) proprietà e un altro ridimensionamento aggiungendo un'animazione una [ScaleTransform](#) applicati a relativo [RenderTransform](#) proprietà. Ogni pulsante contiene testo. Inizialmente, il testo viene visualizzato lo stesso in entrambi i pulsanti, ma come vengono ridimensionati i pulsanti, il testo del secondo pulsante diventa distorto.

Esempio

```

<!-- AnimatingSizeExample.xaml
This example shows two ways of animating the size
of a framework element. -->
<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="Microsoft.Samples.Animation.AnimatingSizeExample"
    WindowTitle="Animating Size Example">
    <Canvas Width="650" Height="400">

        <Button Name="AnimatedWidthButton"
            Canvas.Left="20" Canvas.Top="20"
            Width="200" Height="150"
            BorderBrush="Red" BorderThickness="5">
            Click Me
            <Button.Triggers>

                <!-- Animate the button's Width property. -->
                <EventTrigger RoutedEvent="Button.Loaded">
                    <BeginStoryboard>
                        <Storyboard>
                            <DoubleAnimation
                                Storyboard.TargetName="AnimatedWidthButton"
                                Storyboard.TargetProperty="(Button.Width)"
                                To="500" Duration="0:0:10" AutoReverse="True"
                                RepeatBehavior="Forever" />
                        </Storyboard>
                    </BeginStoryboard>
                </EventTrigger>
            </Button.Triggers>
        </Button>

        <Button
            Canvas.Left="20" Canvas.Top="200"
            Width="200" Height="150"
            BorderBrush="Black" BorderThickness="3">
            Click Me
            <Button.RenderTransform>
                <ScaleTransform x:Name="MyAnimatedScaleTransform"
                    ScaleX="1" ScaleY="1" />
            </Button.RenderTransform>
            <Button.Triggers>

                <!-- Animate the ScaleX property of a ScaleTransform
                    applied to the button. -->
                <EventTrigger RoutedEvent="Button.Loaded">
                    <BeginStoryboard>
                        <Storyboard>
                            <DoubleAnimation
                                Storyboard.TargetName="MyAnimatedScaleTransform"
                                Storyboard.TargetProperty="(ScaleTransform.ScaleX)"
                                To="3.0" Duration="0:0:10" AutoReverse="True"
                                RepeatBehavior="Forever" />
                        </Storyboard>
                    </BeginStoryboard>
                </EventTrigger>
            </Button.Triggers>
        </Button>
    </Canvas>
</Page>

```

Quando si trasforma un elemento, l'intero elemento e il relativo contenuto viene trasformate. Quando si modifica direttamente la dimensione di un elemento, come nel caso del primo pulsante, il contenuto dell'elemento non viene ridimensionato, a meno che le dimensioni e posizione dipendono dalle dimensioni del relativo elemento padre.

Animazione delle dimensioni di un elemento applicando una trasformazione animata in relativi [RenderTransform](#) proprietà offre prestazioni migliori rispetto all'animazione relativi [Width](#) e [Height](#) direttamente, poiché il [RenderTransform](#) proprietà non attiva un passaggio di layout.

Per altre informazioni sulle proprietà di animazione, vedere la [Cenni preliminari sull'animazione](#). Per altre informazioni sulle trasformazioni, vedere la [Cenni preliminari sulle trasformazioni](#).

Procedura: Determinare se un oggetto Freezable è bloccato

23/10/2019 • 2 minutes to read • [Edit Online](#)

In questo esempio viene illustrato come determinare se un [Freezable](#) oggetto è stato bloccato. Se si prova a modificare un oggetto bloccato [Freezable](#) dell'oggetto, genera un [InvalidOperationException](#). Per evitare la generazione di questa eccezione, usare il [IsFrozen](#) proprietà del [Freezable](#) oggetto per determinare se è bloccato.

Esempio

Nell'esempio seguente si blocca una [SolidColorBrush](#) e quindi verifica l'evento utilizzando il [IsFrozen](#) proprietà per determinare se è bloccato.

```
Button myButton = new Button();
SolidColorBrush myBrush = new SolidColorBrush(Colors.Yellow);

if (myBrush.CanFreeze)
{
    // Makes the brush unmodifiable.
    myBrush.Freeze();
}

myButton.Background = myBrush;

if (myBrush.IsFrozen) // Evaluates to true.
{
    // If the brush is frozen, create a clone and
    // modify the clone.
    SolidColorBrush myBrushClone = myBrush.Clone();
    myBrushClone.Color = Colors.Red;
    myButton.Background = myBrushClone;
}
else
{
    // If the brush is not frozen,
    // it can be modified directly.
    myBrush.Color = Colors.Red;
}
```

```
Dim myButton As New Button()
Dim myBrush As New SolidColorBrush(Colors.Yellow)

If myBrush.CanFreeze Then
    ' Makes the brush unmodifiable.
    myBrush.Freeze()
End If

myButton.Background = myBrush

If myBrush.IsFrozen Then ' Evaluates to true.
    ' If the brush is frozen, create a clone and
    ' modify the clone.
    Dim myBrushClone As SolidColorBrush = myBrush.Clone()
    myBrushClone.Color = Colors.Red
    myButton.Background = myBrushClone
Else
    ' If the brush is not frozen,
    ' it can be modified directly.
    myBrush.Color = Colors.Red
End If
```

Per altre informazioni sulle [Freezable](#) oggetti, vedere la [Cenni preliminari sugli oggetti Freezable](#).

Vedere anche

- [Freezable](#)
- [IsFrozen](#)
- [Cenni preliminari sugli oggetti Freezable](#)
- [Procedure relative alle proprietà](#)

Procedura: Gestire un evento caricato

23/10/2019 • 2 minutes to read • [Edit Online](#)

In questo esempio viene illustrato come gestire il `FrameworkElement.Loaded` evento e uno scenario adatto per la gestione di tale evento. Il gestore crea un `Button` quando la pagina viene caricata.

Esempio

L'esempio seguente usa Extensible Application Markup Language (XAML) insieme a un file code-behind.

```
<StackPanel  
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
    x:Class="SDKSample.FELoaded"  
    Loaded="OnLoad"  
    Name="root"  
>  
</StackPanel>
```

```
void OnLoad(object sender, RoutedEventArgs e)  
{  
    Button b1 = new Button();  
    b1.Content = "New Button";  
    root.Children.Add(b1);  
    b1.Height = 25;  
    b1.Width = 200;  
    b1.HorizontalAlignment = HorizontalAlignment.Left;  
}
```

```
Private Sub OnLoad(ByVal sender As Object, ByVal e As RoutedEventArgs)  
    Dim b1 As Button = New Button()  
    b1.Content = "New Button"  
    root.Children.Add(b1)  
    b1.Height = 25  
    b1.Width = 200  
    b1.HorizontalAlignment = HorizontalAlignment.Left  
End Sub
```

Vedere anche

- [FrameworkElement](#)
- [Eventi di durata degli oggetti](#)
- [Cenni preliminari sugli eventi indirizzati](#)
- [Procedure relative alle proprietà](#)

Procedura: Impostare i margini di elementi e controlli

23/10/2019 • 2 minutes to read • [Edit Online](#)

In questo esempio viene descritto come impostare il [Margin](#) proprietà, modificando i valori di proprietà esistenti per il margine nel code-behind. Il [Margin](#) è una proprietà della [FrameworkElement](#) elemento di base e pertanto viene ereditata da un'ampia gamma di controlli e altri elementi.

In questo esempio è scritto in Extensible Application Markup Language (XAML), con un code-behind del file che il XAML fa riferimento a. Il code-behind viene visualizzato sia in un C# e una versione di Microsoft Visual Basic.

Esempio

```
<Button Click="OnClick" Margin="10" Name="btn1">  
Click To See Change!!</Button>
```

```
void OnClick(object sender, RoutedEventArgs e)  
{  
    // Get the current value of the property.  
    Thickness marginThickness = btn1.Margin;  
    // If the current leftlength value of margin is set to 10 then change it to a new value.  
    // Otherwise change it back to 10.  
    if(marginThickness.Left == 10)  
    {  
        btn1.Margin = new Thickness(60);  
    } else {  
        btn1.Margin = new Thickness(10);  
    }  
}
```

```
Private Sub OnClick(ByVal sender As Object, ByVal e As RoutedEventArgs)  
  
    ' Get the current value of the property.  
    Dim marginThickness As Thickness  
    marginThickness = btn1.Margin  
    ' If the current leftlength value of margin is set to 10 then change it to a new value.  
    ' Otherwise change it back to 10.  
    If marginThickness.Left = 10 Then  
        btn1.Margin = New Thickness(60)  
    Else  
        btn1.Margin = New Thickness(10)  
    End If  
End Sub
```

Procedura: impostare la proprietà di sola lettura per un oggetto Freezable

04/11/2019 • 2 minutes to read • [Edit Online](#)

In questo esempio viene illustrato come creare un [Freezable](#) di sola lettura chiamando il relativo metodo [Freeze](#).

Non è possibile bloccare un oggetto [Freezable](#) se una delle condizioni seguenti è `true` sull'oggetto:

- Dispone di proprietà animate o con associazione a dati.
- Dispone di proprietà impostate da una risorsa dinamica. Per ulteriori informazioni sulle risorse dinamiche, vedere le [risorse XAML](#).
- Contiene [Freezable](#) oggetti secondari che non possono essere bloccati.

Se queste condizioni vengono `false` per l'oggetto [Freezable](#) e non si intende modificarlo, provare a bloccarlo per ottenere vantaggi in termini di prestazioni.

Esempio

Nell'esempio seguente viene bloccato un [SolidColorBrush](#), che è un tipo di [Freezable](#) oggetto.

```
Button myButton = new Button();
SolidColorBrush myBrush = new SolidColorBrush(Colors.Yellow);

if (myBrush.CanFreeze)
{
    // Makes the brush unmodifiable.
    myBrush.Freeze();
}

myButton.Background = myBrush;
```

```
Dim myButton As New Button()
Dim myBrush As New SolidColorBrush(Colors.Yellow)

If myBrush.CanFreeze Then
    ' Makes the brush unmodifiable.
    myBrush.Freeze()
End If

myButton.Background = myBrush
```

Per ulteriori informazioni sugli oggetti [Freezable](#), vedere [Cenni preliminari sugli oggetti Freezable](#).

Vedere anche

- [Freezable](#)
- [CanFreeze](#)
- [Freeze](#)
- [Cenni preliminari sugli oggetti Freezable](#)
- [Procedure relative alle proprietà](#)

Procedura: Ottenere una copia scrivibile di un oggetto Freezable di sola lettura

23/10/2019 • 2 minutes to read • [Edit Online](#)

Questo esempio illustra come usare il [Clone](#) metodo per creare una copia scrivibile di sola lettura [Freezable](#).

Dopo un [Freezable](#) oggetto è contrassegnato come di sola lettura ("bloccato"), è possibile modificarla. Tuttavia, è possibile usare il [Clone](#) metodo per creare un clone modificabile dell'oggetto bloccato.

Esempio

L'esempio seguente crea un clone modificabile di un oggetto bloccato [SolidColorBrush](#) oggetto.

```
Button myButton = new Button();
SolidColorBrush myBrush = new SolidColorBrush(Colors.Yellow);

// Freezing a Freezable before it provides
// performance improvements if you don't
// intend on modifying it.
if (myBrush.CanFreeze)
{
    // Makes the brush unmodifiable.
    myBrush.Freeze();
}

myButton.Background = myBrush;

// If you need to modify a frozen brush,
// the Clone method can be used to
// create a modifiable copy.
SolidColorBrush myBrushClone = myBrush.Clone();

// Changing myBrushClone does not change
// the color of myButton, because its
// background is still set by myBrush.
myBrushClone.Color = Colors.Red;

// Replacing myBrush with myBrushClone
// makes the button change to red.
myButton.Background = myBrushClone;
```

```

Dim myButton As New Button()
Dim myBrush As New SolidColorBrush(Colors.Yellow)

' Freezing a Freezable before it provides
' performance improvements if you don't
' intend on modifying it.
If myBrush.CanFreeze Then
    ' Makes the brush unmodifiable.
    myBrush.Freeze()
End If

myButton.Background = myBrush

' If you need to modify a frozen brush,
' the Clone method can be used to
' create a modifiable copy.
Dim myBrushClone As SolidColorBrush = myBrush.Clone()

' Changing myBrushClone does not change
' the color of myButton, because its
' background is still set by myBrush.
myBrushClone.Color = Colors.Red

' Replacing myBrush with myBrushClone
' makes the button change to red.
myButton.Background = myBrushClone

```

Per altre informazioni sulle [Freezable](#) oggetti, vedere la [Cenni preliminari sugli oggetti Freezable](#).

Vedere anche

- [Freezable](#)
- [CloneCurrentValue](#)
- [Cenni preliminari sugli oggetti Freezable](#)
- [Procedure relative alle proprietà](#)

Procedura: Capovolgere un oggetto UIElement orizzontalmente o verticalmente

23/10/2019 • 2 minutes to read • [Edit Online](#)

In questo esempio viene illustrato come utilizzare un [ScaleTransform](#) capovolgere un [UIElement](#) orizzontalmente o verticalmente. In questo esempio, un [Button](#) controllo (un tipo di [UIElement](#)) viene capovolto applicando una [ScaleTransform](#) al relativo [RenderTransform](#) proprietà.

Esempio

La figura seguente mostra il pulsante per capovolgere.



UIElement da capovolgere

Di seguito viene illustrato il codice che crea il pulsante.

```
<Button Content="Flip me!" Padding="5">  
</Button>
```

Esempio

Per capovolgere orizzontalmente il pulsante, creare un [ScaleTransform](#) e impostare il [ScaleX](#) proprietà su -1. Si applicano i [ScaleTransform](#) del pulsante [RenderTransform](#) proprietà.

```
<Button Content="Flip me!" Padding="5">  
  <Button.RenderTransform>  
    <ScaleTransform ScaleX="-1" />  
  </Button.RenderTransform>  
</Button>
```



Il pulsante dopo l'applicazione ScaleTransform

Esempio

Come può notare dalla figura precedente, il pulsante è stato capovolto, ma è anche stato spostato. Ciò avviene

perché il pulsante è stato capovolto dall'angolo superiore sinistro. Per invertire il pulsante posto, si desidera applicare il [ScaleTransform](#) al relativo centro, non all'angolo. Un modo semplice per applicare la [ScaleTransform](#) ai pulsanti center consiste nell'impostare il pulsante [RenderTransformOrigin](#) proprietà su 0,5, 0,5.

```
<Button Content="Flip me!" Padding="5"
    RenderTransformOrigin="0.5,0.5">
    <Button.RenderTransform>
        <ScaleTransform ScaleX="-1" />
    </Button.RenderTransform>
</Button>
```



Il pulsante con RenderTransformOrigin del valore pari a 0,5, 0,5

Esempio

Per capovolgere verticalmente il pulsante, impostare il [ScaleTransform](#) dell'oggetto [ScaleY](#) proprietà invece della relativa [ScaleX](#) proprietà.

```
<Button Content="Flip me!" Padding="5"
    RenderTransformOrigin="0.5,0.5">
    <Button.RenderTransform>
        <ScaleTransform ScaleY="-1" />
    </Button.RenderTransform>
</Button>
```



Pulsante capovolto verticalmente

Vedere anche

- [Cenni preliminari sulle trasformazioni](#)

Procedura: Usare un oggetto ThicknessConverter

23/10/2019 • 2 minutes to read • [Edit Online](#)

Esempio

In questo esempio viene illustrato come creare un'istanza di [ThicknessConverter](#) e usarlo per modificare lo spessore di un bordo.

L'esempio definisce un metodo personalizzato denominato `changeThickness`; questo metodo converte prima di tutto il contenuto di un [ListBoxItem](#), come definito in un oggetto separato Extensible Application Markup Language (XAML) file, a un'istanza di [Thickness](#); lo converte in un secondo momento il contenuto in un [String](#). Questo metodo passa il [ListBoxItem](#) a un [ThicknessConverter](#) oggetto, che converte le [Content](#) di un [ListBoxItem](#) a un'istanza di [Thickness](#). Questo valore viene quindi passato nuovamente come valore dei [BorderThickness](#) proprietà del [Border](#).

Questo esempio non viene eseguito.

```
private void changeThickness(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem li = ((sender as ListBox).SelectedItem as ListBoxItem);
    ThicknessConverter myThicknessConverter = new ThicknessConverter();
    Thickness th1 = (Thickness)myThicknessConverter.ConvertFromString(li.Content.ToString());
    border1.BorderThickness = th1;
    bThickness.Text = "Border.BorderThickness =" + li.Content.ToString();
}
```

```
Private Sub changeThickness(ByVal sender As Object, ByVal args As SelectionChangedEventArgs)

    Dim li As ListBoxItem = CType(CType(sender, ListBox).SelectedItem, ListBoxItem)
    Dim myThicknessConverter As System.Windows.ThicknessConverter = New System.Windows.ThicknessConverter()
    Dim th1 As Thickness = CType(myThicknessConverter.ConvertFromString(li.Content.ToString()), Thickness)
    border1.BorderThickness = th1
    bThickness.Text = "Border.BorderThickness =" + li.Content.ToString()
End Sub
```

Vedere anche

- [Thickness](#)
- [ThicknessConverter](#)
- [Border](#)
- [Procedura: Modificare la proprietà Margin](#)
- [Procedura: Convertire un oggetto ListBoxItem in un nuovo tipo di dati](#)
- [Cenni preliminari sugli elementi Panel](#)

Procedura: Gestire l'evento ContextMenuOpening

23/10/2019 • 10 minutes to read • [Edit Online](#)

Il [ContextMenuOpening](#) evento può essere gestito in un'applicazione per modificare un esistente prima di menu di scelta rapida per visualizzare o eliminare il menu che verrà altrimenti visualizzato impostando il [Handled](#) proprietà `true` nei dati dell'evento. La ragione più comune per l'impostazione [Handled](#) al `true` dei dati nell'evento consiste nella sostituzione menu interamente con un nuovo [ContextMenu](#) dell'oggetto, che a volte richiede l'annullamento dell'operazione e l'avvio di un nuovo open. Se si scrivono i gestori per il [ContextMenuOpening](#) evento, è necessario essere consapevoli dei problemi di temporizzazione tra un [ContextMenu](#) controllo e il servizio che è responsabile dell'apertura e il posizionamento di menu di scelta rapida per i controlli in generale. In questo argomento vengono illustrate alcune delle tecniche di codice per scenari di apertura dei vari menu di scelta rapida e viene illustrato un caso in cui il problema di temporizzazione entra in gioco.

Esistono diversi scenari per la gestione di [ContextMenuOpening](#) evento:

- Modificare le voci di menu prima della visualizzazione.
- Sostituire l'intero menu prima della visualizzazione.
- Completamente l'eliminazione di qualsiasi menu di scelta rapida esistente e non visualizzare alcun menu di scelta rapida.

Esempio

Modificare le voci di Menu prima della visualizzazione

Modificando le voci di menu esistente è piuttosto semplice e probabilmente lo scenario più comune. È possibile farlo per poter aggiungere o sottrarre le opzioni di menu di scelta rapida in risposta alle informazioni sullo stato corrente dell'applicazione o informazioni sullo stato determinato che sono disponibile come una proprietà dell'oggetto in cui viene richiesto il menu di scelta rapida.

La tecnica generale consiste nell'ottenere l'origine dell'evento, vale a dire il controllo specifico acquisendone, e Ottieni la [ContextMenu](#) proprietà da quest'ultimo. In genere si vuole controllare la [Items](#) insieme per vedere quali voci di menu di scelta rapida già esiste nel menu e quindi aggiungere o rimuovere appropriato new [MenuItem](#) da o verso la raccolta di elementi.

```
void AddItemToCM(object sender, ContextMenuEventArgs e)
{
    //check if Item4 is already there, this will probably run more than once
    FrameworkElement fe = e.Source as FrameworkElement;
    ContextMenu cm = fe.ContextMenu;
    foreach (MenuItem mi in cm.Items)
    {
        if ((String)mi.Header == "Item4") return;
    }
    MenuItem mi4 = new MenuItem();
    mi4.Header = "Item4";
    fe.ContextMenu.Items.Add(mi4);
}
```

Sostituire l'intero Menu prima della visualizzazione

Uno scenario alternativo è se si desidera sostituire il menu di scelta rapida intero. Naturalmente è possibile usare

anche una variante del codice precedente, per rimuovere ogni elemento di un menu di scelta rapida esistente e aggiungerne di nuovi, iniziando dall'elemento zero. Ma l'approccio più intuitiva per la sostituzione di tutti gli elementi nel menu di scelta rapida consiste nel creare una nuova [ContextMenu](#) viene popolato con gli elementi e quindi impostare il [FrameworkElement.ContextMenu](#) proprietà di un controllo che sarà il nuovo [ContextMenu](#).

Di seguito è riportato il codice del gestore semplice per la sostituzione di un [ContextMenu](#). Il codice fa riferimento a un oggetto personalizzato [BuildMenu](#) metodo, che è separato perché viene chiamato da più di uno dei gestori di esempio.

```
void HandlerForCMO(object sender, ContextMenuEventArgs e)
{
    FrameworkElement fe = e.Source as FrameworkElement;
    fe.ContextMenu = BuildMenu();
}
```

```
ContextMenu BuildMenu()
{
    ContextMenu theMenu = new ContextMenu();
    MenuItem mia = new MenuItem();
    mia.Header = "Item1";
    MenuItem mib = new MenuItem();
    mib.Header = "Item2";
    MenuItem mic = new MenuItem();
    mic.Header = "Item3";
    theMenu.Items.Add(mia);
    theMenu.Items.Add(mib);
    theMenu.Items.Add(mic);
    return theMenu;
}
```

Tuttavia, se si utilizza questo stile di visualizzazione del gestore per [ContextMenuOpening](#), si può esporre potenzialmente un problema di temporizzazione se l'oggetto in cui si sta impostando il [ContextMenu](#) non dispone di un menu di scelta rapida preesistente. Quando un utente fa clic di un controllo [ContextMenuOpening](#) viene generato anche se l'oggetto esistente [ContextMenu](#) è vuoto o null. Ma in questo caso, qualsiasi nuovo [ContextMenu](#) è impostata sull'origine elemento arriva troppo tardi per poter essere visualizzati. Inoltre, se l'utente a fare doppio clic su una seconda volta, questa volta il nuovo [ContextMenu](#) visualizzata, il valore è non null e il gestore verrà sostituire correttamente e verrà visualizzato il menu di scelta quando il gestore esegue una seconda volta. Ciò suggerisce possibili due soluzioni alternative:

1. Assicurarsi che [ContextMenuOpening](#) gestori di eseguire sempre i controlli che hanno almeno un segnaposto [ContextMenu](#) disponibili, che si prevede essere sostituito con il codice del gestore. In questo caso, è comunque possibile usare il gestore illustrato nell'esempio precedente, ma in genere si vuole assegnare un segnaposto [ContextMenu](#) nel markup iniziale:

```
<StackPanel>
    <Rectangle Fill="Yellow" Width="200" Height="100" ContextMenuOpening="HandlerForCMO">
        <Rectangle.ContextMenu>
            <ContextMenu>
                <MenuItem>Initial menu; this will be replaced ...</MenuItem>
            </ContextMenu>
        </Rectangle.ContextMenu>
    </Rectangle>
    <TextBlock>Right-click the rectangle above, context menu gets replaced</TextBlock>
</StackPanel>
```

2. Si supponga che iniziale [ContextMenu](#) valore potrebbe essere null, basato su logica preliminare. È possibile archiviarla [ContextMenu](#) è null o usare un flag nel codice per verificare se il gestore di è stato eseguito

almeno una volta. Poiché si presuppone che il [ContextMenu](#) sta per essere visualizzato, il gestore quindi imposta [Handled](#) a `true` nei dati dell'evento. Per il [ContextMenuService](#) responsabile per la visualizzazione di menu di scelta rapida, una `true` value per [Handled](#) nell'evento dati rappresentano una richiesta di annullare la visualizzazione per il menu di scelta / controllo combinazione che ha generato l'evento.

Dopo aver eliminato il menu di scelta rapida potenzialmente sospetti, il passaggio successivo è fornire uno nuovo, quindi visualizzarlo. L'impostazione di nuovo uno è fondamentalmente lo stesso come il gestore precedente: si crea un nuovo [ContextMenu](#) e impostare l'origine di controllo [FrameworkElement.ContextMenu](#) proprietà con esso. Il passaggio aggiuntivo è che ora è necessario forzare la visualizzazione del menu di scelta rapida, perché è stato evitato al primo tentativo. Per forzare la visualizzazione, si imposta la [Popup.IsOpen](#) proprietà `true` all'interno del gestore. Prestare attenzione quando si esegue questa operazione, poiché aprendo il menu di scelta rapida nel gestore genera il [ContextMenuOpening](#) nuovo evento. Se si accede nuovamente al gestore, diventa infinitamente ricorsiva. Ecco perché è sempre necessario verificare la presenza `null` oppure usare un flag, se si apre un menu di scelta rapida dall'interno una [ContextMenuOpening](#) gestore dell'evento.

Eliminazione di qualsiasi menu di scelta rapida esistente e non visualizzare alcun menu di scelta rapida

Nello scenario finale, scrivere un gestore che elimina completamente, un menu è insolito. Se un determinato controllo non deve per visualizzare un menu di scelta rapida, esistono modi probabilmente più appropriati per assicurare che questa rispetto all'eliminazione del menu solo quando un utente lo richiede. Se si desidera utilizzare il gestore per eliminare un menu di scelta rapida e non visualizzare nulla, allora il gestore di è sufficiente deve impostare [Handled](#) a `true` nei dati dell'evento. Il [ContextMenuService](#) che è responsabile della visualizzazione di un menu di scelta rapida controllerà i dati dell'evento dell'evento generato sul controllo. Se l'evento è stato contrassegnato come [Handled](#) in qualsiasi punto lungo la route, quindi l'azione di Apri menu di scelta rapida che ha avviato l'evento viene eliminato.

```
void HandlerForCM02(object sender, ContextMenuEventArgs e)
{
    if (!FlagForCustomContextMenu)
    {
        e.Handled = true; //need to suppress empty menu
        FrameworkElement fe = e.Source as FrameworkElement;
        fe.ContextMenu = BuildMenu();
        FlagForCustomContextMenu = true;
        fe.ContextMenu.IsOpen = true;
    }
}
```

Vedere anche

- [ContextMenu](#)
- [FrameworkElement.ContextMenu](#)
- [Cenni preliminari sugli elementi di base](#)
- [Panoramica sull'oggetto ContextMenu](#)

Struttura ad albero e serializzazione degli elementi

04/11/2019 • 2 minutes to read • [Edit Online](#)

Gli elementi della programmazione WPF spesso esistono sotto forma di relazioni all'interno di un albero. Ad esempio, l'interfaccia utente di un'applicazione creata in XAML può essere definita dal punto di vista concettuale come albero di oggetti. L'albero degli elementi può essere suddiviso ulteriormente in due alberi discreti e tuttavia a volte paralleli: l'albero logico e la struttura ad albero visuale. In WPF la serializzazione include il salvataggio dello stato di questi due alberi oltre che dello stato dell'applicazione e la scrittura di tale stato in un file, in genere XAML.

Contenuto della sezione

[Strutture ad albero in WPF](#)

[Limitazioni relative alla serializzazione di XamlWriter.Save](#)

[Inizializzazione di elementi oggetto non presenti in una struttura ad albero di oggetti](#)

[Procedure relative alle proprietà](#)

Reference

[System.Windows.Markup](#)

[LogicalTreeHelper](#)

[VisualTreeHelper](#)

Sezioni correlate

[Architettura WPF](#)

[XAML in WPF](#)

[Elementi di base](#)

[Proprietà](#)

[Eventi](#)

[Input](#)

[Risorse](#)

[Applicazione di stili e modelli](#)

[Modello di threading](#)

Strutture ad albero in WPF

03/02/2020 • 22 minutes to read • [Edit Online](#)

In molte tecnologie gli elementi e i componenti sono organizzati in una struttura ad albero in cui gli sviluppatori modificano direttamente i nodi degli oggetti nell'albero per influire sul rendering o sul comportamento di un'applicazione. Windows Presentation Foundation (WPF) usa anche molte metafore correlate alla struttura ad albero per definire le relazioni tra gli elementi del programma. In genere gli sviluppatori WPF possono creare un'applicazione nel codice o definire parti dell'applicazione in XAML usando come riferimento concettuale la metafora della struttura ad albero di oggetti, ma chiameranno un'API specifica o useranno un markup specifico a tale scopo anziché un'API di modifica della struttura ad albero di oggetti generica simile a quella usata nel modello DOM XML. WPF espone due classi helper che forniscono una visualizzazione della metafora dell'albero [LogicalTreeHelper](#) e [VisualTreeHelper](#). Nella documentazione di WPF vengono inoltre usati i termini struttura ad albero visuale e albero logico, in quanto tali strutture ad albero sono utili per la comprensione del comportamento di alcune funzionalità chiave di WPF. In questo argomento vengono descritte le caratteristiche della struttura ad albero visuale e dell'albero logico, viene illustrato il modo in cui tali alberi sono correlati a un concetto generale di albero degli oggetti e vengono introdotti [LogicalTreeHelper](#) e [VisualTreeHelper](#).

Strutture ad albero in WPF

La struttura ad albero più completa in WPF è la struttura ad albero di oggetti. Se si definisce la pagina di un'applicazione in XAML e quindi si carica il codice XAML, la struttura ad albero viene creata in base alle relazioni di annidamento degli elementi nel markup. Se si definisce un'applicazione o una parte dell'applicazione nel codice, la struttura ad albero viene creata in base a come si assegnano i valori per le proprietà che implementano il modello di contenuto per un determinato oggetto. In Windows Presentation Foundation (WPF) la struttura ad albero di oggetti completa viene concettualizzata e può essere segnalata alla relativa API pubblica in due modi diversi: come albero logico e come struttura ad albero visuale. Le distinzioni tra albero logico e struttura ad albero visuale non sono sempre necessariamente importanti, ma possono talvolta causare problemi con alcuni sottosistemi WPF e influire sulle scelte fatte nel markup o nel codice.

Anche se non sempre si modifica direttamente l'albero logico o la struttura ad albero visuale, la corretta comprensione dei concetti correlati all'interazione delle strutture ad albero è utile per comprendere WPF in quanto tecnologia. Il concetto di WPF come metafora della struttura ad albero è anche essenziale per comprendere il funzionamento dell'ereditarietà delle proprietà e del routing degli eventi in WPF.

NOTE

Poiché la struttura ad albero di oggetti è più un concetto che un'API effettiva, è possibile pensare a tale concetto anche come a un oggetto grafico. In pratica, in fase di esecuzione ci sono relazioni tra gli oggetti per cui la metafora di struttura ad albero non è valida. Ciononostante, in particolare con l'interfaccia utente definita in XAML, la metafora della struttura ad albero è sufficientemente pertinente da far sì che nella maggior parte della documentazione di WPF venga usato il termine struttura ad albero di oggetti per fare riferimento a tale concetto generale.

Albero logico

In WPF è possibile aggiungere contenuto agli elementi dell'interfaccia utente impostando proprietà degli oggetti di supporto per tali elementi. Ad esempio, si aggiungono elementi a un controllo [ListBox](#) modificando la relativa proprietà [Items](#). Questa operazione consente di inserire elementi nell'[ItemCollection](#) che corrisponde al valore della proprietà [Items](#). Analogamente, per aggiungere oggetti a una [DockPanel](#), è possibile modificare il relativo valore della proprietà [Children](#). Qui si aggiungono oggetti al [UIElementCollection](#). Per un esempio di codice,

vedere [procedura: aggiungere un elemento in modo dinamico](#).

In Extensible Application Markup Language (XAML), quando si inseriscono voci di elenco in un [ListBox](#) o in controlli o altri elementi dell'interfaccia utente in un [DockPanel](#), si usano anche le proprietà [Items](#) e [Children](#), in modo esplicito o implicito, come nell'esempio seguente.

```
<DockPanel  
    Name="ParentElement"  
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
    >  
    <!--implicit: <DockPanel.Children>-->  
    <ListBox DockPanel.Dock="Top">  
        <!--implicit: <ListBox.Items>-->  
        <ListBoxItem>  
            <TextBlock>Dog</TextBlock>  
        </ListBoxItem>  
        <ListBoxItem>  
            <TextBlock>Cat</TextBlock>  
        </ListBoxItem>  
        <ListBoxItem>  
            <TextBlock>Fish</TextBlock>  
        </ListBoxItem>  
        <!--implicit: </ListBox.Items>-->  
    </ListBox>  
    <Button Height="20" Width="100" DockPanel.Dock="Top">Buy a Pet</Button>  
    <!--implicit: </DockPanel.Children>-->  
</DockPanel>
```

Se si elaborasse questo codice XAML come XML in un modello DOM (Document Object Model) e fossero stati inclusi tag impostati come commenti impliciti (operazione consentita), la struttura ad albero DOM XML risultante includerebbe elementi per `<ListBox.Items>` e altri elementi impliciti. Poiché tuttavia XAML non elabora in questo modo durante la lettura del markup e la scrittura negli oggetti, l'oggetto grafico risultante non includerà letteralmente `ListBox.Items`. Dispone tuttavia di una proprietà [ListBox](#) denominata `Items` che contiene una [ItemCollection](#) che [ItemCollection](#) viene inizializzata ma vuota quando viene elaborato il [ListBox](#) XAML. Quindi, ogni elemento oggetto figlio esistente come contenuto per il [ListBox](#) viene aggiunto al [ItemCollection](#) dalle chiamate del parser per `ItemCollection.Add`. Fino a questo punto, in questo esempio di elaborazione di XAML in una struttura ad albero di oggetti la struttura ad albero di oggetti creata sembra fondamentalmente essere un albero logico.

Tuttavia, l'albero logico non è l'intero oggetto grafico esistente per l'interfaccia utente dell'applicazione in fase di esecuzione, anche in caso di factoring degli elementi della sintassi implicita XAML. Il motivo principale è che sono gli oggetti visivi e i modelli. Si consideri, ad esempio, il [Button](#). L'albero logico segnala l'oggetto [Button](#) e anche la relativa stringa `Content`. Nella struttura ad albero di oggetti di runtime, tuttavia, questo pulsante è molto più complesso. In particolare, il pulsante viene visualizzato sullo schermo nel modo in cui è stato applicato un modello di controllo [Button](#) specifico. Gli oggetti visivi provenienti da un modello applicato, ad esempio il [Border](#) definito dal modello di grigio scuro intorno al pulsante visivo, non vengono segnalati nell'albero logico, anche se si sta osservando l'albero logico in fase di esecuzione, ad esempio gestendo un evento di input dall'interfaccia utente visibile e quindi leggendo l'albero logico. Per trovare gli oggetti visivi del modello, è invece necessario esaminare la struttura ad albero visuale.

Per altre informazioni sul mapping tra la sintassi XAML e l'oggetto grafico creato e sulla sintassi implicita in XAML, vedere [Descrizione dettagliata della sintassi XAML](#) o [Cenni preliminari su XAML \(WPF\)](#).

Scopo dell'albero logico

L'albero logico consente ai modelli di contenuto di scorrere rapidamente i relativi oggetti figlio possibili e rende i modelli di contenuto estendibili. L'albero logico fornisce inoltre un framework per determinate notifiche, ad esempio relative al caricamento di tutti gli oggetti nell'albero logico stesso. Fondamentalmente l'albero logico è

un'approssimazione di un oggetto grafico di runtime a livello di framework, che esclude gli oggetti visivi, ma è efficace per molte operazioni di query sulla composizione dell'applicazione di runtime.

Inoltre, i riferimenti a risorse statiche e dinamiche vengono risolti osservando verso l'alto l'albero logico per le raccolte di [Resources](#) nell'oggetto richiedente iniziale, quindi proseguendo l'albero logico e controllando ogni [FrameworkElement](#) (o [FrameworkContentElement](#)) per un altro valore [Resources](#) contenente un [ResourceDictionary](#), possibilmente contenente tale chiave. L'albero logico viene usato per la ricerca delle risorse, quando sono presenti sia l'albero logico, sia la struttura ad albero visuale. Per altre informazioni sui dizionari risorse e sulla ricerca, vedere [Risorse XAML](#).

Composizione dell'albero logico

L'albero logico viene definito a livello di Framework WPF, il che significa che l'elemento di base WPF più rilevante per le operazioni di albero logico è [FrameworkElement](#) o [FrameworkContentElement](#). Tuttavia, come si può vedere se si usa effettivamente l'API [LogicalTreeHelper](#), l'albero logico contiene talvolta nodi che non sono [FrameworkElement](#) o [FrameworkContentElement](#). Ad esempio, l'albero logico segnala il valore [Text](#) di un [TextBlock](#), che è una stringa.

Override dell'albero logico

Gli autori di controlli avanzati possono eseguire l'override dell'albero logico eseguendo l'override di diverse API che definiscono il modo in cui un oggetto generale o un modello di contenuto aggiunge o rimuove oggetti all'interno dell'albero logico. Per un esempio relativo all'override dell'albero logico, vedere [Eseguire l'override dell'albero logico](#).

Ereditarietà del valore della proprietà

L'ereditarietà dei valori delle proprietà funziona tramite un albero ibrido. I metadati effettivi che contengono la proprietà [Inherits](#) che consente l'ereditarietà della proprietà è la classe [FrameworkPropertyMetadata](#) a livello di Framework WPF. Pertanto, sia l'elemento padre che include il valore originale che l'oggetto figlio che eredita tale valore devono essere [FrameworkElement](#) o [FrameworkContentElement](#) entrambi devono essere parte di un albero logico. Per le proprietà WPF esistenti che supportano l'ereditarietà delle proprietà, tuttavia, l'ereditarietà dei valori delle proprietà può essere mantenuta tramite un nuovo oggetto non incluso nell'albero logico. Questa caratteristica è pertinente per lo più se si vuole fare in modo che gli elementi del modello usino valori di proprietà ereditati impostati sull'istanza basata sul modello o a livelli ancora superiori di composizione a livello di pagina e quindi più in alto nell'albero logico. Per garantire un funzionamento coerente dell'ereditarietà dei valori delle proprietà oltre tale limite, la proprietà che eredita deve essere registrata come proprietà associata. È consigliabile seguire questo modello anche se si vuole definire una proprietà di dipendenza personalizzata con un comportamento di ereditarietà della proprietà. L'albero esatto usato per l'ereditarietà delle proprietà non può essere completamente previsto da un metodo di utilità di una classe helper, nemmeno in fase di esecuzione. Per altre informazioni, vedere [Ereditarietà del valore della proprietà](#).

Struttura ad albero visuale

In WPF oltre al concetto di albero logico c'è anche il concetto di struttura ad albero visuale. La struttura ad albero visuale descrive la struttura degli oggetti visivi, come rappresentati dalla classe di base [Visual](#). Quando si scrive un modello per un controllo, si definisce o ridefinisce la struttura ad albero visuale relativa a quel controllo. La struttura ad albero visuale è di interesse anche per gli sviluppatori che vogliono un controllo di livello inferiore sul disegno per ragioni di prestazioni e ottimizzazione. Un'esposizione della struttura ad albero visuale come parte della programmazione di applicazioni WPF convenzionale è rappresentata dal fatto che le route degli eventi indirizzati percorrono per lo più la struttura ad albero visuale e non l'albero logico. Questa sottigliezza del comportamento degli eventi indirizzati potrebbe non essere immediatamente visibile, a meno che l'utente non sia un autore di controlli. Il routing di eventi nella struttura ad albero visuale consente ai controlli che implementano la composizione a livello visivo di gestire eventi o creare setter di eventi.

Alberi, elementi di contenuto e host di contenuto

Gli elementi di contenuto (classi che derivano da [ContentElement](#)) non fanno parte della struttura ad albero visuale. non ereditano da [Visual](#) e non dispongono di una rappresentazione visiva. Per essere visualizzato in un'interfaccia utente, è necessario che un [ContentElement](#) sia ospitato in un host di contenuto che sia un [Visual](#) e un partecipante dell'albero logico. Un oggetto di questo tipo è in genere un [FrameworkElement](#). È possibile pensare all'host di contenuto come a un "browser" per il contenuto, che sceglie come visualizzare tale contenuto all'interno dell'area dello schermo controllata dall'host. Quando il contenuto è ospitato, può partecipare ad alcuni processi dell'albero che normalmente sono associati alla struttura ad albero visuale. In genere, la classe host [FrameworkElement](#) include il codice di implementazione che aggiunge qualsiasi [ContentElement](#) ospitata alla route dell'evento tramite sottonodi dell'albero logico del contenuto, anche se il contenuto ospitato non fa parte della struttura ad albero visuale vera e propria. Questa operazione è necessaria in modo che un [ContentElement](#) possa originare un evento indirizzato che viene indirizzato a qualsiasi elemento diverso da se stesso.

Attraversamento dell'albero

La classe [LogicalTreeHelper](#) fornisce i metodi [GetChildren](#), [GetParent](#) e [FindLogicalNode](#) per l'attraversamento dell'albero logico. Nella maggior parte dei casi, non è necessario attraversare l'albero logico dei controlli esistenti, perché i controlli espongono quasi sempre i relativi elementi figlio logici come proprietà di raccolta dedicata che supporta l'accesso alla raccolta, ad esempio [Add](#), un indicizzatore e così via. L'attraversamento dell'albero è principalmente uno scenario usato dagli autori di controlli che scelgono di non derivare da pattern di controllo previsti, ad esempio [ItemsControl](#) o [Panel](#) in cui le proprietà della raccolta sono già definite e che desiderano fornire il proprio supporto della proprietà della raccolta.

La struttura ad albero visuale supporta inoltre una classe helper per l'attraversamento della struttura ad albero visuale [VisualTreeHelper](#). La struttura ad albero visuale non è esposta in modo appropriato tramite proprietà specifiche del controllo, quindi la classe [VisualTreeHelper](#) è il modo consigliato per attraversare la struttura ad albero visuale, se necessario per lo scenario di programmazione. Per altre informazioni, vedere [Cenni preliminari sul rendering della grafica WPF](#).

NOTE

A volte è necessario esaminare la struttura ad albero visuale di un modello applicato. Quando si usa questa tecnica, è necessario procedere con attenzione. Anche se si attraversa una struttura ad albero visuale per un controllo in cui si definisce il modello, gli utenti del controllo possono sempre modificare il modello impostando la proprietà [Template](#) sulle istanze e anche l'utente finale può influenzare il modello applicato modificando il tema del sistema.

Route per eventi indirizzati come "albero"

Come indicato in precedenza, la route di qualsiasi evento indirizzato percorre un singolo percorso predeterminato di un albero che consiste in una forma ibrida delle rappresentazioni di struttura ad albero visuale e albero logico. La route degli eventi può percorrere l'albero procedendo verso l'alto o verso il basso, a seconda che si tratti di un evento indirizzato di bubbling o di tunneling. Il concetto di route dell'evento non prevede una classe helper di supporto diretto, che potrebbe essere usata per "percorrere" la route dell'evento indipendentemente dalla generazione di un evento che effettivamente percorre una route. Esiste una classe che rappresenta la route, [EventRoute](#), ma i metodi di tale classe sono in genere per uso interno.

Dizionari risorse e alberi

La ricerca nei dizionari risorse di tutti gli oggetti [Resources](#) definiti in una pagina attraversa fondamentalmente l'albero logico. Gli oggetti non inclusi nell'albero logico possono fare riferimento a risorse con chiave, ma la sequenza di ricerca delle risorse inizia nel punto in cui l'oggetto è connesso all'albero logico. In WPF solo i nodi della struttura ad albero logica possono avere una proprietà [Resources](#) che contiene una [ResourceDictionary](#), pertanto non vi è alcun vantaggio nell'attraversamento della struttura ad albero visuale per la ricerca di risorse

con chiave da una [ResourceDictionary](#).

La ricerca delle risorse può però estendersi anche oltre l'albero logico diretto. Per il markup dell'applicazione, la ricerca delle risorse può proseguire verso l'altro nei dizionari risorse a livello di applicazione e quindi verso il supporto dei temi e i valori di sistema a cui viene fatto riferimento come chiavi o proprietà statiche. Se i riferimenti alle risorse sono dinamici, i temi stessi possono fare riferimento anche ai valori di sistema esterni all'albero logico del tema. Per altre informazioni sui dizionari risorse e sulla logica di ricerca, vedere [Risorse XAML](#).

Vedere anche

- [Cenni preliminari sull'input](#)
- [Cenni preliminari sul rendering della grafica WPF](#)
- [Cenni preliminari sugli eventi indirizzati](#)
- [Inizializzazione di elementi oggetto non presenti in una struttura ad albero di oggetti](#)
- [Architettura WPF](#)

Limitazioni relative alla serializzazione di XamlWriter.Save

08/11/2019 • 5 minutes to read • [Edit Online](#)

Il [Save](#) API può essere usato per serializzare il contenuto di un'applicazione Windows Presentation Foundation (WPF) come file Extensible Application Markup Language (XAML). Esistono tuttavia limitazioni considerevoli relative alla definizione precisa dell'oggetto della serializzazione. Queste limitazioni vengono esaminate in questo argomento, insieme ad alcune considerazioni generali.

Rappresentazione della fase di esecuzione, non della fase di progettazione

La filosofia di base di ciò che viene serializzato da una chiamata a [Save](#) è che il risultato sarà una rappresentazione dell'oggetto che viene serializzato in fase di esecuzione. Molte proprietà della fase di progettazione del file di XAML originale potrebbero essere già state ottimizzate o perse nel momento in cui la XAML viene caricata come oggetti in memoria e non vengono mantenute quando si chiama [Save](#) per la serializzazione. Il risultato della serializzazione è una rappresentazione efficace dell'albero logico dell'applicazione costruito, ma non necessariamente del file XAML che lo ha prodotto. Questi problemi rendono estremamente difficile utilizzare la serializzazione [Save](#) come parte di un'ampia XAML area di progettazione.

La serializzazione è autonoma

L'output serializzato di [Save](#) è indipendente; tutto ciò che viene serializzato è contenuto all'interno di una XAML singola pagina, con un singolo elemento radice e senza riferimenti esterni diversi dagli URI. Ad esempio, se una pagina faceva riferimento a risorse dalle risorse dell'applicazione, queste appariranno come componenti della pagina serializzata.

I riferimenti alle estensioni vengono dereferenziati

I riferimenti comuni agli oggetti realizzati in diversi formati di estensione di markup, ad esempio `StaticResource` o `Binding`, saranno dereferenziati durante il processo di serializzazione. Tali oggetti sono già stati dereferenziati nel momento in cui gli oggetti in memoria sono stati creati dal runtime dell'applicazione e la logica [Save](#) non rivisita la XAML originale per ripristinare tali riferimenti all'output serializzato. In questo modo, potenzialmente qualsiasi valore associato ai dati o proveniente dalla risorsa viene bloccato sull'ultimo valore usato dalla rappresentazione in fase di esecuzione, con possibilità solo limitate o indirette di distinguere da qualsiasi altro valore impostato localmente. Le immagini vengono inoltre serializzate come riferimenti a oggetti alle immagini esistenti nel progetto, anziché come riferimenti originali di origine, perdendo il nome file o l'URI a cui è stato originariamente fatto riferimento. Perfino le risorse dichiarate all'interno della stessa pagina sono serializzate nel punto in cui vengono referenziate, anziché essere conservate come chiave della raccolta di risorse.

La gestione degli eventi non viene mantenuta

Quando i gestori eventi aggiunti tramite XAML vengono serializzati, non vengono mantenuti. XAML senza code-behind (e senza il relativo meccanismo x:Code) non consente di serializzare la logica procedurale di runtime. Poiché la serializzazione è autonoma e limitata all'albero logico, non sono disponibili funzionalità per l'archiviazione dei gestori eventi. Di conseguenza, gli attributi dei gestori eventi, sia l'attributo stesso che il valore della stringa indicante il nome del gestore, vengono rimossi dall'output XAML.

Scenari realistici per l'uso di XamlWriter.Save

Sebbene le limitazioni elencate qui siano piuttosto sostanziali, esistono ancora diversi scenari appropriati per l'uso di [Save](#) per la serializzazione.

- Output vettoriale o grafico: l'output dell'area sottoposta a rendering può essere usato per riprodurre lo stesso vettore o la stessa grafica se caricato nuovamente.
- Documenti RTF o dinamici: il testo con tutta la formattazione e il contenimento degli elementi inclusi viene mantenuto nell'output. Questo può essere utile per i meccanismi che si avvicinano a una funzionalità degli Appunti.
- Conservazione dei dati degli oggetti business: se i dati sono stati archiviati in elementi personalizzati, ad esempio i dati XML, a condizione che gli oggetti business seguano regole di base XAML come la fornitura di costruttori e conversione personalizzati per i valori delle proprietà per riferimento, queste attività è possibile perpetuare gli oggetti tramite la serializzazione.

Inizializzazione di elementi oggetto non presenti in una struttura ad albero di oggetti

04/11/2019 • 6 minutes to read • [Edit Online](#)

Alcuni aspetti dell'inizializzazione di Windows Presentation Foundation (WPF) vengono rinviati ai processi che in genere si basano sulla connessione dell'elemento interessato all'albero logico o alla struttura ad albero visuale. In questo argomento vengono descritti i passaggi necessari per inizializzare un elemento non connesso ad alcuno di questi alberi.

Elementi e albero logico

Quando si crea un'istanza di una classe Windows Presentation Foundation (WPF) all'interno del codice, è necessario tenere presente che molti aspetti dell'inizializzazione dell'oggetto per una classe Windows Presentation Foundation (WPF) non fanno deliberatamente parte del codice che viene eseguito quando si chiama il costruttore della classe. In particolare, per le classi di controlli, la maggior parte della rappresentazione visiva di un controllo non viene definita dal costruttore, ma dal modello del controllo. Sebbene possa provenire da diverse origini, nella maggior parte dei casi il modello viene ottenuto dagli stili dei temi. I modelli sono in pratica un binding tardivo. Il modello necessario viene associato al controllo in questione solo quando il controllo è pronto per il layout e quest'ultima condizione si verifica solo quando il controllo viene associato a un albero logico che si connette a una superficie di rendering nella radice. È tale elemento a livello radice che avvia il rendering di tutti i relativi elementi figlio definiti nell'albero logico.

Anche la struttura ad albero visuale partecipa a questo processo. Anche un'istanza completa per gli elementi che fanno parte della struttura ad albero visuale tramite i modelli viene creata solo quando gli elementi stessi vengono connessi.

La conseguenza di questo comportamento è la necessità di passaggi aggiuntivi per alcune operazioni basate sulle caratteristiche visive completate di un elemento. Un esempio è rappresentato dal tentativo di ottenere le caratteristiche visive di una classe costruita, ma non ancora associata a un albero. Se, ad esempio, si desidera chiamare `Render` in un `RenderTargetBitmap` e l'oggetto visivo che si sta passando è un elemento non connesso a un albero, l'elemento non viene completato visivamente fino a quando non vengono completati altri passaggi di inizializzazione.

Uso di `BeginInit` e `EndInit` per inizializzare l'elemento

Varie classi di WPF implementano l'interfaccia `ISupportInitialize`. Usare i metodi `BeginInit` e `EndInit` dell'interfaccia per indicare un'area nel codice che contiene passaggi di inizializzazione, ad esempio l'impostazione di valori di proprietà che influiscono sul rendering. Dopo la chiamata di `EndInit` nella sequenza, il sistema di layout può elaborare l'elemento e iniziare a cercare uno stile implicito.

Se l'elemento in cui si impostano le proprietà è un `FrameworkElement` o `FrameworkContentElement` classe derivata, è possibile chiamare le versioni della classe `BeginInit` e `EndInit` anziché eseguire il cast a `ISupportInitialize`.

Codice di esempio

Nell'esempio seguente è riportato il codice di esempio per un'applicazione console che usa le API di rendering e `XamlReader.Load(Stream)` di un file di XAML libero per illustrare il posizionamento appropriato di `BeginInit` e `EndInit` intorno ad altre chiamate API che regolano le proprietà che influiscono sul rendering.

L'esempio illustrata solo la funzione principale. Le funzioni `Rasterize` e `Save` (non illustrate) sono funzioni di utilità che gestiscono l'elaborazione di immagini e l'I/O.

```
[STAThread]
static void Main(string[] args)
{
    UIElement e;
    string file = Directory.GetCurrentDirectory() + "\\starting.xaml";
    using (Stream stream = File.Open(file, FileMode.Open))
    {
        // loading files from current directory, project settings take care of copying the file
        ParserContext pc = new ParserContext();
        pc.BaseUri = new Uri(file, UriKind.Absolute);
        e = (UIElement)XamlReader.Load(stream, pc);
    }

    Size paperSize = new Size(8.5 * 96, 11 * 96);
    e.Measure(paperSize);
    e.Arrange(new Rect(paperSize));
    e.UpdateLayout();

/*
 *    Render effect at normal dpi, indicator is the original RED rectangle
 */
RenderTargetBitmap image1 = Rasterize(e, paperSize.Width, paperSize.Height, 96, 96);
Save(image1, "render1.png");

Button b = new Button();
b.BeginInit();
b.Background = Brushes.Blue;
b.Width = b.Height = 200;
b.EndInit();
b.Measure(paperSize);
b.Arrange(new Rect(paperSize));
b.UpdateLayout();

// now render the altered version, with the element built up and initialized

RenderTargetBitmap image2 = Rasterize(b, paperSize.Width, paperSize.Height, 96, 96);
Save(image2, "render2.png");
}
```

```

<STAThread>
Shared Sub Main(ByVal args() As String)
    Dim e As UIElement
    Dim _file As String = Directory.GetCurrentDirectory() & "\starting.xaml"
    Using stream As Stream = File.Open(_file, FileMode.Open)
        ' loading files from current directory, project settings take care of copying the file
        Dim pc As New ParserContext()
        pc.BaseUri = New Uri(_file, UriKind.Absolute)
        e = CType(XamlReader.Load(stream, pc), UIElement)
    End Using

    Dim paperSize As New Size(8.5 * 96, 11 * 96)
    e.Measure(paperSize)
    e.Arrange(New Rect(paperSize))
    e.UpdateLayout()

    '
    * Render effect at normal dpi, indicator is the original RED rectangle

    Dim image1 AsRenderTargetBitmap = Rasterize(e, paperSize.Width, paperSize.Height, 96, 96)
    Save(image1, "render1.png")

    Dim b As New Button()
    b.BeginInit()
    b.Background = Brushes.Blue
    b.Height = 200
    b.Width = b.Height
    b.EndInit()
    b.Measure(paperSize)
    b.Arrange(New Rect(paperSize))
    b.UpdateLayout()

    ' now render the altered version, with the element built up and initialized

    Dim image2 AsRenderTargetBitmap = Rasterize(b, paperSize.Width, paperSize.Height, 96, 96)
    Save(image2, "render2.png")
End Sub

```

Vedere anche

- [Strutture ad albero in WPF](#)
- [Cenni preliminari sul rendering della grafica WPF](#)
- [Cenni preliminari su XAML \(WPF\)](#)

Procedure relative alla struttura ad albero e alla serializzazione degli elementi

23/10/2019 • 2 minutes to read • [Edit Online](#)

Gli argomenti in questa sezione descrivono come usare l'albero degli elementi WPF.

In questa sezione

[Trovare un elemento in base al nome](#)

[Eseguire l'override dell'albero logico](#)

Riferimenti

[LogicalTreeHelper](#)

[VisualTreeHelper](#)

[System.Windows.Markup](#)

Sezioni correlate

Procedura: Trovare un elemento in base al nome

23/10/2019 • 2 minutes to read • [Edit Online](#)

In questo esempio viene descritto come utilizzare il [FindName](#) metodo per trovare un elemento dal relativo [Name](#) valore.

Esempio

In questo esempio, il metodo per trovare un particolare elemento in base al relativo nome viene scritto come il gestore di eventi di un pulsante. `stackPanel` è il [Name](#) della radice [FrameworkElement](#) cercato, e il metodo di esempio visivamente indica quindi l'elemento trovato eseguendo il cast come [TextBlock](#) e modificare uno del [TextBlock](#) visibile Interfaccia utente proprietà.

```
void Find(object sender, RoutedEventArgs e)
{
    object wantedNode = stackPanel.FindName("dog");
    if (wantedNode is TextBlock)
    {
        // Following executed if Text element was found.
        TextBlock wantedChild = wantedNode as TextBlock;
        wantedChild.Foreground = Brushes.Blue;
    }
}
```

```
Private Sub Find(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Dim wantedNode As Object = stackPanel.FindName("dog")
    If TypeOf wantedNode Is TextBlock Then
        ' Following executed if Text element was found.
        Dim wantedChild As TextBlock = TryCast(wantedNode, TextBlock)
        wantedChild.Foreground = Brushes.Blue
    End If
End Sub
```

Procedura: Eseguire l'override dell'albero logico

23/10/2019 • 2 minutes to read • [Edit Online](#)

Benché nella maggior parte dei casi non sia necessario, gli autori di controlli avanzati hanno la possibilità di eseguire l'override dell'albero logico.

Esempio

In questo esempio viene illustrato come creare una sottoclasse `StackPanel` all'override dell'albero logico, in questo caso per imporre un comportamento che il pannello può avere solo e verrà eseguito solo il rendering di un singolo elemento figlio. Benché non si tratti necessariamente di un comportamento utile, viene presentato come mezzo per illustrare lo scenario di override del normale albero logico di un elemento.

```
public class SingletonPanel : StackPanel
{
    //private UIElementCollection _children;
    private FrameworkElement _child;

    public SingletonPanel()
    {
    }

    public FrameworkElement SingleChild
    {

        get { return _child; }
        set
        {
            if (value == null)
            {
                RemoveLogicalChild(_child);
            }
            else
            {
                if (_child == null)
                {
                    _child = value;
                }
                else
                {
                    // raise an exception?
                    MessageBox.Show("Needs to be a single element");
                }
            }
        }
    }

    public void SetSingleChild(object child)
    {
        this.AddLogicalChild(child);
    }

    public new void AddLogicalChild(object child)
    {
        _child = (FrameworkElement)child;
        if (this.Children.Count == 1)
        {
            this.RemoveLogicalChild(this.Children[0]);
            this.Children.Add((UIElement)child);
        }
        else
    }
}
```

```
        {
            this.Children.Add((UIElement)child);
        }
    }

    public new void RemoveLogicalChild(object child)
    {
        _child = null;
        this.Children.Clear();
    }

    protected override IEnumerator LogicalChildren
    {
        get
        {
            // cheat, make a list with one member and return the enumerator
            ArrayList _list = new ArrayList();
            _list.Add(_child);
            return (IEnumerator)_list.GetEnumerator();
        }
    }
}
```

Per altre informazioni sull'albero logico, vedere [Strutture ad albero in WPF](#).

Proprietà (WPF)

03/02/2020 • 2 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) fornisce un set di servizi che possono essere utilizzati per estendere la funzionalità di una proprietà di Common Language Runtime (CLR). In genere, questi servizi vengono definiti collettivamente come sistema di proprietà WPF. Una proprietà supportata dal sistema di proprietà WPF è nota come proprietà di dipendenza.

Contenuto della sezione

- [Cenni preliminari sulle proprietà di dipendenza](#)
- [Cenni preliminari sulle proprietà associate](#)
- [Proprietà di dipendenza personalizzate](#)
- [Metadati delle proprietà di dipendenza](#)
- [Callback e convalida delle proprietà di dipendenza](#)
- [Metadati delle proprietà del framework](#)
- [Precedenza del valore della proprietà di dipendenza](#)
- [Proprietà di dipendenza di sola lettura](#)
- [Ereditarietà del valore della proprietà](#)
- [Sicurezza delle proprietà di dipendenza](#)
- [Modelli di costruttore sicuri per DependencyObject](#)
- [Proprietà di dipendenza di tipo raccolta](#)
- [Caricamento XAML e proprietà di dipendenza](#)
- [Procedure relative alla struttura ad albero e alla serializzazione degli elementi](#)

Riferimento

[DependencyProperty](#)

[PropertyMetadata](#)

[FrameworkPropertyMetadata](#)

[DependencyObject](#)

Sezioni correlate

[Architettura WPF](#)

[XAML in WPF](#)

[Elementi di base](#)

[Albero degli elementi e serializzazione](#)

[Eventi](#)

[Input](#)

[Risorse](#)

[Modello di contenuto WPF](#)

[Modello di threading](#)

Panoramica sulle proprietà di dipendenza

25/11/2019 • 26 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) include un set di servizi che è possibile usare per estendere la funzionalità di una [proprietà](#) di un tipo. In genere, questi servizi vengono definiti collettivamente come sistema di proprietà WPF. Una proprietà supportata dal sistema di proprietà WPF è nota come proprietà di dipendenza. Questa panoramica descrive il sistema di proprietà WPF e le funzionalità di una proprietà di dipendenza. Viene anche spiegato come usare le proprietà di dipendenza esistenti in XAML e nel codice. In questa panoramica vengono anche illustrati aspetti specifici delle proprietà di dipendenza, ad esempio i metadati delle proprietà di dipendenza e la creazione di una proprietà di dipendenza in una classe personalizzata.

Prerequisites

In questo argomento si presuppone che l'utente disponga di una conoscenza di base del sistema di tipi .NET e della programmazione orientata a oggetti. Per seguire gli esempi illustrati in questo argomento, è anche necessario conoscere XAML e saper scrivere applicazioni WPF. Per altre informazioni, vedere [Procedura dettagliata: La prima applicazione desktop WPF](#).

Proprietà di dipendenza e proprietà CLR

In WPF le proprietà vengono generalmente esposte come [proprietà](#) .NET standard. A un livello di base, si potrebbe interagire direttamente con queste proprietà senza sapere che vengono implementate come una proprietà di dipendenza. Tuttavia, è necessario acquisire familiarità con alcune o tutte le funzionalità del sistema di proprietà WPF, in modo da poterle sfruttare.

Lo scopo delle proprietà di dipendenza consiste nel fornire un modo per calcolare il valore di una proprietà in base al valore di altri input. Questi potrebbero includere proprietà del sistema, quali temi e preferenze dell'utente, meccanismi di determinazione della proprietà JIT, ad esempio data binding dati e animazioni o storyboard, modelli multiuso, quali risorse e stili oppure valori noti tramite relazioni padre-figlio con altri elementi dell'albero degli elementi. Inoltre, una proprietà di dipendenza può essere implementata per fornire una convalida autonoma, valori predefiniti, callback per il monitoraggio delle modifiche di altre proprietà, nonché un sistema che può assegnare forzatamente valori della proprietà in base a potenziali informazioni di runtime. Le classi derivate possono anche modificare alcune caratteristiche specifiche di una proprietà esistente, eseguendo l'override dei metadati della proprietà di dipendenza invece dell'override dell'implementazione effettiva delle proprietà esistenti oppure della creazione di nuove proprietà.

Nel riferimento SDK, è possibile identificare la proprietà di dipendenza grazie alla sezione Informazioni sulle proprietà di dipendenza, presente nella pagina di riferimento gestita per quella proprietà. Nella sezione sono inclusi un collegamento al campo dell'identificatore [DependencyProperty](#) per quella proprietà di dipendenza e un elenco delle opzioni dei metadati impostati per quella proprietà, informazioni sull'override per classe e altri dettagli.

Proprietà di dipendenza che supportano le proprietà CLR

Le proprietà di dipendenza e il sistema di proprietà WPF estendono le funzionalità della proprietà fornendo un tipo che supporta una proprietà, come implementazione alternativa al modello standard di supporto della proprietà con un campo privato. Il nome del tipo è [DependencyProperty](#). L'altro tipo importante che definisce il sistema di proprietà WPF è [DependencyObject](#). [DependencyObject](#) definisce la classe di base che può essere registrata ed essere proprietaria di una proprietà di dipendenza.

Di seguito è elencata la terminologia usata con le proprietà di dipendenza:

- **Proprietà di dipendenza:** proprietà supportata da un oggetto [DependencyProperty](#).
- **Identificatore della proprietà di dipendenza:** istanza di [DependencyProperty](#), ottenuta come valore restituito quando si registra una proprietà di dipendenza e quindi archiviata come membro statico di una classe. Questo identificatore viene usato come parametro per molte API che interagiscono con il sistema di proprietà WPF.
- **"Wrapper" CLR:** le implementazioni effetti Get e Set per la proprietà. Queste implementazioni includono l'identificatore della proprietà di dipendenza usandolo nelle chiamate [GetValue](#) e [SetValue](#), in modo da fornire il supporto per la proprietà tramite il sistema di proprietà WPF.

L'esempio seguente definisce la proprietà di dipendenza `IsSpinning` e mostra la relazione tra l'identificatore [DependencyProperty](#) e la proprietà supportata.

```
public static readonly DependencyProperty IsSpinningProperty =
    DependencyProperty.Register(
        "IsSpinning", typeof(Boolean),
        typeof(MyCode)
    );
public bool IsSpinning
{
    get { return (bool)GetValue(IsSpinningProperty); }
    set { SetValue(IsSpinningProperty, value); }
}
```

```
Public Shared ReadOnly IsSpinningProperty As DependencyProperty =
    DependencyProperty.Register("IsSpinning",
        GetType(Boolean),
        GetType(MyCode))

Public Property IsSpinning() As Boolean
    Get
        Return CBool(GetValue(IsSpinningProperty))
    End Get
    Set(ByVal value As Boolean)
        SetValue(IsSpinningProperty, value)
    End Set
End Property
```

La convenzione di denominazione della proprietà e del relativo campo [DependencyProperty](#) di supporto è importante. Il nome del campo è sempre il nome della proprietà al quale viene aggiunto il suffisso `Property`. Per altre informazioni su questa convenzione e i relativi motivi, vedere [Proprietà di dipendenza personalizzate](#).

Impostazione dei valori delle proprietà

È possibile impostare le proprietà nel codice o in XAML.

Impostazione dei valori delle proprietà in XAML

L'esempio di XAML seguente specifica il colore di sfondo rosso per un pulsante. Questo esempio illustra un caso in cui il valore di stringa semplice per un attributo XAML viene convertito dal parser XAML WPF in un tipo WPF ([Color](#) tramite [SolidColorBrush](#)) nel codice generato.

```
<Button Background="Red" Content="Button!" />
```

XAML supporta diverse sintassi per l'impostazione delle proprietà. La sintassi da usare per una proprietà particolare dipende dal tipo di valore usato da una proprietà e da altri fattori, quali la presenza di un convertitore dei tipi. Per altre informazioni sulla sintassi XAML per l'impostazione di proprietà, vedere [Cenni preliminari su XAML \(WPF\)](#) e [Descrizione dettagliata della sintassi XAML](#).

Come esempio di sintassi senza attributi, nell'esempio di XAML seguente viene illustrato lo sfondo di un altro pulsante. Questa volta, invece di impostare un semplice colore a tinta unita, lo sfondo viene impostato su un'immagine, con un elemento che rappresenta tale immagine e la relativa origine, specificate come un attributo dell'elemento annidato. Si tratta di un esempio di sintassi per gli elementi proprietà.

```
<Button Content="Button!">
  <Button.Background>
    <ImageBrush ImageSource="wavy.jpg"/>
  </Button.Background>
</Button>
```

Impostazione delle proprietà nel codice

L'impostazione dei valori delle proprietà di dipendenza nel codice è in genere semplicemente una chiamata all'implementazione del set esposta dal "wrapper" CLR.

```
Button myButton = new Button();
myButton.Width = 200.0;
```

```
Dim myButton As New Button()
myButton.Width = 200.0
```

Anche il recupero di un valore della proprietà prevede sostanzialmente una chiamata all'implementazione del "wrapper" Get:

```
double whatWidth;
whatWidth = myButton.Width;
```

```
Dim whatWidth As Double
whatWidth = myButton.Width
```

È anche possibile chiamare le API del sistema di proprietà [GetValue](#) e [SetValue](#) direttamente. Questa operazione non è in genere necessaria se si usano proprietà esistenti (i wrapper sono più pratici e offrono una migliore esposizione della proprietà per gli strumenti di sviluppo), ma la chiamata diretta delle API è appropriata per determinati scenari.

È anche possibile impostare le proprietà in XAML e successivamente accedervi nel codice tramite code-behind. Per informazioni dettagliate, vedere [Code-behind e XAML in WPF](#).

Funzionalità offerte da una proprietà di dipendenza

Una proprietà di dipendenza fornisce funzionalità che consentono di estendere la funzionalità di una proprietà, rispetto a una proprietà supportata da un campo. Spesso tale funzionalità rappresenta o supporta una delle funzionalità specifiche seguenti:

- [Risorse](#)
- [Data binding](#)

- [Stili](#)
- [Animazioni](#)
- [Override dei metadati](#)
- [Ereditarietà del valore della proprietà](#)
- [Integrazione di WPF Designer](#)

Risorse

Un valore della proprietà di dipendenza può essere impostato facendo riferimento a una risorsa. Le risorse vengono in genere specificate come valore della proprietà `Resources` di un elemento radice della pagina o dell'applicazione (questi percorsi consentono un accesso più semplice alla risorsa). L'esempio seguente illustra come definire una risorsa [SolidColorBrush](#).

```
<DockPanel.Resources>
  <SolidColorBrush x:Key="MyBrush" Color="Gold"/>
</DockPanel.Resources>
```

Una volta definita, è possibile fare riferimento alla risorsa e usarla per fornire un valore della proprietà:

```
<Button Background="{DynamicResource MyBrush}" Content="I am gold" />
```

A questa risorsa particolare viene fatto riferimento come [estensione del markup DynamicResource](#). In XAML WPF è possibile usare un riferimento di risorsa statica o dinamica. Per usare un riferimento di risorsa dinamica, è necessario eseguire l'impostazione su una proprietà di dipendenza, in modo che il sistema di proprietà WPF attivi l'utilizzo specifico del riferimento di risorsa dinamica. Per altre informazioni, vedere [Risorse XAML](#).

NOTE

Le risorse vengono considerate come valore locale, per cui se si imposta un altro valore locale, il riferimento di risorsa sarà eliminato. Per altre informazioni, vedere [Precedenza del valore della proprietà di dipendenza](#).

Associazione dati

Una proprietà di dipendenza può fare riferimento a un valore tramite il data binding, che funziona tramite una sintassi per estensione di markup specifica in XAML o l'oggetto [Binding](#) nel codice. Con il data binding, la determinazione del valore della proprietà finale viene rinviata fino alla fase di esecuzione, momento in cui il valore viene ottenuto da un'origine dati.

L'esempio seguente imposta la proprietà [Content](#) per un oggetto [Button](#) tramite un'associazione dichiarata in XAML. L'associazione usa un contesto dei dati ereditato e un'origine dati [XmlDataProvider](#) (non illustrata). L'associazione stessa specifica la proprietà di origine desiderata mediante l'oggetto [XPath](#) all'interno dell'origine dati.

```
<Button Content="{Binding XPath=Team/@TeamName}"/>
```

NOTE

I binding vengono considerati come valore locale, per cui se si imposta un altro valore locale, il binding verrà eliminato. Per altri dettagli, vedere [Precedenza del valore della proprietà di dipendenza](#).

Le proprietà di dipendenza, o la classe [DependencyObject](#), non supportano [INotifyPropertyChanged](#) in modo nativo per la generazione di notifiche delle modifiche nel valore della proprietà di origine [DependencyObject](#) per le operazioni di associazione di dati. Per altre informazioni su come creare proprietà da usare nel data binding, che consentano di segnalare modifiche di una destinazione di data binding, vedere [Panoramica sul data binding](#).

Stili

Stili e modelli sono due degli scenari principali che giustificano l'uso delle proprietà di dipendenza. Gli stili sono particolarmente utili per l'impostazione di proprietà che definiscono l'interfaccia utente dell'applicazione. In genere gli stili vengono definiti come risorse in XAML e interagiscono con il sistema di proprietà in quanto di solito contengono "setter" per proprietà particolari, nonché "trigger" che modificano un valore della proprietà in base al valore in tempo reale per un'altra proprietà.

L'esempio seguente crea uno stile molto semplice (definito all'interno di un dizionario [Resources](#), non illustrato), e quindi lo applica direttamente alla proprietà [Style](#) di un oggetto [Button](#). Il setter all'interno dello stile imposta la proprietà [Background](#) di un oggetto [Button](#) con stile su verde.

```
<Style x:Key="GreenButtonStyle">
    <Setter Property="Control.Background" Value="Green"/>
</Style>
```

```
<Button Style="{StaticResource GreenButtonStyle}">I am green!</Button>
```

Per altre informazioni, vedere [Applicazione di stili e modelli](#).

Animations

Alle proprietà di dipendenza è possibile aggiungere un'animazione. Quando un'animazione viene applicata ed è in esecuzione, il valore a cui è stata aggiunta un'animazione opera a un livello di precedenza superiore rispetto a qualsiasi altro valore (ad esempio un valore locale) della proprietà.

L'esempio seguente aggiunge un'animazione alla proprietà [Background](#) di un oggetto [Button](#). Tecnicamente, l'animazione viene aggiunta alla proprietà [Background](#) tramite la sintassi degli elementi proprietà per specificare un [SolidColorBrush](#) vuoto come [Background](#) e quindi la proprietà [Color](#) dell'oggetto [SolidColorBrush](#) è la proprietà a cui viene direttamente applicata l'animazione.

```
<Button>I am animated
    <Button.Background>
        <SolidColorBrush x:Name="AnimBrush"/>
    </Button.Background>
    <Button.Triggers>
        <EventTrigger RoutedEvent="Button.Loaded">
            <BeginStoryboard>
                <Storyboard>
                    <ColorAnimation
                        Storyboard.TargetName="AnimBrush"
                        Storyboard.TargetProperty="(SolidColorBrush.Color)"
                        From="Red" To="Green" Duration="0:0:5"
                        AutoReverse="True" RepeatBehavior="Forever" />
                </Storyboard>
            </BeginStoryboard>
        </EventTrigger>
    </Button.Triggers>
</Button>
```

Per altre informazioni sull'animazione di proprietà, vedere [Cenni preliminari sull'animazione](#) e [Cenni preliminari sugli storyboard](#).

Override dei metadati

È possibile modificare determinati comportamenti di una proprietà di dipendenza eseguendo l'override dei metadati per quella proprietà, quando si deriva dalla classe che registra originariamente la proprietà di dipendenza. L'override dei metadati si basa sull'identificatore [DependencyProperty](#). L'override dei metadati non richiede una nuova implementazione della proprietà. La modifica dei metadati viene gestita in modo nativo dal sistema di proprietà. Ogni classe contiene, potenzialmente, i metadati specifici per tutte le proprietà ereditate dalle classi di base, in base al tipo.

L'esempio seguente esegue l'override dei metadati per una proprietà di dipendenza [DefaultStyleKey](#).

L'override dei metadati di questa particolare proprietà di dipendenza fa parte di un modello di implementazione che consente di creare controlli che possono usare stili predefiniti dai temi.

```
public class SpinnerControl : ItemsControl
{
    static SpinnerControl()
    {
        DefaultStyleKeyProperty.OverrideMetadata(
            typeof(SpinnerControl),
            new FrameworkPropertyMetadata(typeof(SpinnerControl))
        );
    }
}
```

```
Public Class SpinnerControl
    Inherits ItemsControl
    Shared Sub New()
        DefaultStyleKeyProperty.OverrideMetadata(GetType(SpinnerControl), New
FrameworkPropertyMetadata(GetType(SpinnerControl)))
    End Sub
End Class
```

Per altre informazioni sull'override dei metadati delle proprietà o su come ottenere i metadati delle proprietà, vedere [Metadati delle proprietà di dipendenza](#).

Ereditarietà del valore della proprietà

Un elemento può ereditare il valore di una proprietà di dipendenza dal relativo elemento padre nell'albero di oggetti.

NOTE

Il comportamento dell'ereditarietà del valore della proprietà non viene abilitato a livello globale per tutte le proprietà di dipendenza, poiché il tempo di calcolo per l'ereditarietà influisce negativamente sulle prestazioni. In genere, l'ereditarietà del valore della proprietà viene abilitata solo per le proprietà in cui uno scenario particolare suggerisce che tale ereditarietà è appropriata. La possibilità di ereditare di una proprietà di dipendenza può essere determinata consultando la sezione **Informazioni proprietà di dipendenza** per una determinata proprietà di dipendenza, nel riferimento SDK.

L'esempio seguente descrive un'associazione e imposta la proprietà [DataContext](#) che specifica l'origine dell'associazione, non illustrata nell'esempio di associazione precedente. Eventuali associazioni successive in oggetti figlio non devono necessariamente specificare l'origine, ma possono usare il valore ereditato da [DataContext](#) nell'oggetto [StackPanel](#) padre. In alternativa, un oggetto figlio può invece scegliere di specificare direttamente il proprio [DataContext](#) o un [Source](#) in [Binding](#) e non usare deliberatamente il valore ereditato per il contesto dei dati delle sue associazioni.

```
<StackPanel Canvas.Top="50" DataContext="{Binding Source={StaticResource XmlTeamsSource}}">
    <Button Content="{Binding XPath=Team/@TeamName}"/>
</StackPanel>
```

Per altre informazioni, vedere [Ereditarietà del valore della proprietà](#).

Integrazione di WPF Designer

Un controllo personalizzato con proprietà implementate come proprietà di dipendenza riceverà la finestra di progettazione WPF appropriata per il supporto di Visual Studio. Un esempio è rappresentato dalla capacità di modificare proprietà di dipendenza dirette e associate tramite la finestra **Proprietà**. Per altre informazioni, vedere [Cenni preliminari sulla modifica di controlli](#).

Precedenza del valore della proprietà di dipendenza

Quando si ottiene il valore di una proprietà di dipendenza, si ottiene potenzialmente un valore impostato per tale proprietà tramite uno qualsiasi degli altri input basati su proprietà che fanno parte del sistema di proprietà WPF. La precedenza del valore della proprietà di dipendenza consente interazioni prevedibili per un'ampia gamma di scenari relativi al modo in cui le proprietà ottengono i rispettivi valori.

Si osservi l'esempio riportato di seguito. L'esempio include uno stile che si applica a tutti i pulsanti e alle loro proprietà **Background**, ma specifica anche un pulsante con un valore **Background** impostato localmente.

NOTE

Nella documentazione SDK i termini "valore locale" o "valore impostato localmente" vengono usati talvolta per la descrizione delle proprietà di dipendenza. Un valore impostato localmente è un valore di proprietà che viene impostato direttamente in un'istanza di oggetto nel codice o come attributo di un elemento in XAML.

In teoria, per il primo pulsante la proprietà viene impostata due volte, ma viene applicato un solo valore, quello con la precedenza più alta. Un valore impostato localmente ha la massima precedenza (eccetto per un'animazione in esecuzione; tuttavia, in questo esempio non viene applicata nessuna animazione), pertanto per lo sfondo del primo pulsante viene usato il valore impostato localmente invece del valore del setter di stile. Il secondo pulsante non ha un valore locale (e nessun altro valore con precedenza più alta rispetto a un setter di stile), pertanto lo sfondo di quel pulsante proviene dal setter di stile.

```
<StackPanel>
    <StackPanel.Resources>
        <Style x:Key="{x:Type Button}" TargetType="{x:Type Button}">
            <Setter Property="Background" Value="Red"/>
        </Style>
    </StackPanel.Resources>
    <Button Background="Green">I am NOT red!</Button>
    <Button>I am styled red</Button>
</StackPanel>
```

Motivi dell'esistenza della precedenza delle proprietà di dipendenza

In genere, non si vuole che gli stili vengano sempre applicati e che nascondano persino un valore impostato localmente di un singolo elemento, altrimenti sarebbe molto difficile usare gli stili o gli elementi in generale. Per questo motivo, i valori che provengono dagli stili operano a un livello di precedenza inferiore rispetto a un valore impostato localmente. Per un elenco più completo delle proprietà di dipendenza e informazioni sulla possibile provenienza di un valore effettivo di una proprietà di dipendenza, vedere [Precedenza del valore della proprietà di dipendenza](#).

NOTE

Molte proprietà definite negli elementi WPF non sono proprietà di dipendenza. In generale, le proprietà vengono implementate come proprietà di dipendenza solo quando è necessario supportare almeno uno degli scenari abilitati dal sistema di proprietà: data binding, applicazione degli stili, animazione, supporto del valore predefinito, ereditarietà, proprietà associate o invalidamento.

Altre informazioni sulle proprietà di dipendenza

- Una proprietà associata è un tipo di proprietà che supporta una sintassi specializzata in XAML. Una proprietà associata spesso non ha una corrispondenza 1:1 con una proprietà Common Language Runtime (CLR) e non è necessariamente una proprietà di dipendenza. Lo scopo tipico di una proprietà associata consiste nel consentire agli elementi figlio di segnalare i valori della proprietà a un elemento padre, anche se quest'ultimo e l'elemento figlio non possiedono tale proprietà, come parte degli elenchi dei membri della classe. Uno scenario principale è l'abilitazione di elementi figlio per informare l'elemento padre di come devono essere presentati in Interfaccia utente. Per un esempio, vedere [Dock](#) o [Left](#). Per informazioni dettagliate, vedere [Cenni preliminari sulle proprietà associate](#).
- Gli sviluppatori di componenti o di applicazioni possono decidere di creare una proprietà di dipendenza personalizzata al fine di abilitare funzionalità quali il data binding o il supporto degli stili oppure per il supporto dell'invalidamento e della coercizione del valore. Per informazioni dettagliate, vedere [Proprietà di dipendenza personalizzate](#).
- Generalmente, le proprietà di dipendenza devono essere considerate come proprietà pubbliche, accessibili o almeno individuabili da parte di qualsiasi chiamante con accesso a un'istanza. Per altre informazioni, vedere [Sicurezza delle proprietà di dipendenza](#).

Vedere anche

- [Proprietà di dipendenza personalizzate](#)
- [Proprietà di dipendenza di sola lettura](#)
- [Cenni preliminari su XAML \(WPF\)](#)
- [Architettura WPF](#)

Cenni preliminari sulle proprietà associate

25/11/2019 • 21 minutes to read • [Edit Online](#)

Una proprietà associata è un concetto definito da XAML. Una proprietà associata deve essere usata come un tipo di proprietà globale che è possibile impostare su qualsiasi oggetto. In Windows Presentation Foundation (WPF) le proprietà associate sono in genere definite come un tipo specializzato di proprietà di dipendenza che non dispone della proprietà "wrapper" convenzionale.

Prerequisiti

Questo argomento presuppone la conoscenza delle proprietà di dipendenza dal punto di vista di un consumer delle proprietà di dipendenza esistenti nelle classi di Windows Presentation Foundation (WPF), nonché la lettura dell'argomento [Panoramica sulle proprietà di dipendenza](#). Per seguire gli esempi in questo argomento, è necessario comprendere anche XAML e sapere come scrivere applicazioni WPF.

Perché usare le proprietà associate

Una delle finalità di una proprietà associata consiste nel consentire a diversi elementi figlio di specificare valori univoci di una proprietà effettivamente definita in un elemento padre. Un'applicazione specifica di questo scenario consente agli elementi figlio di notificare all'elemento padre la modalità con cui devono essere presentati nell'interfaccia utente. Un esempio è la proprietà [DockPanel.Dock](#). La proprietà [DockPanel.Dock](#) viene creata come proprietà associata perché è progettata per essere impostata su elementi contenuti all'interno di un [DockPanel](#), anziché su [DockPanel](#) stessa. La classe [DockPanel](#) definisce il campo [DependencyProperty](#) statico denominato [DockProperty](#), quindi fornisce i metodi [GetDock](#) e [SetDock](#) come funzioni di accesso pubbliche per la proprietà associata.

Proprietà associate in XAML

In XAML, è possibile impostare le proprietà associate usando la sintassi
ProviderProprietàAssociata.NomeProprietà

Di seguito è riportato un esempio di come è possibile impostare [DockPanel.Dock](#) in XAML:

```
<DockPanel>
  <CheckBox DockPanel.Dock="Top">Hello</CheckBox>
</DockPanel>
```

Si noti che l'utilizzo è piuttosto simile a una proprietà statica. si fa sempre riferimento al tipo [DockPanel](#) proprietario e si registra la proprietà associata, anziché fare riferimento a qualsiasi istanza specificata in base al nome.

Inoltre, dato che una proprietà associata in XAML è un attributo che viene impostato nel markup, solo l'operazione di impostazione ha una certa rilevanza. Non è possibile ottenere direttamente una proprietà in XAML, sebbene esistano alcuni meccanismi indiretti per confrontare i valori, ad esempio i trigger negli stili. Per altri dettagli, vedere [Applicazione di stili e modelli](#).

Implementazione delle proprietà associate in WPF

In Windows Presentation Foundation (WPF), la maggior parte delle proprietà associate presenti nei tipi WPF correlati alla presentazione dell'interfaccia utente vengono implementate come proprietà di dipendenza. Le proprietà associate sono un concetto XAML, mentre le proprietà di dipendenza sono un concetto WPF. Poiché le

proprietà associate a WPF sono proprietà di dipendenza, supportano concetti della proprietà di dipendenza quali i metadati della proprietà e i valori predefiniti dei metadati della proprietà.

Modalità di utilizzo delle proprietà associate da parte del tipo proprietario

Anche se le proprietà associate possono essere impostate per qualsiasi oggetto, ciò non significa che l'impostazione della proprietà produce un risultato tangibile o che il valore verrà usato da un altro oggetto. In genere, le proprietà associate sono concepite in modo che gli oggetti provenienti da un'ampia varietà di possibili gerarchie di classi o di relazioni logiche possano ciascuno riportare informazioni comuni al tipo che definisce la proprietà associata. Il tipo che definisce la proprietà associata si attiene di regola a uno di questi modelli:

- Il tipo che definisce la proprietà associata è progettato in modo da poter essere l'elemento padre degli elementi che imposteranno i valori per la proprietà associata. Il tipo scorre quindi gli oggetti figlio attraverso la logica interna in base a una struttura ad albero degli oggetti, ottiene i valori e agisce in qualche modo su tali valori.
- Il tipo che definisce la proprietà associata verrà usato come elemento figlio per un'ampia gamma di elementi padre e modelli di contenuto possibili.
- Il tipo che definisce la proprietà associata rappresenta un servizio. Gli altri tipi impostano i valori per la proprietà associata. Pertanto, quando l'elemento che impone la proprietà viene valutato nel contesto del servizio, i valori della proprietà associata vengono ottenuti tramite la logica interna della classe del servizio.

Esempio di proprietà associata definita dall'elemento padre

Lo scenario più comune in cui WPF definisce una proprietà associata è quando un elemento padre supporta una raccolta di elementi figlio e implementa anche un comportamento in cui le specifiche del comportamento vengono segnalate singolarmente per ogni elemento figlio.

[DockPanel](#) definisce la proprietà associata [DockPanel.Dock](#) e [DockPanel](#) dispone di codice a livello di classe come parte della logica di rendering (in particolare, [MeasureOverride](#) e [ArrangeOverride](#)). Un'istanza di [DockPanel](#) verificherà sempre se uno degli elementi figlio immediati ha impostato un valore per [DockPanel.Dock](#). In questo caso, tali valori diventano l'input per la logica di rendering applicata a quel particolare elemento figlio. Le istanze di [DockPanel](#) annidate considerano ciascuna le raccolte di elementi figlio immediati, ma tale comportamento è specifico dell'implementazione per il modo in cui [DockPanel](#) elabora [DockPanel.Dock](#) i valori. In teoria, è possibile che alcune proprietà associate abbiano effetto su elementi più distanti dell'elemento padre immediato. Se la [DockPanel.Dock](#) proprietà associata è impostata su un elemento senza [DockPanel](#) elemento padre su cui agire, non viene generato alcun errore o eccezione. Ciò significa semplicemente che è stato impostato un valore di proprietà globale, ma non dispone di [DockPanel](#) padre corrente che potrebbe utilizzare le informazioni.

Proprietà associate nel codice

Le proprietà associate in WPF non dispongono dei tipici metodi "wrapper" di CLR per un accesso semplice di tipo get/set. Ciò è dovuto al fatto che la proprietà associata non appartiene necessariamente allo spazio dei nomi CLR per le istanze in cui è impostata la proprietà. Tuttavia, un processore XAML deve essere in grado di impostare tali valori quando il codice XAML viene analizzato. Per supportare un utilizzo efficace delle proprietà associate, il tipo di proprietario della proprietà associata deve implementare metodi delle funzioni di accesso dedicate nel formato **Get_PropertyName_ e Set_PropertyName_**. Questi metodi della funzione di accesso dedicati sono anche utili per ottenere o impostare la proprietà associata nel codice. Dal punto di vista del codice, una proprietà associata è simile a un campo sottostante che dispone di funzioni di accesso ai metodi anziché di funzioni di accesso alle proprietà e tale campo sottostante può trovarsi in qualsiasi oggetto senza che sia necessario definirlo in modo specifico.

L'esempio seguente illustra come impostare una proprietà associata nel codice. In questo esempio `myCheckBox` è

un'istanza della classe [CheckBox](#).

```
DockPanel myDockPanel = new DockPanel();
CheckBox myCheckBox = new CheckBox();
myCheckBox.Content = "Hello";
myDockPanel.Children.Add(myCheckBox);
DockPanel.SetDock(myCheckBox, Dock.Top);
```

```
Dim myDockPanel As New DockPanel()
Dim myCheckBox As New CheckBox()
myCheckBox.Content = "Hello"
myDockPanel.Children.Add(myCheckBox)
DockPanel.SetDock(myCheckBox, Dock.Top)
```

Analogamente al caso XAML, se `myCheckBox` non è già stato aggiunto come elemento figlio di `myDockPanel` dalla terza riga di codice, la quarta riga di codice non genera un'eccezione, ma il valore della proprietà non interagisce con un elemento padre [DockPanel](#) e pertanto lo farebbe Nothing. Solo un valore [DockPanel.Dock](#) impostato su un elemento figlio combinato con la presenza di un elemento padre [DockPanel](#) provocherà un comportamento effettivo nell'applicazione sottoposta a rendering. In questo caso, è possibile impostare la proprietà associata e quindi effettuare l'associazione alla struttura ad albero. In alternativa, è possibile effettuare l'associazione alla struttura ad albero, quindi impostare la proprietà associata. Qualunque sia l'ordine delle azioni, il risultato è il medesimo.

Metadati delle proprietà associate

Quando si registra la proprietà, [FrameworkPropertyMetadata](#) viene impostato per specificare le caratteristiche della proprietà, ad esempio se la proprietà influisca sul rendering, sulla misurazione e così via. I metadati di una proprietà associata non sono in genere diversi rispetto a quelli presenti per una proprietà di dipendenza. Se si specifica un valore predefinito in un override per i metadati di una proprietà associata, tale valore diventa il valore predefinito della proprietà associata implicita nelle istanze della classe che esegue l'override. In particolare, il valore predefinito viene segnalato se un processo esegue una query per recuperare il valore di una proprietà associata tramite la funzione di accesso al metodo `Get` di quella proprietà, specificando un'istanza della classe in cui sono stati definiti i metadati e il valore per quella proprietà collegata non è stato impostato diversamente.

Se si vuole abilitare l'ereditarietà del valore di una proprietà, è necessario usare le proprietà associate anziché le proprietà di dipendenza non associate. Per informazioni dettagliate, vedere [Ereditarietà del valore della proprietà](#).

Proprietà associate personalizzate

Quando creare una proprietà associata

È possibile creare una proprietà associata quando è necessario disporre di un meccanismo di impostazione delle proprietà per le classi diverse dalla classe di definizione. In questo caso, lo scenario più comune è il layout. Esempi di proprietà di layout esistenti sono [DockPanel.Dock](#), [Panel.ZIndex](#) e [Canvas.Top](#). Nello scenario abilitato in questo contesto, gli elementi disponibili come elementi figlio degli elementi di controllo del layout sono in grado di indicare i requisiti del layout agli elementi padre del layout singolarmente, impostando ciascuno un valore della proprietà definito dall'elemento padre come proprietà associata.

Un altro scenario per l'uso di una proprietà associata è quello in cui la classe rappresenta un servizio e si vuole che le classi siano in grado di integrare il servizio in modo più trasparente.

Un altro scenario è la ricezione del supporto di progettazione WPF di Visual Studio, ad esempio la modifica della finestra **Proprietà**. Per altre informazioni, vedere [Cenni preliminari sulla modifica di controlli](#).

Come indicato in precedenza, è necessario eseguire la registrazione come proprietà associata se si vuole usare

l'ereditarietà del valore della proprietà.

Come creare una proprietà associata

Se la classe definisce la proprietà associata esclusivamente per l'uso in altri tipi, la classe non deve derivare da [DependencyObject](#). Tuttavia, è necessario derivare da [DependencyObject](#) se si segue il modello WPF generale di che la proprietà associata sia anche una proprietà di dipendenza.

Definire la proprietà associata come proprietà di dipendenza dichiarando un campo `public static readonly` di tipo [DependencyProperty](#). Questo campo viene definito usando il valore restituito del metodo [RegisterAttached](#). Il nome del campo deve corrispondere al nome della proprietà associata, aggiunto con la stringa `Property`, per seguire il modello WPF stabilito di denominazione dei campi di identificazione rispetto alle proprietà che rappresentano. Il provider di proprietà associate deve fornire anche metodi di [Get_PropertyName_](#) e [Set_PropertyName_](#) statici come funzioni di accesso per la proprietà associata. In caso contrario, il sistema di proprietà non potrà utilizzare la proprietà associata.

NOTE

Se si omette la funzione di accesso get della proprietà associata, data binding nella proprietà non funzionerà negli strumenti di progettazione, ad esempio Visual Studio e Blend per Visual Studio.

Funzione di accesso Get

La firma per la funzione di accesso [Get_PropertyName_](#) deve essere:

```
public static object GetPropertyName(object target)
```

- L'oggetto `target` può essere specificato come tipo più specifico nell'implementazione. Il metodo [DockPanel.GetDock](#), ad esempio, digita il parametro come [UIElement](#), perché la proprietà associata è destinata solo a essere impostata su [UIElement](#) istanze.
- Il valore restituito può essere specificato come tipo più specifico nell'implementazione. Ad esempio, il metodo [GetDock](#) lo digita come [Dock](#), perché il valore può essere impostato solo su tale enumerazione.

Funzione di accesso Set

La firma per la funzione di accesso [Set_PropertyName_](#) deve essere:

```
public static void SetPropertyName(object target, object value)
```

- L'oggetto `target` può essere specificato come tipo più specifico nell'implementazione. Ad esempio, il metodo [SetDock](#) lo digita come [UIElement](#), perché la proprietà associata è destinata solo a essere impostata per [UIElement](#) istanze.
- L'oggetto `value` può essere specificato come tipo più specifico nell'implementazione. Ad esempio, il metodo [SetDock](#) lo digita come [Dock](#), perché il valore può essere impostato solo su tale enumerazione. Tenere presente che il valore per questo metodo è l'input proveniente dal caricatore XAML quando rileva la proprietà associata in un utilizzo della proprietà associata nel markup. Tale input è il valore specificato come valore di attributo XAML nel markup. Pertanto, per il tipo usato devono essere disponibili la conversione di tipo, il serializzatore del valore o il supporto per l'estensione di markup, in modo da poter creare il tipo appropriato in base al valore dell'attributo, rappresentato in pratica semplicemente da una stringa.

Nell'esempio seguente viene illustrata la registrazione della proprietà di dipendenza (tramite il metodo [RegisterAttached](#)), nonché le funzioni di accesso [Get_PropertyName_](#) e [Set_PropertyName_](#). Nell'esempio, la proprietà associata è denominata `IsBubbleSource`. Pertanto, le funzioni di accesso devono essere chiamate `GetIsBubbleSource` e `SetIsBubbleSource`.

```

public static readonly DependencyProperty IsBubbleSourceProperty = DependencyProperty.RegisterAttached(
    "IsBubbleSource",
    typeof(Boolean),
    typeof(AquariumObject),
    new FrameworkPropertyMetadata(false, FrameworkPropertyMetadataOptions.AffectsRender)
);
public static void SetIsBubbleSource(UIElement element, Boolean value)
{
    element.SetValue(IsBubbleSourceProperty, value);
}
public static Boolean GetIsBubbleSource(UIElement element)
{
    return (Boolean)element.GetValue(IsBubbleSourceProperty);
}

```

```

Public Shared ReadOnly IsBubbleSourceProperty As DependencyProperty =
DependencyProperty.RegisterAttached("IsBubbleSource", GetType(Boolean), GetType(AquariumObject), New
FrameworkPropertyMetadata(False, FrameworkPropertyMetadataOptions.AffectsRender))
Public Shared Sub SetIsBubbleSource(ByVal element As UIElement, ByVal value As Boolean)
    element.SetValue(IsBubbleSourceProperty, value)
End Sub
Public Shared Function GetIsBubbleSource(ByVal element As UIElement) As Boolean
    Return CType(element.GetValue(IsBubbleSourceProperty), Boolean)
End Function

```

Attributi delle proprietà associate

WPF definisce diversi attributi .NET che hanno lo scopo di fornire informazioni sulle proprietà associate ai processi di reflection e agli utenti tipici delle informazioni sulla reflection e sulle proprietà, ad esempio le finestre di progettazione. Dato che le proprietà associate hanno un tipo con ambito illimitato, le finestre di progettazione devono disporre di un modo per evitare di sopraffare gli utenti con un elenco globale di tutte le proprietà associate definite in una particolare implementazione della tecnologia che usa XAML. Gli attributi .NET definiti da WPF per le proprietà associate possono essere utilizzati per definire l'ambito delle situazioni in cui una determinata proprietà associata deve essere visualizzata in una finestra delle proprietà. È possibile prendere in considerazione l'applicazione di questi attributi anche per le proprietà associate personalizzate. Lo scopo e la sintassi degli attributi .NET sono descritti nelle pagine di riferimento appropriate:

- [AttachedPropertyBrowsableAttribute](#)
- [AttachedPropertyBrowsableForChildrenAttribute](#)
- [AttachedPropertyBrowsableForTypeAttribute](#)
- [AttachedPropertyBrowsableWhenAttributePresentAttribute](#)

Ulteriori informazioni sulle proprietà associate

- Per altre informazioni sulla creazione di una proprietà associata, vedere [Registrare una proprietà associata](#).
- Per scenari di utilizzo più avanzati delle proprietà di dipendenza e delle proprietà associate, vedere [Proprietà di dipendenza personalizzate](#).
- È anche possibile registrare una proprietà come proprietà associata e proprietà di dipendenza, ma continuare a esporre le implementazioni del "wrapper". In questo caso, la proprietà può essere impostata in tale elemento o in qualsiasi elemento tramite la sintassi per le proprietà associate XAML. Un esempio di proprietà con uno scenario appropriato per gli utilizzi standard e collegati è [FrameworkElement.FlowDirection](#).

Vedere anche

- [DependencyProperty](#)
- [Panoramica sulle proprietà di dipendenza](#)
- [Proprietà di dipendenza personalizzate](#)
- [Cenni preliminari su XAML \(WPF\)](#)
- [Registrare una proprietà associata](#)

Proprietà di dipendenza personalizzate

08/11/2019 • 30 minutes to read • [Edit Online](#)

Questo argomento descrive le ragioni per cui sviluppatori di applicazioni Windows Presentation Foundation (WPF) e autori di componenti potrebbero decidere di creare una proprietà di dipendenza personalizzata e illustra i passaggi dell'implementazione oltre ad alcune opzioni di implementazione in grado di migliorare le prestazioni, l'usabilità o la versatilità della proprietà.

Prerequisites

In questo argomento si presuppongono la conoscenza delle proprietà di dipendenza dal punto di vista di un consumer di proprietà di dipendenza esistenti nelle classi WPF, nonché la lettura dell'argomento [Panoramica sulle proprietà di dipendenza](#). Per seguire gli esempi illustrati in questo argomento, è anche necessario conoscere Extensible Application Markup Language (XAML) e saper scrivere applicazioni WPF.

Che cos'è una proprietà di dipendenza?

È possibile abilitare ciò che altrimenti sarebbe una proprietà Common Language Runtime (CLR) per supportare lo stile, la data binding, l'ereditarietà, le animazioni e i valori predefiniti implementando tale proprietà come proprietà di dipendenza. Le proprietà di dipendenza sono proprietà registrate con il sistema di proprietà WPF chiamando il metodo di [Register](#) (o [RegisterReadOnly](#)) e supportate da un campo dell'identificatore [DependencyProperty](#). Le proprietà di dipendenza possono essere utilizzate solo da tipi di [DependencyObject](#), ma [DependencyObject](#) è piuttosto elevato nella gerarchia di classi WPF, quindi la maggior parte delle classi disponibili in WPF può supportare le proprietà di dipendenza. Per altre informazioni sulle proprietà di dipendenza e alcune delle terminologie e delle convenzioni usate per descriverle in questo SDK, vedere [Cenni preliminari sulle proprietà di dipendenza](#).

Esempio di proprietà di dipendenza

Esempi di proprietà di dipendenza implementate nelle classi WPF includono la proprietà [Background](#), la proprietà [Width](#) e la proprietà [Text](#), tra le molte altre. Ogni proprietà di dipendenza esposta da una classe dispone di un campo statico pubblico corrispondente di tipo [DependencyProperty](#) esposto sulla stessa classe. Si tratta dell'identificatore per la proprietà di dipendenza. L'identificatore viene denominato mediante la seguente convenzione: nome della proprietà di dipendenza seguito dalla stringa `Property`. Ad esempio, il campo identificatore [DependencyProperty](#) corrispondente per la proprietà [Background](#) è [BackgroundProperty](#). L'identificatore archivia le informazioni sulla proprietà di dipendenza mentre è stata registrata e l'identificatore viene quindi usato in un secondo momento per altre operazioni che coinvolgono la proprietà di dipendenza, ad esempio la chiamata di [SetValue](#).

Come indicato nella [Panoramica sulle proprietà di dipendenza](#), tutte le proprietà di dipendenza in WPF (eccetto la maggior parte delle proprietà associate) sono anche proprietà CLR a causa dell'implementazione "wrapper". Pertanto, dal codice, è possibile ottenere o impostare le proprietà di dipendenza chiamando funzioni di accesso CLR che definiscono i wrapper nello stesso modo in cui si utilizzeranno altre proprietà CLR. In qualità di consumer di proprietà di dipendenza stabilite, in genere non si utilizzano i metodi di [DependencyObject GetValue](#) e [SetValue](#), ovvero il punto di connessione al sistema di proprietà sottostante. Piuttosto, l'implementazione esistente delle proprietà CLR avrà già chiamato [GetValue](#) e [SetValue](#) all'interno delle implementazioni di `get` e `set` wrapper della proprietà, usando il campo dell'identificatore in modo appropriato. Se l'implementazione di una proprietà di dipendenza personalizzata viene eseguita in modo autonomo, il wrapper verrà definito in modo simile.

Quando implementare una proprietà di dipendenza

Quando si implementa una proprietà in una classe, purché la classe derivi da [DependencyObject](#), è possibile eseguire il backup della proprietà con un identificatore di [DependencyProperty](#) e quindi impostarla come proprietà di dipendenza. Trasformare una proprietà in una proprietà di dipendenza non sempre è necessario o appropriato e dipende dalle esigenze dello scenario. Talvolta è opportuno usare la tecnica normale che consiste nel supportare la proprietà con un campo privato. Tuttavia, se si vuole che la proprietà supporti una o più delle seguenti funzionalità WPF, è necessario implementarla come proprietà di dipendenza:

- Si vuole che la proprietà possa essere impostata in uno stile. Per altre informazioni, vedere [Applicazione di stili e modelli](#).
- Si vuole che la proprietà supporti il data binding. Per altre informazioni sulle proprietà di dipendenza di data binding, vedere [Eseguire l'associazione delle proprietà di due controlli](#).
- Si vuole che la proprietà possa essere impostata con un riferimento di risorsa dinamica. Per altre informazioni, vedere [Risorse XAML](#).
- Si vuole ereditare un valore di proprietà automaticamente da un elemento padre nell'albero degli elementi. In questo caso, eseguire la registrazione con il metodo [RegisterAttached](#), anche se si crea anche un wrapper di proprietà per l'accesso a CLR. Per altre informazioni, vedere [Ereditarietà del valore della proprietà](#).
- Si vuole che la proprietà sia animata. Per altre informazioni, vedere [Panoramica dell'animazione](#).
- Si vuole che il sistema di proprietà segnali quando il valore precedente della proprietà è stato modificato da azioni intraprese dal sistema di proprietà, dall'ambiente o dall'utente oppure tramite la lettura e l'uso degli stili. Se si usano i metadati della proprietà, la proprietà può specificare un metodo di callback che verrà richiamato ogni volta il sistema di proprietà determina che il valore della proprietà è stato modificato definitivamente. Un concetto correlato è la coercizione del valore di proprietà. Per altre informazioni, vedere [Callback e convalida delle proprietà di dipendenza](#).
- Si vogliono usare convenzioni di metadati definite usate anche dai processi WPF, ad esempio la segnalazione dell'eventualità che la modifica di un valore di proprietà richieda che il sistema di layout ricomponga gli elementi visivi di un elemento. Oppure si vogliono usare override di metadati in modo che le classi derivate possano modificare caratteristiche basate sui metadati quali il valore predefinito.
- Si desidera che le proprietà di un controllo personalizzato ricevano il supporto di progettazione WPF di Visual Studio, ad esempio la modifica della finestra **Proprietà**. Per altre informazioni, vedere [Cenni preliminari sulla modifica di controlli](#).

Quando si esaminano questi scenari, è necessario considerare anche se è possibile realizzare lo scenario eseguendo l'override dei metadati di una proprietà di dipendenza esistente, piuttosto che implementando una proprietà completamente nuova. La praticità di un override di metadati dipende dallo scenario e dalla somiglianza di tale scenario con l'implementazione nelle proprietà di dipendenza WPF esistenti e nelle classi. Per altre informazioni sull'override dei metadati nelle proprietà esistenti, vedere [Metadati delle proprietà di dipendenza](#).

Elenco di controllo per la definizione di una proprietà di dipendenza

La definizione di una proprietà di dipendenza include quattro concetti distinti. Questi concetti non rappresentano necessariamente passaggi rigidi di una procedura, in quanto alcuni di questi finiscono per essere combinati come singole righe di codice nell'implementazione:

- (Facoltativo) Creare metadati di proprietà per la proprietà di dipendenza.
- Registrare il nome della proprietà con il sistema di proprietà, specificando un tipo di proprietario e il tipo del valore di proprietà. Se usati, specificare anche i metadati della proprietà.
- Definire un identificatore `DependencyProperty` come `public` campo `static` `readonly` sul tipo di proprietario.
- Definire una proprietà "wrapper" CLR il cui nome corrisponda al nome della proprietà di dipendenza. Implementare le funzioni di accesso `get` e `set` della proprietà "wrapper" CLR per connettersi alla proprietà di dipendenza che la esegue.

Registrazione della proprietà nel sistema di proprietà

Affinché la proprietà sia una proprietà di dipendenza, è necessario registrarla in una tabella gestita dal sistema di proprietà e assegnarle un identificatore univoco usato come qualificatore per le successive operazioni del sistema di proprietà. Queste operazioni possono essere operazioni interne o il codice che chiama le API del sistema di proprietà. Per registrare la proprietà, chiamare il metodo `Register` all'interno del corpo della classe (all'interno della classe, ma all'esterno di qualsiasi definizione di membro). Il campo identificatore viene inoltre fornito dalla chiamata al metodo `Register` come valore restituito. Il motivo per cui la chiamata `Register` viene eseguita al di fuori delle altre definizioni dei membri è perché si usa questo valore restituito per assegnare e creare un `public` `static` `readonly` campo di tipo `DependencyProperty` come parte della classe. Questo campo diventa l'identificatore per la proprietà di dipendenza.

```
public static readonly DependencyProperty AquariumGraphicProperty = DependencyProperty.Register(
    "AquariumGraphic",
    typeof(Uri),
    typeof(AquariumObject),
    new FrameworkPropertyMetadata(null,
        FrameworkPropertyMetadataOptions.AffectsRender,
        new PropertyChangedCallback(OnUriChanged)
    )
);
```

```
Public Shared ReadOnly AquariumGraphicProperty As DependencyProperty =
DependencyProperty.Register("AquariumGraphic", GetType(Uri), GetType(AquariumObject), New
FrameworkPropertyMetadata(Nothing, FrameworkPropertyMetadataOptions.AffectsRender, New
PropertyChangedCallback(AddressOf OnUriChanged)))
```

Convenzioni di denominazione delle proprietà di dipendenza

Esistono convenzioni di denominazione definite relative alle proprietà di dipendenza che è necessario seguire in tutte le circostanze tranne in casi eccezionali.

La proprietà di dipendenza stessa avrà un nome di base, "AquariumGraphic", come in questo esempio, fornito come primo parametro di `Register`. Tale nome deve essere univoco all'interno di ogni tipo di registrazione. Le proprietà di dipendenza ereditate tramite tipi di base sono considerate già parte del tipo di registrazione. I nomi delle proprietà ereditate non possono essere registrati nuovamente. Tuttavia, esiste una tecnica per aggiungere una classe come proprietario di una proprietà di dipendenza persino quando quella proprietà di dipendenza non è ereditata. Per informazioni dettagliate, vedere [Metadati delle proprietà di dipendenza](#).

Quando si crea il campo dell'identificatore, denominare questo campo con il nome della proprietà registrata, più il suffisso `Property`. Questo campo è l'identificatore della proprietà di dipendenza e verrà usato in un secondo momento come input per il `SetValue` e `GetValue` le chiamate che verranno effettuate nei wrapper, da qualsiasi altro codice che accede alla proprietà tramite il proprio codice, da qualsiasi accesso al codice esterno consentito dal sistema di proprietà e potenzialmente da XAML processor.

NOTE

La definizione della proprietà di dipendenza nel corpo della classe costituisce l'implementazione tipica, ma è anche possibile definire una proprietà di dipendenza nel costruttore statico della classe. Questo approccio potrebbe essere utile nel caso in cui siano necessarie più righe di codice per inizializzare la proprietà di dipendenza.

Implementazione del "wrapper"

L'implementazione del wrapper deve chiamare [GetValue](#) nell'implementazione di `get` e [SetValue](#) nell'implementazione di `set` (la chiamata di registrazione originale e il campo vengono visualizzati anche per maggiore chiarezza).

In circostanze eccezionali, le implementazioni del wrapper devono eseguire solo le azioni [GetValue](#) e [SetValue](#), rispettivamente. La ragione è discussa nell'argomento [Caricamento XAML e proprietà di dipendenza](#).

Tutte le proprietà di dipendenza pubbliche esistenti fornite nelle classi WPF usano questo semplice modello di implementazione del wrapper. La maggior parte della complessità della modalità di funzionamento delle proprietà di dipendenza è dovuta a un comportamento intrinseco del sistema di proprietà oppure è implementata tramite altri concetti quali la coercizione o i callback di modifica della proprietà mediante i metadati della proprietà.

```
public static readonly DependencyProperty AquariumGraphicProperty = DependencyProperty.Register(
    "AquariumGraphic",
    typeof(Uri),
    typeof(AquariumObject),
    new FrameworkPropertyMetadata(null,
        FrameworkPropertyMetadataOptions.AffectsRender,
        new PropertyChangedCallback(OnUriChanged)
    )
);
public Uri AquariumGraphic
{
    get { return (Uri)GetValue(AquariumGraphicProperty); }
    set { SetValue(AquariumGraphicProperty, value); }
}
```

```
Public Shared ReadOnly AquariumGraphicProperty As DependencyProperty =
DependencyProperty.Register("AquariumGraphic", GetType(Uri), GetType(AquariumObject), New
FrameworkPropertyMetadata(Nothing, FrameworkPropertyMetadataOptions.AffectsRender, New
PropertyChangedCallback(AddressOf OnUriChanged)))
Public Property AquariumGraphic() As Uri
    Get
        Return CType(GetValue(AquariumGraphicProperty), Uri)
    End Get
    Set(ByVal value As Uri)
        SetValue(AquariumGraphicProperty, value)
    End Set
End Property
```

Anche in questo caso, per convenzione, il nome della proprietà wrapper deve corrispondere al nome scelto e specificato come primo parametro della chiamata [Register](#) che ha registrato la proprietà. Se la proprietà non segue la convenzione, non necessariamente vengono disabilitati tutti i possibili usi, ma si riscontreranno vari problemi rilevanti:

- Determinati aspetti di stili e modelli non funzioneranno.

- La maggior parte degli strumenti e delle finestre di progettazione devono basarsi sulle convenzioni di denominazione per serializzare correttamente XAML o per fornire assistenza relativamente all'ambiente della finestra di progettazione a livello di singola proprietà.
- L'implementazione corrente del caricatore XAML WPF ignora completamente i wrapper e si basa sulla convenzione di denominazione al momento dell'elaborazione dei valori dell'attributo. Per altre informazioni, vedere [Caricamento XAML e proprietà di dipendenza](#).

Metadati della proprietà per una nuova proprietà di dipendenza

Quando si registra una proprietà di dipendenza, la registrazione tramite il sistema di proprietà crea un oggetto dei metadati che archivia le caratteristiche della proprietà. Molte di queste caratteristiche hanno impostazioni predefinite che vengono impostate se la proprietà è registrata con le semplici firme di [Register](#). Altre firme di [Register](#) consentono di specificare i metadati desiderati durante la registrazione della proprietà. I metadati più comuni per le proprietà di dipendenza consistono nel fornire a tali proprietà un valore predefinito che viene applicato alle nuove istanze che usano la proprietà.

Se si crea una proprietà di dipendenza esistente in una classe derivata di [FrameworkElement](#), è possibile usare la classe di metadati più specializzata [FrameworkPropertyMetadata](#) invece della classe [PropertyMetadata](#) di base. Il costruttore per la classe [FrameworkPropertyMetadata](#) dispone di diverse firme in cui è possibile specificare varie caratteristiche dei metadati in combinazione. Se si desidera specificare solo il valore predefinito, utilizzare la firma che accetta un solo parametro di tipo [Object](#). Passare il parametro dell'oggetto come valore predefinito specifico del tipo per la proprietà (il valore predefinito fornito deve corrispondere al tipo specificato come parametro `propertyType` nella chiamata [Register](#)).

Per [FrameworkPropertyMetadata](#), è anche possibile specificare i flag di opzione dei metadati per la proprietà. Questi flag vengono convertiti in proprietà discrete nei metadati della proprietà dopo la registrazione e usati per comunicare determinate istruzioni condizionali agli altri processi quali il motore di layout.

Impostazione dei flag di metadati appropriati

- Se la proprietà o le modifiche del relativo valore influiscono sul interfaccia utente in particolare sul modo in cui il sistema di layout deve ridimensionare o eseguire il rendering dell'elemento in una pagina, impostare uno o più dei flag seguenti: [AffectsMeasure](#), [AffectsArrange](#)[AffectsRender](#).
 - [AffectsMeasure](#) indica che una modifica a questa proprietà richiede una modifica al rendering Interfaccia utente in cui l'oggetto contenitore potrebbe richiedere più o meno spazio all'interno dell'elemento padre. Ad esempio, è necessario impostare questo flag per una proprietà "Width".
 - [AffectsArrange](#) indica che una modifica a questa proprietà richiede una modifica al rendering Interfaccia utente che in genere non richiede una modifica nello spazio dedicato, ma indica che il posizionamento all'interno dello spazio è stato modificato. Ad esempio, è necessario impostare questo flag per una proprietà "Alignment".
 - [AffectsRender](#) indica che si è verificata un'altra modifica che non influirà sul layout e sulla misura, ma richiede un altro rendering. Un esempio potrebbe essere una proprietà che modifica un colore di un elemento esistente, ad esempio "Background".
 - Questi flag vengono spesso usati come protocollo nei metadati per le implementazioni di override del sistema di proprietà o per i callback del layout. Ad esempio, è possibile che si disponga di un callback [OnPropertyChanged](#) che chiama [InvalidateArrange](#) se una proprietà dell'istanza segnala una modifica del valore e ha [AffectsArrange](#) come `true` nei relativi metadati.
- Alcune proprietà possono influire sulle caratteristiche di rendering dell'elemento padre contenitore, in modo maggiore rispetto alle modifiche nelle dimensioni necessarie indicate in precedenza. Un

esempio è la proprietà [MinOrphanLines](#) utilizzata nel modello di documento dinamico, in cui le modifiche apportate a tale proprietà possono modificare il rendering globale del documento dinamico che contiene il paragrafo. Usare [AffectsParentArrange](#) o [AffectsParentMeasure](#) per identificare casi simili nelle proprietà.

- Per impostazione predefinita, le proprietà di dipendenza supportano il data binding. È possibile disabilitare intenzionalmente il data binding per i casi in cui non esiste uno scenario realistico per il data binding o nei quali le prestazioni del data binding per un oggetto di grandi dimensioni viene considerata un problema.
- Per impostazione predefinita, il valore predefinito di data binding [Mode](#) per le proprietà di dipendenza è [OneWay](#). È sempre possibile modificare l'associazione in modo che sia [TwoWay](#) per ogni istanza di associazione. Per informazioni dettagliate, vedere [specificare la direzione del binding](#). Tuttavia, come autore della proprietà di dipendenza, è possibile scegliere di fare in modo che la proprietà usi [TwoWay](#) modalità di associazione per impostazione predefinita. Un esempio di una proprietà di dipendenza esistente è [MenuItem.Is_submenuOpen](#); lo scenario per questa proprietà è che la logica di impostazione della [Is_submenuOpen](#) e la composizione di [MenuItem](#) interagiscono con lo stile del tema predefinito. La logica della proprietà [Is_submenuOpen](#) utilizza data binding in modo nativo per mantenere lo stato della proprietà in base alle altre proprietà di stato e alle chiamate ai metodi. Un'altra proprietà di esempio che associa [TwoWay](#) per impostazione predefinita è [TextBox.Text](#).
- È anche possibile abilitare l'ereditarietà delle proprietà in una proprietà di dipendenza personalizzata impostando il flag [Inherits](#). L'ereditarietà della proprietà è utile per uno scenario in cui gli elementi padre e gli elementi figlio hanno una proprietà in comune e avrebbe senso per gli elementi figlio avere quel particolare valore di proprietà impostato sullo stesso valore impostato dall'elemento padre. Una proprietà ereditabile di esempio è [DataContext](#), che viene usata per le operazioni di associazione per abilitare lo scenario Master-Detail importante per la presentazione dei dati. Rendendo [DataContext](#) ereditabile, anche tutti gli elementi figlio ereditano il contesto dati. A causa dell'ereditarietà del valore della proprietà, è possibile specificare un contesto dati alla radice della pagina o dell'applicazione e non è necessario specificarlo di nuovo per i binding in tutti i possibili elementi figlio. [DataContext](#) è anche un buon esempio per illustrare che l'ereditarietà esegue l'override del valore predefinito, ma può sempre essere impostata localmente su un particolare elemento figlio. Per informazioni dettagliate, vedere [usare il modello Master-Details con dati gerarchici](#). L'ereditarietà del valore della proprietà può incidere negativamente sulle prestazioni e pertanto deve essere usata sporadicamente. Per informazioni dettagliate, vedere [Ereditarietà del valore della proprietà](#).
- Impostare il flag [Journal](#) per indicare se la proprietà di dipendenza deve essere rilevata o utilizzata dai servizi di journaling di navigazione. Un esempio è la proprietà [SelectedIndex](#); qualsiasi elemento selezionato in un controllo di selezione deve essere reso permanente quando viene spostata la cronologia del journaling.

Proprietà di dipendenza di sola lettura

È possibile definire una proprietà di dipendenza di sola lettura. Tuttavia, gli scenari in base ai quali è possibile definire la proprietà come proprietà di sola lettura sono piuttosto diversi, allo stesso modo della procedura per registrare queste proprietà con il sistema di proprietà e di quella per esporre l'identificatore. Per altre informazioni, vedere [Proprietà di dipendenza di sola lettura](#).

Proprietà di dipendenza di tipo raccolta

Le proprietà di dipendenza di tipo di raccolta presentano alcuni problemi di implementazione aggiuntivi che è opportuno considerare. Per altri dettagli, vedere [Proprietà di dipendenza di tipo raccolta](#).

Considerazioni sulla sicurezza delle proprietà di dipendenza

Le proprietà di dipendenza devono essere dichiarate come proprietà pubbliche. I campi dell'identificatore delle proprietà di dipendenza devono essere dichiarati come campi statici pubblici. Anche se si tenta di dichiarare altri livelli di accesso (ad esempio protected), è sempre possibile accedere a una proprietà di dipendenza tramite l'identificatore in combinazione con le API del sistema di proprietà. Anche un campo dell'identificatore protetto è potenzialmente accessibile a causa della creazione di report dei metadati o delle API di determinazione del valore che fanno parte del sistema di proprietà, ad esempio [LocalValueEnumerator](#). Per altre informazioni, vedere [Sicurezza delle proprietà di dipendenza](#).

Proprietà di dipendenza e costruttori di classi

Esiste un principio generale nella programmazione del codice gestito (spesso applicato mediante strumenti di analisi del codice quale FxCop) in base al quale i costruttori di classi non devono chiamare metodi virtuali. Il motivo sta nel fatto che i costruttori possono essere chiamati come inizializzazione di base di un costruttore di classe derivata, pertanto l'uso del metodo virtuale tramite il costruttore potrebbe avvenire a uno stato incompleto dell'inizializzazione dell'istanza di oggetto in corso di creazione. Quando si deriva da una classe che deriva già da [DependencyObject](#), è necessario tenere presente che il sistema di proprietà stesso chiama ed espone internamente i metodi virtuali. Tali metodi virtuali sono parte dei servizi del sistema di proprietà WPF. L'esecuzione dell'override dei metodi consente alle classi derivate di partecipare alla determinazione del valore. Per evitare eventuali problemi con l'inizializzazione del runtime, è consigliabile evitare di impostare valori della proprietà di dipendenza all'interno dei costruttori di classi, a meno che non si segua un modello del costruttore molto specifico. Per altri dettagli, vedere [Modelli di costruttore sicuri per DependencyObject](#).

Vedere anche

- [Panoramica sulle proprietà di dipendenza](#)
- [Metadati delle proprietà di dipendenza](#)
- [Cenni preliminari sulla modifica di controlli](#)
- [Proprietà di dipendenza di tipo raccolta](#)
- [Sicurezza delle proprietà di dipendenza](#)
- [Caricamento XAML e proprietà di dipendenza](#)
- [Modelli di costruttore sicuri per DependencyObject](#)

Metadati delle proprietà di dipendenza

23/10/2019 • 21 minutes to read • [Edit Online](#)

Il Windows Presentation Foundation (WPF) sistema di proprietà include un sistema di creazione di report dei metadati che va oltre quello che può essere segnalato su una proprietà tramite reflection o caratteristiche generali di Common Language Runtime (CLR). I metadati di una proprietà di dipendenza possono anche essere assegnati in modo univoco dalla classe che definisce una proprietà di dipendenza, possono essere modificati quando la proprietà di dipendenza viene aggiunta a una classe diversa ed è possibile eseguirne specificatamente l'override tramite tutte le classi derivate che ereditano la proprietà di dipendenza dalla classe di base in fase di definizione.

Prerequisiti

Questo argomento presuppone la conoscenza delle proprietà di dipendenza dal punto di vista di un consumer delle proprietà di dipendenza esistenti nelle classi di Windows Presentation Foundation (WPF), nonché la lettura dell'argomento [Panoramica sulle proprietà di dipendenza](#). Per seguire gli esempi illustrati in questo argomento, è anche necessario conoscere XAML e saper scrivere applicazioni WPF.

Come usare i metadati delle proprietà di dipendenza

I metadati della proprietà di dipendenza sono un oggetto sul quale è possibile eseguire query per esaminare le caratteristiche di una proprietà di dipendenza. Anche il sistema di proprietà accede frequentemente ai metadati durante l'elaborazione di qualsiasi proprietà di dipendenza. L'oggetto metadati di una proprietà di dipendenza può contenere i seguenti tipi di informazioni:

- Valore predefinito della proprietà di dipendenza, se non è possibile determinare altri valori per la proprietà di dipendenza in base a valore locale, stile, ereditarietà e così via. Per una discussione dettagliata del ruolo dei valori predefiniti nella precedenza usata dal sistema di proprietà durante l'assegnazione dei valori per le proprietà di dipendenza, vedere [Precedenza del valore della proprietà di dipendenza](#).
- I riferimenti a implementazioni di callback che influiscono sui comportamenti di coercizione o di notifica di modifica in base al tipo di proprietario. Notare che questi callback vengono spesso definiti con un livello di accesso non pubblico, pertanto in genere non è possibile ottenere i riferimenti effettivi dai metadati, a meno che i riferimenti non rientrino nell'ambito di accesso consentito. Per altre informazioni sui callback per le proprietà di dipendenza, vedere [Callback e convalida delle proprietà di dipendenza](#).
- Se la proprietà di dipendenza in questione viene considerata come una proprietà a livello di framework WPF, i metadati potrebbero contenere caratteristiche della proprietà di dipendenza a livello di framework WPF, che segnalano informazioni e stato per servizi come la logica di ereditarietà delle proprietà e il motore di layout a livello di framework WPF. Per altre informazioni su questo aspetto dei metadati delle proprietà di dipendenza, vedere [Metadati delle proprietà del framework](#).

API dei metadati

Il tipo che indica la maggior parte delle informazioni sui metadati utilizzate dal sistema di proprietà [PropertyMetadata](#) è la classe. Le istanze dei metadati possono essere specificate facoltativamente quando le proprietà di dipendenza vengono registrate nel sistema di proprietà e una seconda volta per i tipi che si aggiungono come proprietari o che eseguono l'override dei metadati ereditati dalla definizione della proprietà di dipendenza della classe di base. Per i casi in cui una registrazione di proprietà non specifica metadati, viene

[PropertyMetadata](#) creando un valore predefinito con i valori predefiniti per tale classe. I metadati registrati vengono restituiti come [PropertyMetadata](#) quando si chiamano i vari [GetMetadata](#) overload che ottengono metadati da una proprietà di dipendenza in un' [DependencyObject](#) istanza di.

La [PropertyMetadata](#) classe viene quindi derivata da per fornire metadati più specifici per le divisioni di architettura, ad esempio le classi a livello di Framework WPF. [UIPropertyMetadata](#) aggiunge la creazione di report [FrameworkPropertyMetadata](#) di animazione e fornisce le proprietà a livello di Framework WPF indicate nella sezione precedente. Quando le proprietà di dipendenza vengono registrate, possono essere registrate con [PropertyMetadata](#) queste classi derivate. Quando i metadati vengono esaminati, è [PropertyMetadata](#) possibile eseguire il cast del tipo di base alle classi derivate in modo da poter esaminare le proprietà più specifiche.

NOTE

Le caratteristiche della proprietà che è possibile specificare [FrameworkPropertyMetadata](#) in sono talvolta indicate in questa documentazione come "flag". Quando si creano nuove istanze di metadati da usare nelle registrazioni delle proprietà di dipendenza o negli override dei metadati, è necessario specificare questi

[FrameworkPropertyMetadataOptions](#) valori usando l'enumerazione flag, quindi fornire i valori eventualmente concatenati dell'enumerazione al [FrameworkPropertyMetadata](#) costruttore. Tuttavia, una volta costruite, queste caratteristiche delle opzioni vengono [FrameworkPropertyMetadata](#) esposte all'interno di un come una serie di proprietà booleane anziché il valore di enumerazione di costruzione. Le proprietà booleane consentono di verificare ogni istruzione condizionale anziché richiedere l'applicazione di una maschera a un valore di enumerazione basato su flag per ottenere le informazioni necessarie. Il costruttore utilizza l'oggetto concatenato [FrameworkPropertyMetadataOptions](#) per garantire una ragionevole lunghezza della firma del costruttore, mentre i metadati costruiti effettivi espongono le proprietà discrete per rendere più intuitiva l'esecuzione di query sui metadati.

Esecuzione dell'override dei metadati e derivazione di una classe

Il sistema di proprietà di WPF include funzionalità consolidate che consentono di modificare alcune caratteristiche delle proprietà di dipendenza senza richiederne una implementazione nuova e completa. Questo risultato si ottiene creando un'istanza diversa dei metadati della proprietà per la proprietà di dipendenza presente in un tipo particolare. Notare che le proprietà di dipendenza non sono per la maggior parte proprietà virtuali, pertanto, a rigor di termini, "una nuova implementazione" di queste proprietà nelle classi ereditate potrebbe essere realizzata solo nascondendo il membro esistente.

Se lo scenario che si tenta di abilitare per una proprietà di dipendenza in un tipo non può essere realizzato modificando le caratteristiche delle proprietà di dipendenza esistenti, potrebbe essere necessario creare una classe derivata e successivamente dichiarare una proprietà di dipendenza personalizzata nella classe derivata. Una proprietà di dipendenza personalizzata si comporta in modo identico alle proprietà di dipendenza WPF definite dalle API. Per altri dettagli sulle proprietà di dipendenza personalizzate, vedere [Proprietà di dipendenza personalizzate](#).

Una caratteristica rilevante di una proprietà di dipendenza di cui non è possibile eseguire l'override è il tipo di valore. Se si eredita una proprietà di dipendenza che si comporta nel modo richiesto, ma è richiesto un tipo diverso per questa proprietà, sarà necessario implementare una proprietà di dipendenza personalizzata e forse collegare le proprietà tramite la conversione dei tipi oppure un'altra implementazione nella classe personalizzata. Non è inoltre possibile sostituire un oggetto [ValidateValueCallback](#) esistente perché questo callback esiste nel campo di registrazione e non nei relativi metadati.

Scenari per la modifica dei metadati esistenti

Se si usano i metadati di una proprietà di dipendenza esistente, uno scenario comune per modificare i metadati della proprietà di dipendenza consiste nella modifica del valore predefinito. La modifica o l'aggiunta di callback del sistema di proprietà rappresenta uno scenario più avanzato. Questa operazione può essere necessaria quando l'implementazione di una classe derivata ha molte interrelazioni tra le proprietà di dipendenza. Una

delle condizioni per poter disporre di un modello di programmazione che supporti l'utilizzo del codice e l'utilizzo dichiarativo consiste nella possibilità di impostare le proprietà in qualsiasi ordine. Le proprietà dipendenti devono quindi essere impostate JIT senza contesto e senza la possibilità di basarsi sulla conoscenza di un ordine di impostazione, come potrebbe accadere in un costruttore. Per altre informazioni su questo aspetto del sistema di proprietà, vedere [Callback e convalida delle proprietà di dipendenza](#). Notare che i callback di convalida non fanno parte dei metadati, ma dell'identificatore della proprietà di dipendenza. I callback di convalida non possono quindi essere modificati mediante l'override dei metadati.

In alcuni casi, è possibile che si decida di modificare le opzioni dei metadati delle proprietà a livello di framework WPF nelle proprietà di dipendenza esistenti. Queste opzioni consentono di comunicare alcune condizioni note relative alle proprietà a livello di framework WPF ad altri processi a livello di framework WPF, ad esempio il sistema di layout. L'impostazione delle opzioni viene in genere eseguita solo quando si registra una nuova proprietà di dipendenza, ma è anche possibile modificare i metadati della proprietà a livello di Framework WPF come [OverrideMetadata](#) parte [AddOwner](#) di una chiamata a o. Per conoscere i valori specifici da usare e per altre informazioni, vedere [Metadati delle proprietà del framework](#). Per altre informazioni relative alla modalità di impostazione di queste opzioni per una proprietà di dipendenza appena registrata, vedere [Proprietà di dipendenza personalizzate](#).

Override dei metadati

Lo scopo dell'override dei metadati è principalmente la possibilità di modificare i diversi comportamenti derivati dai metadati applicati alla proprietà di dipendenza esistente per il tipo. Le ragioni di questa operazione vengono spiegate più dettagliatamente nella sezione [Metadati](#). Per altre informazioni e alcuni esempi di codice, vedere [Eseguire l'override dei metadati per una proprietà di dipendenza](#).

I metadati della proprietà possono essere forniti per una proprietà di dipendenza durante la [Register](#) chiamata di registrazione (). Tuttavia, in molti casi è possibile fornire metadati specifici del tipo per la classe, quando questa eredita tale proprietà di dipendenza. Questa operazione può essere eseguita chiamando il [OverrideMetadata](#) metodo. Per un esempio WPF delle API, la [FrameworkElement](#) classe è il tipo che prima registra la [Focusable](#) proprietà di dipendenza. Tuttavia, [Control](#) la classe esegue l'override dei metadati per la proprietà di dipendenza per fornire il proprio valore predefinito `false` iniziale `true`, cambiandolo da a e altrimenti riutilizza l'implementazione originale [Focusable](#).

Quando si esegue l'override dei metadati, le diverse caratteristiche dei metadati vengono unite oppure sostituite.

- [PropertyChangedCallback](#) viene unito. Se si aggiunge un nuovo [PropertyChangedCallback](#) oggetto, tale callback viene archiviato nei metadati. Se non si specifica un oggetto [PropertyChangedCallback](#) nell'override, il valore di [PropertyChangedCallback](#) viene promosso come riferimento dal predecessore più vicino che lo ha specificato nei metadati.
- Il comportamento effettivo del sistema di [PropertyChangedCallback](#) proprietà per è che le implementazioni per tutti i proprietari di metadati nella gerarchia vengono mantenute e aggiunte a una tabella, con ordine di esecuzione da parte del sistema di proprietà perché i callback della classe più derivata vengono richiamati per primi.
- [DefaultValue](#) viene sostituito. Se non si specifica un oggetto [DefaultValue](#) nell'override, il valore di [DefaultValue](#) deriva dal predecessore più vicino che lo ha specificato nei metadati.
- [CoerceValueCallback](#) implementazioni vengono sostituite. Se si aggiunge un nuovo [CoerceValueCallback](#) oggetto, tale callback viene archiviato nei metadati. Se non si specifica un oggetto [CoerceValueCallback](#) nell'override, il valore di [CoerceValueCallback](#) viene promosso come riferimento dal predecessore più vicino che lo ha specificato nei metadati.
- Il comportamento del sistema di proprietà è che [CoerceValueCallback](#) solo l'oggetto nei metadati immediati viene richiamato. Non viene mantenuto alcun [CoerceValueCallback](#) riferimento ad altre implementazioni nella gerarchia.

Questo comportamento viene implementato da [Merge](#) ed è possibile eseguirne l'override nelle classi di metadati derivate.

Override dei metadati delle proprietà associate

In WPF, le proprietà associate vengono implementate come proprietà di dipendenza. In altri termini, dispongono anche di metadati della proprietà di cui le singole classi possono eseguire l'override. Le considerazioni relative all'ambito per una proprietà WPF associata sono in [DependencyObject](#) genere di cui è possibile impostare una proprietà associata. Pertanto, qualsiasi [DependencyObject](#) classe derivata può eseguire l'override dei metadati per qualsiasi proprietà associata, in quanto potrebbe essere impostata su un'istanza della classe. È possibile eseguire l'override dei valori predefiniti, dei callback o delle proprietà di segnalazione delle caratteristiche a livello di framework WPF. Se la proprietà associata viene impostata su un'istanza della classe, vengono applicate le caratteristiche dei metadati della proprietà di override. È possibile, ad esempio, eseguire l'override del valore predefinito, in modo che il valore di override venga segnalato come valore della proprietà associata nelle istanze della classe, tutte le volte che la proprietà non viene impostata diversamente.

NOTE

La [Inherits](#) proprietà non è pertinente per le proprietà associate.

Aggiunta di un classe come proprietario di una proprietà di dipendenza esistente

Una classe può aggiungersi come proprietario di una proprietà di dipendenza che è già stata registrata, usando il [AddOwner](#) metodo. In questo modo, la classe può usare una proprietà di dipendenza registrata originariamente per un tipo diverso. In genere, la classe aggiunta non è una classe derivata del tipo che ha registrato per primo quella proprietà di dipendenza come proprietario. In realtà, in questo modo la classe e le relative classi derivate possono "ereditare" un'implementazione della proprietà di dipendenza senza che la classe proprietario originale e quella aggiunta si trovino nella stessa gerarchia di classi. Inoltre, la classe aggiunta (e tutte le classi derivate) possono quindi fornire metadati specifici sul tipo per la proprietà di dipendenza originale.

Oltre ad aggiungersi come proprietario tramite i metodi di utilità del sistema di proprietà, la classe aggiunta deve dichiarare ulteriori membri pubblici al proprio interno al fine di far partecipare completamente la proprietà di dipendenza nel sistema di proprietà con l'esposizione al codice e al markup. Una classe che aggiunge una proprietà di dipendenza esistente ha le stesse responsabilità, per quanto riguarda l'esposizione del modello a oggetti per quella proprietà di dipendenza, di una classe che definisce una nuova proprietà di dipendenza personalizzata. Il primo di questi membri da esporre è un campo dell'identificatore della proprietà di dipendenza. Questo campo deve essere un `public static readonly` campo di tipo [DependencyProperty](#), che viene assegnato al valore restituito della [AddOwner](#) chiamata. Il secondo membro da definire è la proprietà "wrapper" Common Language Runtime (CLR). Il wrapper rende molto più comodo modificare la proprietà di dipendenza nel codice (si evitano chiamate a [SetValue](#) ogni volta e si può effettuare la chiamata solo una volta nel wrapper). Il wrapper viene implementato con la stessa procedura valida per i wrapper implementati nella registrazione di una proprietà di dipendenza personalizzata. Per altre informazioni sull'implementazione di una proprietà di dipendenza, vedere [Proprietà di dipendenza personalizzate](#) e [Aggiungere un tipo di proprietario per una proprietà di dipendenza](#).

AddOwner e proprietà associate

È possibile chiamare [AddOwner](#) per una proprietà di dipendenza definita come proprietà associata dalla classe Owner. In genere, questa operazione viene eseguita per esporre la proprietà precedentemente associata come proprietà di dipendenza non associata. Si esporrà quindi il [AddOwner](#) valore restituito `public static readonly` come campo da utilizzare come identificatore della proprietà di dipendenza e verranno definite le proprietà "wrapper" appropriate, in modo che la proprietà venga visualizzata nella tabella dei membri e supporti una proprietà non associata utilizzo nella classe.

Vedere anche

- [PropertyMetadata](#)
- [DependencyObject](#)
- [DependencyProperty](#)
- [GetMetadata](#)
- [Panoramica sulle proprietà di dipendenza](#)
- [Metadati delle proprietà del framework](#)

Callback e convalida delle proprietà di dipendenza

23/10/2019 • 13 minutes to read • [Edit Online](#)

Questo argomento descrive come creare proprietà di dipendenza usando le implementazioni personalizzate alternative per funzionalità correlate alle proprietà, ad esempio la determinazione della convalida, i callback richiamati ogni volta che il valore effettivo della proprietà viene modificato e l'override delle possibili influenze esterne sulla determinazione del valore. In questo argomento vengono anche presentati scenari in cui è possibile espandere i comportamenti predefiniti del sistema di proprietà usando queste tecniche.

Prerequisiti

Nell'argomento si presuppone la conoscenza degli scenari di base dell'implementazione di una proprietà di dipendenza e del modo in cui i metadati vengono applicati a una proprietà di dipendenza personalizzata. Per il contesto, vedere [Proprietà di dipendenza personalizzate](#) e [Metadati delle proprietà di dipendenza](#).

Callback di convalida

I callback di convalida possono essere assegnati a una proprietà di dipendenza al momento della sua prima registrazione. Il callback di convalida non fa parte dei metadati della proprietà. Si tratta di un input diretto del [Register](#) metodo. Di conseguenza, dopo averlo creato per una proprietà di dipendenza, un callback di convalida non può essere sottoposto a override da parte di una nuova implementazione.

```
public static readonly DependencyProperty CurrentReadingProperty = DependencyProperty.Register(
    "CurrentReading",
    typeof(double),
    typeof(Gauge),
    new FrameworkPropertyMetadata(
        Double.NaN,
        FrameworkPropertyMetadataOptions.AffectsMeasure,
        new PropertyChangedCallback(OnCurrentReadingChanged),
        new CoerceValueCallback(CoerceCurrentReading)
    ),
    new ValidateValueCallback(IsValidReading)
);
public double CurrentReading
{
    get { return (double)GetValue(CurrentReadingProperty); }
    set { SetValue(CurrentReadingProperty, value); }
}
```

```

Public Shared ReadOnly CurrentReadingProperty As DependencyProperty =
    DependencyProperty.Register("CurrentReading",
        GetType(Double), GetType(Gauge),
        New FrameworkPropertyMetadata(Double.NaN,
            FrameworkPropertyMetadataOptions.AffectsMeasure,
            New PropertyChangedCallback(AddressOf OnCurrentReadingChanged),
            New CoerceValueCallback(AddressOf CoerceCurrentReading)),
        New ValidateValueCallback(AddressOf IsValidReading))

```

```

Public Property CurrentReading() As Double
    Get
        Return CDblGetValue(CurrentReadingProperty))
    End Get
    Set(ByVal value As Double)
        SetValue(CurrentReadingProperty, value)
    End Set
End Property

```

I callback vengono implementati in modo che venga loro fornito un valore di oggetto. Restituiscono `true` se il valore fornito è valido per la proprietà e in caso contrario restituiscono `false`. Si presuppone che la proprietà sia del tipo corretto per il tipo registrato nel sistema di proprietà, quindi all'interno dei callback non viene generalmente eseguito il controllo del tipo. I callback vengono usati dal sistema di proprietà in varie operazioni diverse. Questa operazione include l'inizializzazione del tipo iniziale per valore predefinito, la modifica `SetValue` livello di codice richiamando o il tentativo di eseguire l'override dei metadati con il nuovo valore predefinito fornito. Se il callback di convalida viene richiamato da una di queste operazioni e restituisce `false`, verrà generata un'eccezione. Gli autori di applicazioni devono essere preparati a gestire queste eccezioni. Un uso comune dei callback di convalida è la convalida dei valori di enumerazione o il vincolo di valori di Integer o Double quando la proprietà imposta misure che devono essere uguali o maggiori di zero.

I callback di convalida sono specificamente concepiti come validatori di classi, non di istanze. I parametri del callback non comunicano un oggetto `DependencyObject` specifico sul quale sono impostate le proprietà da convalidare. Di conseguenza, i callback di convalida non sono utili per applicare le possibili "dipendenze" che possono influenzare un valore di proprietà, in cui il valore specifico delle istanze di una proprietà è dipendente da fattori quali i valori specifici delle istanze di altre proprietà o lo stato di runtime.

Di seguito è riportato un esempio di codice per uno scenario di callback di convalida molto semplice: la convalida che una `Double` proprietà tipizzata `PositiveInfinity` come `NegativeInfinity` primitiva non sia o.

```

public static bool IsValidReading(object value)
{
    Double v = (Double)value;
    return (!v.Equals(Double.NegativeInfinity) && !v.Equals(Double.PositiveInfinity));
}

```

```

Public Shared Function IsValidReading(ByVal value As Object) As Boolean
    Dim v As Double = CType(value, Double)
    Return ((Not v.Equals(Double.NegativeInfinity)) AndAlso
            (Not v.Equals(Double.PositiveInfinity)))
End Function

```

Callback di valori soggetti a coercizione ed eventi di proprietà modificate

I callback di valori con coercione passano l' `DependencyObject` istanza specifica per le proprietà, `PropertyChangedCallback` come le implementazioni richiamate dal sistema di proprietà ogni volta che il valore di una proprietà di dipendenza viene modificato. Usando questi due callback in combinazione, è possibile creare

una serie di proprietà su elementi in cui le modifiche in una proprietà imporranno una coercizione o una rivalutazione di un'altra proprietà.

Un scenario tipico per l'uso di un collegamento di proprietà di dipendenza è il caso in cui si dispone di una proprietà basata su un'interfaccia utente in cui l'elemento contiene una proprietà per il valore minimo e una per il valore massimo, oltre a una terza proprietà per il valore effettivo o corrente. In questo caso, se il massimo è stato regolato in modo che il valore corrente superi il nuovo massimo, sarà opportuno assegnare il valore corrente in modo che non sia maggiore del nuovo massimo e impostare una relazione simile per il valore minimo rispetto al valore corrente.

Di seguito viene riportato un brevissimo esempio di codice per una delle tre proprietà di dipendenza che illustrano questa relazione. Nell'esempio viene illustrata la modalità di registrazione della proprietà `CurrentReading` di un set Min/Max/Current di proprietà *Reading. Viene usata la convalida come illustrato nella sezione precedente.

```
public static readonly DependencyProperty CurrentReadingProperty = DependencyProperty.Register(
    "CurrentReading",
    typeof(double),
    typeof(Gauge),
    new FrameworkPropertyMetadata(
        Double.NaN,
        FrameworkPropertyMetadataOptions.AffectsMeasure,
        new PropertyChangedCallback(OnCurrentReadingChanged),
        new CoerceValueCallback(CoerceCurrentReading)
    ),
    new ValidateValueCallback(IsValidReading)
);
public double CurrentReading
{
    get { return (double)GetValue(CurrentReadingProperty); }
    set { SetValue(CurrentReadingProperty, value); }
}
```

```
Public Shared ReadOnly CurrentReadingProperty As DependencyProperty =
    DependencyProperty.Register("CurrentReading",
        GetType(Double), GetType(Gauge),
        New FrameworkPropertyMetadata(Double.NaN,
            FrameworkPropertyMetadataOptions.AffectsMeasure,
            New PropertyChangedCallback(AddressOf OnCurrentReadingChanged),
            New CoerceValueCallback(AddressOf CoerceCurrentReading)),
        New ValidateValueCallback(AddressOf IsValidReading))

Public Property CurrentReading() As Double
    Get
        Return CDbl(GetValue(CurrentReadingProperty))
    End Get
    Set(ByVal value As Double)
        SetValue(CurrentReadingProperty, value)
    End Set
End Property
```

Il callback per proprietà modificate per il valore Current viene usato per inoltrare la modifica ad altre proprietà dipendenti, richiamando in modo esplicito i callback di valori soggetti a coercione registrati per tali proprietà:

```
private static void OnCurrentReadingChanged(DependencyObject d, DependencyPropertyChangedEventArgs e)
{
    d.CoerceValue(MinReadingProperty);
    d.CoerceValue(MaxReadingProperty);
}
```

```

Private Shared Sub OnCurrentReadingChanged(ByVal d As DependencyObject, ByVal e As
DependencyPropertyChangedEventArgs)
    d.CoerceValue(MinReadingProperty)
    d.CoerceValue(MaxReadingProperty)
End Sub

```

Il callback di valori soggetti a coercizione verifica i valori delle proprietà da cui la proprietà corrente è potenzialmente dipendente e assegna il valore corrente, se necessario:

```

private static object CoerceCurrentReading(DependencyObject d, object value)
{
    Gauge g = (Gauge)d;
    double current = (double)value;
    if (current < g.MinReading) current = g.MinReading;
    if (current > g.MaxReading) current = g.MaxReading;
    return current;
}

```

```

Private Shared Function CoerceCurrentReading(ByVal d As DependencyObject, ByVal value As Object) As Object
    Dim g As Gauge = CType(d, Gauge)
    Dim current As Double = CDbl(value)
    If current < g.MinReading Then
        current = g.MinReading
    End If
    If current > g.MaxReading Then
        current = g.MaxReading
    End If
    Return current
End Function

```

NOTE

I valori predefiniti delle proprietà non vengono assegnati. Un valore di proprietà uguale al valore predefinito potrebbe verificarsi se il valore di una proprietà ha ancora l'impostazione predefinita iniziale o la cancellazione di [ClearValue](#) altri valori con.

I callback di valori soggetti a coercione e per proprietà modificate fanno parte dei metadati delle proprietà. Pertanto, è possibile modificare i callback per una determinata proprietà di dipendenza quando quest'ultima è presente su un tipo derivato dal tipo proprietario della proprietà di dipendenza, eseguendo l'override dei metadati per tale proprietà sul tipo.

Coercizione avanzata e scenari di callback

Vincoli e valori desiderati

I [CoerceValueCallback](#) callback verranno usati dal sistema di proprietà per assegnare un valore in base alla logica dichiarata, ma un valore forzato di una proprietà impostata localmente manterrà il "valore desiderato" internamente. Se i vincoli sono basati su altri valori di proprietà che possono essere modificati in modo dinamico nel corso della durata dell'applicazione, anche i vincoli di coercione vengono modificati in modo dinamico e la proprietà vincolata può modificare il proprio valore per avvicinarsi il più possibile al valore desiderato in base ai nuovi vincoli. Il valore diventerà il valore desiderato se tutti i vincoli verranno rimossi. È potenzialmente possibile introdurre alcuni scenari di dipendenze piuttosto complicati se si hanno più proprietà dipendenti l'una dall'altra in modo circolare. Ad esempio, nello scenario Min/Max/Current, è possibile scegliere che i valori Minimum e Maximum siano impostabili dall'utente. In questo caso, potrebbe essere necessario imporre che il valore Maximum sia sempre superiore al valore Minimum e viceversa. Ma se tale coercione è

attiva e il valore Maximum viene assegnato a Minimum, il valore Current viene lasciato in uno stato non impostabile, poiché è dipendente da entrambi ed è vincolato all'intervallo compreso tra i valori, che è pari a zero. Se i valori Maximum o Minimum vengono modificati, il valore Current sembrerà seguire uno dei valori, perché il valore desiderato di Current è ancora archiviato e tenta di raggiungere il valore desiderato con l'allentarsi dei vincoli.

Le dipendenze complesse non sono errate dal punto di vista tecnico, ma possono causare una leggera riduzione delle prestazioni se richiedono numerose rivalutazioni, oltre a generare confusione negli utenti se influiscono direttamente sull'interfaccia utente. È necessario prestare attenzione con i callback di valori soggetti a coercizione e per proprietà modificate e accertarsi che la coercizione tentata possa essere considerata nel modo meno ambiguo possibile e non imponga vincoli eccessivi.

Uso di CoerceValue per annullare le modifiche dei valori

Il sistema di proprietà considererà [CoerceValueCallback](#) tutti i che restituiscono il valore [UnsetValue](#) come caso speciale. Questo caso speciale indica che la modifica della proprietà che ha generato la [CoerceValueCallback](#) chiamata deve essere rifiutata dal sistema di proprietà e che il sistema di proprietà deve segnalare qualsiasi valore precedente della proprietà. Questo meccanismo può essere utile per controllare che le modifiche a una proprietà avviate in modo asincrono siano ancora valide per lo stato dell'oggetto corrente e per eliminarle in caso non lo siano. Un altro possibile scenario è costituito dalla possibilità di eliminare in modo selettivo un valore a seconda di quale componente della determinazione dei valori di proprietà sia responsabile del valore segnalato. A tale scopo, è possibile usare l' [DependencyProperty](#) oggetto passato nel callback e l'identificatore di proprietà come input [GetValueSource](#) per [ValueSource](#), quindi elaborare.

Vedere anche

- [Panoramica sulle proprietà di dipendenza](#)
- [Metadati delle proprietà di dipendenza](#)
- [Proprietà di dipendenza personalizzate](#)

Metadati delle proprietà del framework

04/11/2019 • 12 minutes to read • [Edit Online](#)

Per le proprietà degli elementi oggetto considerati situati a livello di framework WPF nell'architettura di Windows Presentation Foundation (WPF) sono segnalate le opzioni di metadati delle proprietà del framework. In generale, la designazione a livello di Framework WPF comporta che funzionalità quali il rendering, la data binding e i perfezionamenti del sistema di proprietà vengono gestiti dalle API di presentazione WPF e dagli eseguibili. Questi sistemi eseguono una query sui metadati delle proprietà del framework per determinare le caratteristiche specifiche delle funzionalità di particolari proprietà dell'elemento.

Prerequisites

Questo argomento presuppone la conoscenza delle proprietà di dipendenza dal punto di vista di un consumer delle proprietà di dipendenza esistenti nelle classi di Windows Presentation Foundation (WPF), nonché la lettura dell'argomento [Panoramica sulle proprietà di dipendenza](#). È inoltre necessario aver letto l'argomento [Metadati delle proprietà di dipendenza](#).

Contenuto dei metadati delle proprietà del framework

I metadati delle proprietà del framework possono essere suddivisi nelle seguenti categorie:

- Segnalazione delle proprietà di layout che influiscono su un elemento ([AffectsArrange](#), [AffectsMeasure](#), [AffectsRender](#)). È possibile impostare questi flag nei metadati se la proprietà ha effetto sui rispettivi aspetti e si implementa anche il [MeasureOverride](#) / metodi di [ArrangeOverride](#) nella classe per fornire un comportamento di rendering specifico e informazioni al sistema di layout. In genere, tale implementazione verificherebbe le convalide delle proprietà nelle proprietà di dipendenza, qualora una di tali proprietà di layout risultasse impostata su true nei metadati della proprietà; solo tali convalide richiederebbero un nuovo passaggio di layout.
- Segnalazione delle proprietà di layout che influiscono sull'elemento padre di un elemento ([AffectsParentArrange](#), [AffectsParentMeasure](#)). Alcuni esempi in cui questi flag sono impostati per impostazione predefinita sono [FixedPage.Left](#) e [Paragraph.KeepWithNext](#).
- [Inherits](#) Per impostazione predefinita, le proprietà di dipendenza non ereditano valori. [OverridesInheritanceBehavior](#) consente anche al percorso di ereditarietà di spostarsi in una struttura ad albero visuale, operazione necessaria per alcuni scenari di composizione del controllo.

NOTE

Il termine "ereditarietà" nel contesto dei valori della proprietà è specifico delle proprietà di dipendenza e indica che gli elementi figlio possono ereditare il valore della proprietà di dipendenza effettivo dagli elementi padre grazie a una funzionalità a livello di framework WPF del sistema di proprietà di WPF. Non è correlato direttamente al tipo di codice gestito né all'ereditarietà dei membri tramite tipi derivati. Per informazioni dettagliate, vedere [Ereditarietà del valore della proprietà](#).

- Funzionalità di Reporting data binding ([IsNotDataBindable](#), [BindsTwoWayByDefault](#)). Per impostazione predefinita, le proprietà di dipendenza nel framework supportano il data binding, con un comportamento di associazione unidirezionale. È possibile disabilitare data binding se non è presente alcuno scenario per esso (perché sono progettati per essere flessibili ed estendibili, non sono disponibili molti esempi di tali proprietà nelle API WPF predefinite). È possibile impostare Binding per avere un valore predefinito

bidirezionale per le proprietà che uniscono i comportamenti di un controllo tra le parti dei componenti ([IsSubmenuOpen](#) è un esempio) o in cui l'associazione bidirezionale è lo scenario comune e previsto per gli utenti ([Text](#) è un esempio). La modifica dei metadati correlati al data binding ha effetto sull'impostazione predefinita. È sempre possibile modificare l'impostazione predefinita per una singola associazione. Per maggiori dettagli sulle modalità di associazione e sull'associazione in generale, vedere [Cenni preliminari sull'associazione dati](#).

- Segnalazione se le proprietà devono essere inserite nel journal da applicazioni o servizi che supportano il journaling ([Journal](#)). Per impostazione predefinita, l'inserimento nel giornale di registrazione non è abilitato per gli elementi generali, ma è abilitato in maniera selettiva per determinati controlli di input dell'utente. Questa proprietà deve essere letta dai servizi di inserimento nel giornale di registrazione, inclusa l'implementazione dell'inserimento nel giornale di registrazione di WPF ed è impostata nei controlli utente, ad esempio le selezioni dell'utente all'interno di elenchi che devono essere resi persistenti nei vari passaggi di navigazione. Per informazioni sull'inserimento nel giornale di registrazione, vedere [Cenni preliminari sulla navigazione](#).

Lettura di FrameworkPropertyMetadata

Ognuna delle proprietà collegate sopra sono le proprietà specifiche che il [FrameworkPropertyMetadata](#) aggiunge alla relativa classe di base immediata [UIPropertyMetadata](#). Per impostazione predefinita, ognuna di queste proprietà sarà `false`. Una richiesta di metadati per una proprietà in cui è importante conoscere il valore di queste proprietà deve tentare di eseguire il cast dei metadati restituiti a [FrameworkPropertyMetadata](#), quindi controllare i valori delle singole proprietà in base alle esigenze.

Specifiche dei metadati

Quando si crea una nuova istanza di metadati ai fini dell'applicazione dei metadati a una nuova registrazione della proprietà di dipendenza, è possibile scegliere la classe di metadati da utilizzare, ovvero la [PropertyMetadata](#) di base o una classe derivata, ad esempio [FrameworkPropertyMetadata](#). In generale, è consigliabile usare [FrameworkPropertyMetadata](#), in particolare se la proprietà ha qualsiasi interazione con il sistema di proprietà e WPF funzioni quali layout e data binding. Un'altra opzione per scenari più sofisticati consiste nel derivare da [FrameworkPropertyMetadata](#) per creare una classe di report dei metadati personalizzata con informazioni aggiuntive trasportate nei relativi membri. In alternativa, è possibile usare [PropertyMetadata](#) o [UIPropertyMetadata](#) per comunicare il livello di supporto per le funzionalità dell'implementazione.

Per le proprietà esistenti (chiamata [AddOwner](#) o [OverrideMetadata](#)), è sempre necessario eseguire l'override con il tipo di metadati usato dalla registrazione originale.

Se si crea un'istanza di [FrameworkPropertyMetadata](#), esistono due modi per popolare i metadati con i valori per le proprietà specifiche che comunicano le caratteristiche delle proprietà del Framework:

1. Usare la firma del costruttore [FrameworkPropertyMetadata](#) che consente un parametro di `flags`. Questo parametro deve essere compilato con tutti i valori combinati desiderati dei flag di enumerazione [FrameworkPropertyMetadataOptions](#).
2. Usare una delle firme senza un parametro di `flags`, quindi impostare ogni proprietà booleana Reporting su [FrameworkPropertyMetadata](#) per `true` per ogni modifica di caratteristica desiderata. In questo caso, è necessario impostare queste proprietà prima della costruzione di qualsiasi elemento con questa proprietà di dipendenza. Le proprietà booleane sono configurate per la lettura e la scrittura, in modo da consentire di popolare i metadati evitando l'inserimento del parametro `flags`; tuttavia, i metadati devono essere contrassegnati come sealed prima che la proprietà venga usata. Il tentativo di impostare le proprietà dopo la richiesta dei metadati sarà pertanto considerato un'operazione non valida.

Comportamento di unione dei metadati delle proprietà del framework

Quando si esegue l'override dei metadati delle proprietà del framework, le diverse caratteristiche dei metadati vengono unite o sostituite.

- [PropertyChangedCallback](#) viene Unito. Se si aggiunge un nuovo [PropertyChangedCallback](#), tale callback viene archiviato nei metadati. Se non si specifica un [PropertyChangedCallback](#) nell'override, il valore di [PropertyChangedCallback](#) viene promosso come riferimento dal predecessore più vicino che lo ha specificato nei metadati.
- Il comportamento effettivo del sistema di proprietà per [PropertyChangedCallback](#) è che le implementazioni per tutti i proprietari di metadati nella gerarchia vengono mantenute e aggiunte a una tabella, con l'ordine di esecuzione da parte del sistema di proprietà che vengono richiamati i callback della classe derivata più profondamente. prima. I callback ereditati vengono eseguiti solo una volta, essendo considerati come appartenenti alla classe che li aveva posizionati nei metadati.
- [DefaultValue](#) viene sostituito. Se non si specifica un [PropertyChangedCallback](#) nell'override, il valore di [DefaultValue](#) deriva dal predecessore più vicino che lo ha specificato nei metadati.
- le implementazioni [CoerceValueCallback](#) vengono sostituite. Se si aggiunge un nuovo [CoerceValueCallback](#), tale callback viene archiviato nei metadati. Se non si specifica un [CoerceValueCallback](#) nell'override, il valore di [CoerceValueCallback](#) viene promosso come riferimento dal predecessore più vicino che lo ha specificato nei metadati.
- Il comportamento del sistema di proprietà è che vengono richiamati solo i [CoerceValueCallback](#) nei metadati immediati. Non viene mantenuto alcun riferimento ad altre implementazioni [CoerceValueCallback](#) nella gerarchia.
- I flag dell'enumerazione [FrameworkPropertyMetadataOptions](#) vengono combinati come operazione OR bit per bit. Se si specifica [FrameworkPropertyMetadataOptions](#), le opzioni originali non vengono sovrascritte. Per modificare un'opzione, impostare la proprietà corrispondente in [FrameworkPropertyMetadata](#). Se, ad esempio, l'oggetto [FrameworkPropertyMetadata](#) originale impone il flag di [FrameworkPropertyMetadataOptions.NotDataBindable](#), è possibile modificarlo impostando [FrameworkPropertyMetadata.IsNotDataBindable](#) su `false`.

Questo comportamento viene implementato da [Mergee](#) può essere sottoposto a override nelle classi di metadati derivate.

Vedere anche

- [GetMetadata](#)
- [Metadati delle proprietà di dipendenza](#)
- [Panoramica sulle proprietà di dipendenza](#)
- [Proprietà di dipendenza personalizzate](#)

Precedenza del valore della proprietà di dipendenza

10/02/2020 • 29 minutes to read • [Edit Online](#)

Questo argomento illustra il modo in cui i meccanismi del sistema di proprietà Windows Presentation Foundation (WPF) possono influire sul valore di una proprietà di dipendenza e descrive la precedenza in base alla quale gli aspetti del sistema di proprietà si applicano al valore effettivo di una proprietà.

Prerequisites

Questo argomento presuppone la conoscenza delle proprietà di dipendenza dal punto di vista di un consumer delle proprietà di dipendenza esistenti nelle classi di WPF, nonché la lettura dell'argomento [Panoramica sulle proprietà di dipendenza](#). Per seguire gli esempi illustrati in questo argomento, è anche necessario conoscere Extensible Application Markup Language (XAML) e saper scrivere applicazioni WPF.

Il sistema di proprietà WPF

Il sistema di proprietà WPF offre un potente strumento per consentire la determinazione del valore delle proprietà di dipendenza in base a numerosi fattori, abilitando funzionalità quali la convalida delle proprietà in tempo reale, l'associazione tardiva e la notifica alle proprietà correlate delle modifiche ai valori di altre proprietà. L'ordine e logica precisi usati per determinare i valori delle proprietà di dipendenza sono piuttosto complessi. La conoscenza di questo ordine consentirà di evitare l'impostazione di proprietà non necessarie e potrebbe anche eliminare i dubbi sui motivi precisi per i quali alcuni tentativi di influenzare o anticipare un valore delle proprietà di dipendenza non hanno determinato il valore atteso.

Le proprietà di dipendenza possono essere impostate in più punti

Di seguito è riportato un esempio XAML in cui la stessa proprietà ([Background](#)) presenta tre diverse operazioni "set" che potrebbero influenzare il valore.

```
<Button Background="Red">
    <Button.Style>
        <Style TargetType="{x:Type Button}">
            <Setter Property="Background" Value="Green"/>
            <Style.Triggers>
                <Trigger Property="IsMouseOver" Value="True">
                    <Setter Property="Background" Value="Blue" />
                </Trigger>
            </Style.Triggers>
        </Style>
    </Button.Style>
    Click
</Button>
```

In questo caso, quale colore ci si aspetta che verrà applicato, rosso, verde o blu?

Ad eccezione dei valori animati e della coercizione, i set di proprietà locali vengono impostati con la precedenza più elevata. Se si imposta localmente un valore, è possibile prevedere che il valore venga accettato, anche al di là degli stili o dei modelli di controllo. Nell'esempio [Background](#) viene impostato su rosso localmente. Pertanto, lo stile definito in questo ambito, anche se si tratta di uno stile implicito altrimenti applicabile a tutti gli elementi di quel tipo in tale ambito, non è la precedenza più elevata per fornire il valore della proprietà [Background](#). Se è

stato rimosso il valore locale Red dall'istanza specifica del controllo Button, lo stile avrà la precedenza e il pulsante otterrà il valore di Background dallo stile. All'interno dello stile, i trigger hanno la precedenza, pertanto il pulsante sarà blu se il mouse è posizionato su di esso e verde in caso contrario.

Elenco delle precedenze per l'impostazione delle proprietà di dipendenza

Di seguito viene riportato l'ordine definitivo usato dal sistema di proprietà per l'assegnazione dei valori di runtime delle proprietà di dipendenza. La precedenza più elevata viene elencata per prima. Questo elenco si basa su alcuni dei concetti generali espressi in [Panoramica sulle proprietà di dipendenza](#).

1. **Coercione del sistema di proprietà.** Per informazioni dettagliate sulla coercione, vedere [Coercione, animazione e valore di base](#) più avanti in questo argomento.
2. **Animazioni attive o animazioni con un comportamento di attesa.** Per ottenere qualsiasi effetto pratico, l'animazione di una proprietà deve essere in grado di avere la precedenza sul valore di base (inanimato), anche se quel valore è stato impostato localmente. Per informazioni dettagliate, vedere [Coercione, animazione e valore di base](#) più avanti in questo argomento.
3. **Valore locale.** Un valore locale può essere impostato tramite la praticità della proprietà "wrapper", che equivale anche a impostare come attributo o elemento proprietà in XAML da una chiamata all'API [SetValue](#) utilizzando una proprietà di un'istanza specifica. Se si imposta un valore locale usando un'associazione o una risorsa, ciascuno di questi elementi dispone della precedenza come se fosse stato impostato un valore diretto.
4. **Proprietà del modello `TemplatedParent`.** Un elemento ha un `TemplatedParent` se è stato creato come parte di un modello ([ControlTemplate](#) o [DataTemplate](#)). Per informazioni dettagliate sulle situazioni in cui viene applicato, vedere [TemplatedParent](#) più avanti in questo argomento. All'interno del modello, viene applicata la seguente precedenza:
 - a. Trigger dal modello di [TemplatedParent](#).
 - b. Set di proprietà (in genere tramite XAML attributi) nel modello di [TemplatedParent](#).
5. **Stile implicito.** Si applica solo alla proprietà `Style`. La proprietà `Style` viene riempita da qualsiasi risorsa di stile con una chiave che corrisponde al tipo di quell'elemento. Quella risorsa di stile deve essere presente nella pagina o nell'applicazione. La ricerca per una risorsa di stile隐式 non viene eseguita nei temi.
6. **Trigger degli stili.** I trigger all'interno di stili da una pagina o un'applicazione (questi stili potrebbero essere stili espliciti o impliciti, ma non derivati dagli stili predefiniti, che hanno una precedenza inferiore).
7. **Trigger dei modelli.** Qualsiasi trigger da un modello all'interno di uno stile oppure un modello applicato direttamente.
8. **Setter di stili.** Valori da un `Setter` all'interno di stili dalla pagina o dall'applicazione.
9. **Stile (tema) predefinito.** Per informazioni dettagliate sui casi in cui viene applicato e sul modo in cui gli stili del tema si riferiscono ai modelli all'interno degli stili del tema, vedere [Stili \(tema\) predefiniti](#) più avanti in questo argomento. All'interno di uno stile predefinito, viene applicato il seguente ordine di precedenza:
 - a. Trigger attivi nello stile del tema.
 - b. Setter nello stile del tema.
10. **Ereditarietà.** Alcune proprietà di dipendenza ereditano i propri valori dall'elemento padre agli elementi figlio, in modo che non sia necessario impostarli in modo specifico per ogni elemento in tutta

l'applicazione. Per informazioni dettagliate, vedere [Ereditarietà del valore della proprietà](#).

11. **Valore predefinito dai metadati delle proprietà di dipendenza.** Qualsiasi proprietà di dipendenza specificata può avere un valore predefinito come stabilito dalla registrazione del sistema di proprietà di quella particolare proprietà. Inoltre, le classi derivate che ereditano una proprietà di dipendenza hanno la possibilità di eseguire l'override di tali metadati (incluso il valore predefinito) in base al tipo. Per altre informazioni, vedere [Metadati delle proprietà di dipendenza](#). Dato che l'ereditarietà viene controllata prima del valore predefinito, per una proprietà ereditata un valore predefinito dell'elemento padre ha la precedenza su un elemento figlio. Di conseguenza, se una proprietà ereditabile non viene impostata ovunque, viene usato il valore predefinito come specificato nella radice o nell'elemento padre invece del valore predefinito dell'elemento figlio.

TemplatedParent

TemplatedParent come un elemento di precedenza non si applica a qualsiasi proprietà di un elemento che viene dichiarata direttamente nel markup dell'applicazione standard. Il concetto di TemplatedParent esiste solo per elementi figlio all'interno di una struttura ad albero visuale che vengono creati tramite l'applicazione del modello. Quando il sistema di proprietà Cerca un valore nel modello di [TemplatedParent](#), sta cercando il modello che ha creato tale elemento. I valori delle proprietà del modello di [TemplatedParent](#) in genere agiscono come se fossero impostati come valore locale nell'elemento figlio, ma questa precedenza minore rispetto al valore locale esiste perché i modelli sono potenzialmente condivisi. Per informazioni dettagliate, vedere [TemplatedParent](#).

Proprietà Style

L'ordine di ricerca descritto in precedenza si applica a tutte le possibili proprietà di dipendenza tranne una: la proprietà [Style](#). La proprietà [Style](#) è univoca in quanto non è possibile applicare uno stile, quindi gli elementi di precedenza da 5 a 8 non vengono applicati. Inoltre, l'animazione o la coercizione di [Style](#) non è consigliata (e l'animazione [Style](#) richiederebbe una classe di animazione personalizzata). In questo modo è possibile impostare la proprietà [Style](#):

- **Stile esplicito.** La proprietà [Style](#) è impostata direttamente. Nella maggior parte degli scenari, lo stile non viene definito inline, ma viene invece indicato come una risorsa, tramite una chiave esplicita. In questo caso la stessa proprietà [Style](#) agisce come se fosse un valore locale, elemento di precedenza 3.
- **Stile implicito.** La proprietà [Style](#) non è impostata direttamente. Tuttavia, il [Style](#) esiste a un certo livello nella sequenza di ricerca delle risorse (pagina, applicazione) e viene immesso con una chiave di risorsa corrispondente al tipo a cui deve essere applicato lo stile. In questo caso, la proprietà [Style](#) stesso agisce con una precedenza identificata nella sequenza come elemento 5. Questa condizione può essere rilevata utilizzando [DependencyPropertyHelper](#) sulla proprietà [Style](#) e cercando [ImplicitStyleReference](#) nei risultati.
- **Stile predefinito**, noto anche come **stile del tema**. La proprietà [Style](#) non è impostata direttamente e in realtà verrà letta come `null` fino alla fase di esecuzione. In questo caso, lo stile deriva dalla valutazione del tema di runtime che fa parte del motore di presentazione WPF.

Per gli stili impliciti non presenti nei temi, il tipo deve corrispondere esattamente a un [MyButton](#) classe derivata da [Button](#) non utilizzerà in modo implicito uno stile per [Button](#).

Stili (tema) predefiniti

Ogni controllo fornito con WPF dispone di uno stile predefinito. Potenzialmente, lo stile predefinito varia in base al tema, motivo per il quale questo stile predefinito viene indicato in alcuni casi come uno stile del tema.

Le informazioni più importanti trovate all'interno di uno stile predefinito per un controllo sono il relativo

modello di controllo, che esiste nello stile del tema come Setter per la relativa proprietà [Template](#). Se non esistesse un modello derivato dagli stili predefiniti, un controllo senza un modello personalizzato come parte di uno stile personalizzato non presenterebbe alcun aspetto visivo. Il modello derivato dallo stile predefinito fornisce all'aspetto visivo di ciascun controllo una struttura di base e definisce anche le connessioni tra proprietà definite nella struttura ad albero visuale del modello e la classe di controlli corrispondente. Ogni controllo espone un set di proprietà che può influenzare l'aspetto visivo del controllo senza sostituire completamente il modello. Si consideri, ad esempio, l'aspetto visivo predefinito di un controllo [Thumb](#), che è un componente di un [ScrollBar](#).

Una [Thumb](#) dispone di determinate proprietà personalizzabili. Il modello predefinito di un [Thumb](#) crea una struttura di base/struttura ad albero visuale con diversi componenti [Border](#) annidati per creare un aspetto smussato. Se una proprietà che fa parte del modello deve essere esposta per la personalizzazione da parte della classe [Thumb](#), tale proprietà deve essere esposta da un oggetto [TemplateBinding](#), all'interno del modello. Nel caso di [Thumb](#), diverse proprietà di questi bordi condividono un'associazione di modelli a proprietà quali [Background](#) o [BorderThickness](#). Ma certe altre proprietà o disposizioni visive sono hardcoded nel modello di controllo o sono associate a valori che derivano direttamente dal tema e non possono essere modificate senza sostituire l'intero modello. Generalmente, se una proprietà deriva da un elemento padre basato su modelli e non viene esposta da un'associazione di modelli, non può essere regolata dagli stili poiché non esiste un modo semplice per fare riferimento ad essa. Ma tale proprietà potrebbe essere ancora influenzata dall'ereditarietà dei valori della proprietà nel modello applicato oppure dal valore predefinito.

Gli stili del tema usano un tipo come chiave nelle definizioni. Tuttavia, quando i temi vengono applicati a una determinata istanza dell'elemento, la ricerca dei temi per questo tipo viene eseguita controllando la proprietà [DefaultStyleKey](#) su un controllo. Questa operazione è in contrasto con l'uso del tipo letterale, adottato dagli stili impliciti. Il valore di [DefaultStyleKey](#) eriterà le classi derivate anche se l'implementatore non lo ha modificato (il modo previsto per modificare la proprietà non è eseguirne l'override a livello di proprietà, ma per modificare il valore predefinito nei metadati della proprietà). Questo riferimento indiretto consente alle classi di base di definire gli stili del tema per gli elementi derivati che non dispongono di un altro stile oppure, in un caso più importante, che non dispongono di un modello all'interno di quello stile e che di conseguenza non avrebbero alcun aspetto visivo predefinito. Pertanto, è possibile derivare [MyButton](#) da [Button](#) e ottenere comunque il [Button](#) modello predefinito. Se si fosse l'autore del controllo di [MyButton](#) e si volesse un comportamento diverso, era possibile eseguire l'override dei metadati della proprietà di dipendenza per [DefaultStyleKey](#) su [MyButton](#) per restituire una chiave diversa, quindi definire gli stili del tema pertinenti, incluso il modello per [MyButton](#) che è necessario creare un pacchetto con il controllo [MyButton](#). Per altri dettagli su temi, stili e creazione di controlli, vedere [Cenni preliminari sulla modifica di controlli](#).

Riferimenti e associazione di risorse dinamiche

I riferimenti alle risorse dinamiche e le operazioni di associazione rispettano la precedenza della posizione su cui sono impostati. Ad esempio, una risorsa dinamica applicata a un valore locale agisce in base all'elemento di precedenza 3, a un'associazione per un setter di proprietà all'interno di un stile del tema viene applicato un elemento di precedenza 9 e così via. Dato che i riferimenti e l'associazione per le risorse dinamiche devono essere in grado di ottenere valori dallo stato della fase di esecuzione dell'applicazione, ciò comporta che anche il processo effettivo di determinazione della precedenza del valore di proprietà per qualsiasi proprietà specificata si estende nella fase di esecuzione.

I riferimenti alle risorse dinamiche non fanno parte in senso stretto del sistema di proprietà, ma hanno un proprio ordine di ricerca che interagisce con la sequenza elencata in precedenza. Tale precedenza è documentata più approfonditamente in [Risorse XAML](#). La somma di base di tale precedenza è: elemento alla radice della pagina, applicazione, tema, sistema.

Le risorse dinamiche e le associazioni hanno la precedenza relativamente a dove sono state impostate, ma il valore viene rinviato. Una conseguenza di ciò è che, se si imposta una risorsa dinamica o un'associazione su un valore locale, qualsiasi modifica al valore locale sostituisce completamente la risorsa dinamica o l'associazione.

Anche se si chiama il metodo [ClearValue](#) per cancellare il valore impostato localmente, la risorsa dinamica o l'associazione non verrà ripristinata. Infatti, se si chiama [ClearValue](#) su una proprietà con una risorsa dinamica o un'associazione sul posto (senza valore locale letterale), vengono cancellate anche dalla chiamata [ClearValue](#).

SetCurrentValue

Il metodo [SetCurrentValue](#) è un altro modo per impostare una proprietà, ma non è in ordine di precedenza. [SetCurrentValue](#) consente invece di modificare il valore di una proprietà senza sovrascrivere l'origine di un valore precedente. È possibile utilizzare [SetCurrentValue](#) ogni volta che si desidera impostare un valore senza assegnare tale valore alla precedenza di un valore locale. Se, ad esempio, una proprietà viene impostata da un trigger e quindi viene assegnato un altro valore tramite [SetCurrentValue](#), il sistema di proprietà rispetta il trigger e la proprietà verrà modificata se si verifica l'azione del trigger. [SetCurrentValue](#) consente di modificare il valore della proprietà senza assegnarle un'origine con precedenza maggiore. Analogamente, è possibile utilizzare [SetCurrentValue](#) per modificare il valore di una proprietà senza sovrascrivere un'associazione.

Coercizione, animazioni e valore di base

La coercione e l'animazione agiscono entrambi su un valore definito come "valore di base" in tutto questo SDK. Di conseguenza, il valore di base è un qualsiasi valore determinato tramite la valutazione verso l'alto negli elementi fino a raggiungere l'elemento 2.

Per un'animazione, il valore di base può avere un effetto sul valore animato, se quell'animazione non specifica le impostazioni "From" e "To" per determinati comportamenti o se l'animazione ripristina intenzionalmente il valore di base quando viene completata. Per osservare questo comportamento nella pratica, eseguire l'[Esempio valori di destinazione dell'animazione From/To/By](#). Tentare di impostare i valori locali dell'altezza del rettangolo nell'esempio, in modo che il valore locale iniziale sia diverso da qualsiasi impostazione "From" nell'animazione. Si noterà che le animazioni iniziano immediatamente usando i valori "From" valori e sostituiscono il valore di base una volta avviate. L'animazione potrebbe specificare per tornare al valore trovato prima dell'animazione dopo che è stato completato specificando l'[FillBehavior](#) stop. In seguito, verrà usata la precedenza normale per la determinazione del valore di base.

Possono essere applicate più animazioni a una sola proprietà, ognuna delle quali definita da punti diversi nella precedenza dei valori. Tuttavia, queste animazioni comporranno i relativi valori, piuttosto che applicare semplicemente l'animazione dalla precedenza più elevata. Ciò dipende dal modo preciso in cui sono definite le animazioni e dal tipo del valore che viene animato. Per altre informazioni sull'animazione di proprietà, vedere [Cenni preliminari sull'animazione](#).

La coercione si applica al livello più elevato in assoluto. Anche un'animazione già in esecuzione è soggetta alla coercione del valore. Alcune proprietà di dipendenza esistenti in WPF dispongono di una coercione incorporata. Per una proprietà di dipendenza personalizzata, si definisce il comportamento di coercione per una proprietà di dipendenza personalizzata scrivendo un [CoerceValueCallback](#) e passando il callback come parte dei metadati quando si crea la proprietà. È anche possibile eseguire l'override del comportamento di coercione di proprietà esistenti eseguendo l'override dei metadati di quella proprietà in una classe derivata. La coercione interagisce con il valore di base in modo tale che i vincoli di coercione vengano applicati come i vincoli esistenti in quel momento, mantenendo tuttavia il valore di base. Pertanto, se i vincoli nella coercione vengono rimossi in un secondo momento, la coercione restituirà il valore più vicino possibile a quel valore di base e l'influenza della coercione su una proprietà cesserà potenzialmente appena vengono rimossi tutti i vincoli. Per altre informazioni sul comportamento di coercione, vedere [Callback e convalida delle proprietà di dipendenza](#).

Comportamenti dei trigger

I controlli spesso definiscono i comportamenti dei trigger come parte dello stile predefinito nei temi. L'impostazione di proprietà locali sui controlli può impedire ai trigger di essere in grado di rispondere agli

eventi generati dagli utenti sia da un punto di vista visivo sia da un punto di vista di comportamento. L'uso più comune di un trigger di proprietà è per le proprietà di controllo o di stato, ad esempio [IsSelected](#). Per impostazione predefinita, ad esempio, quando un [Button](#) è disabilitato (il trigger per [IsEnabled](#) è `false`), il valore di [Foreground](#) nello stile del tema è quello che fa sì che il controllo venga visualizzato in grigio. Tuttavia, se è stato impostato un valore di [Foreground](#) locale, il normale colore grigio viene sovraregolato in base alla priorità del set di proprietà locali, anche in questo scenario attivato dalla proprietà. Prestare attenzione nell'impostazione di valori per proprietà che presentano comportamenti di trigger a livello di tema e accertarsi di non interferire impropriamente con l'esperienza utente prevista per quel controllo.

ClearValue e precedenza dei valori

Il metodo [ClearValue](#) fornisce un espediente per cancellare qualsiasi valore applicato localmente da una proprietà di dipendenza impostata su un elemento. Tuttavia, la chiamata di [ClearValue](#) non garantisce che l'impostazione predefinita, come stabilito nei metadati durante la registrazione della proprietà, sia il nuovo valore effettivo. Tutti gli altri partecipanti nella precedenza dei valori sono ancora attivi. Solo il valore impostato localmente è stato rimosso dalla sequenza di precedenza. Se ad esempio si chiama [ClearValue](#) su una proprietà in cui tale proprietà viene impostata anche da uno stile del tema, il valore del tema viene applicato come nuovo valore anziché come predefinito basato sui metadati. Se si desidera che tutti i partecipanti del valore della proprietà vengano esclusi dal processo e il valore venga impostato sull'impostazione predefinita dei metadati registrati, è possibile ottenere tale valore predefinito eseguendo una query sui metadati della proprietà di dipendenza, quindi è possibile utilizzare il valore predefinito per impostare localmente la proprietà con una chiamata a [SetValue](#).

Vedere anche

- [DependencyObject](#)
- [DependencyProperty](#)
- [Cenni preliminari sulle proprietà di dipendenza](#)
- [Proprietà di dipendenza personalizzate](#)
- [Callback e convalida delle proprietà di dipendenza](#)

Proprietà di dipendenza di sola lettura

04/11/2019 • 8 minutes to read • [Edit Online](#)

Questo argomento descrive le proprietà di dipendenza di sola lettura, incluse le proprietà di dipendenza di sola lettura esistenti e gli scenari e le tecniche per la creazione di una proprietà di dipendenza di sola lettura personalizzata.

Prerequisites

Nell'argomento si presuppone la conoscenza degli scenari di base dell'implementazione di una proprietà di dipendenza e del modo in cui i metadati vengono applicati a una proprietà di dipendenza personalizzata. Per il contesto, vedere [Proprietà di dipendenza personalizzate](#) e [Metadati delle proprietà di dipendenza](#).

Proprietà di dipendenza di sola lettura esistenti

Alcune delle proprietà di dipendenza definite nel framework Windows Presentation Foundation (WPF) sono di sola lettura. La ragione più comune per specificare una proprietà di dipendenza di sola lettura è che si tratta di proprietà che devono essere usate per la determinazione dello stato. Se però lo stato è influenzato da una molteplicità di fattori, la semplice impostazione della proprietà su quello stato non rappresenta la soluzione appropriata nell'ottica della progettazione di un'interfaccia utente. Ad esempio, la proprietà [IsMouseOver](#) è semplicemente lo stato di emersione come determinato dall'input del mouse. Ogni tentativo di impostare questo valore a livello di codice aggirando il vero input del mouse sarebbe imprevedibile e potrebbe causare incongruenze.

Dal momento che non possono essere impostate, le proprietà di dipendenza di sola lettura non sono appropriate per molti degli scenari in cui in genere offrono una soluzione (vale a dire: data binding, possibilità di applicare direttamente uno stile a un valore, convalida, animazione, ereditarietà). Nonostante non possano essere impostate, queste proprietà hanno in ogni caso alcune delle funzionalità aggiuntive supportate dalle proprietà di dipendenza nel sistema di proprietà. La funzionalità più importante consiste nel fatto che la proprietà di dipendenza di sola lettura può essere usata come trigger di proprietà in uno stile. Non è possibile abilitare i trigger con una normale proprietà Common Language Runtime (CLR); deve essere una proprietà di dipendenza. Il [IsMouseOver](#) proprietà precedente è un esempio perfetto di uno scenario in cui potrebbe essere molto utile definire uno stile per un controllo, in cui alcune proprietà visibili, ad esempio uno sfondo, un primo piano o proprietà simili di elementi composti all'interno del controllo modificare quando l'utente posiziona il mouse su un'area definita del controllo. Le modifiche a una proprietà di dipendenza di sola lettura possono anche essere rilevate e segnalate dai processi di invalidamento inerenti del sistema di proprietà, che infatti è dotato del supporto interno della funzionalità del trigger di proprietà.

Creazione di proprietà di dipendenza di sola lettura personalizzate

Assicurarsi di leggere la sezione precedente sui motivi per cui le proprietà di dipendenza di sola lettura non funzionano per molti scenari comuni di proprietà di dipendenza. Se tuttavia si ha uno scenario adatto, è possibile creare una proprietà di dipendenza di sola lettura personalizzata.

Gran parte del processo di creazione di una proprietà di dipendenza di sola lettura è identico a quello descritto negli argomenti [Proprietà di dipendenza personalizzate](#) e [Implementare una proprietà di dipendenza](#). Vi sono tre differenze importanti:

- Quando si registra la proprietà, chiamare il metodo [RegisterReadOnly](#) anziché il normale metodo [Register](#) per la registrazione della proprietà.

- Quando si implementa la proprietà "wrapper" CLR, verificare che anche il wrapper non disponga di un'implementazione impostata, in modo che non vi sia alcuna incoerenza nello stato di sola lettura per il wrapper pubblico esposto.
- L'oggetto restituito dalla registrazione di sola lettura è [DependencyPropertyKey](#) anziché [DependencyProperty](#). Anche se è necessario archiviare il campo come membro, questo in genere non viene reso un membro pubblico del tipo.

Naturalmente, il valore o campo privato sottostante della proprietà di dipendenza di sola lettura può essere scritto usando qualsiasi logica. Tuttavia, il modo più semplice per impostare la proprietà inizialmente o come parte della logica di runtime consiste nell'usare le API del sistema di proprietà, anziché aggirare il sistema di proprietà e impostare direttamente il campo di supporto privato. In particolare, esiste una firma di [SetValue](#) che accetta un parametro di tipo [DependencyPropertyKey](#). La modalità e la posizione in cui questo valore viene impostato a livello di codice all'interno della logica dell'applicazione influirà sul modo in cui è possibile impostare l'accesso per il [DependencyPropertyKey](#) creato al momento della prima registrazione della proprietà di dipendenza. Se si gestisce tutta questa logica all'interno della classe è possibile renderla privata o, se è necessario impostarla da un'altra porzione dell'assembly, è possibile impostarla come interna. Un approccio consiste nel chiamare [SetValue](#) all'interno di un gestore eventi di classe di un evento pertinente che informa un'istanza della classe che è necessario modificare il valore della proprietà archiviata. Un altro approccio consiste nel collegare le proprietà di dipendenza utilizzando [PropertyChangedCallback](#) abbinati e callback di [CoerceValueCallback](#) come parte dei metadati di tali proprietà durante la registrazione.

Poiché la [DependencyPropertyKey](#) è privata e non viene propagata dal sistema di proprietà all'esterno del codice, una proprietà di dipendenza di sola lettura ha una maggiore sicurezza dell'impostazione rispetto a una proprietà di dipendenza di lettura/scrittura. Per una proprietà di dipendenza di lettura e scrittura, il campo di identificazione è pubblico in modo esplicito oppure implicito, per cui la proprietà può essere impostata senza alcuna limitazione. Per informazioni più specifiche, vedere [Sicurezza della proprietà di dipendenza](#).

Vedere anche

- [Panoramica sulle proprietà di dipendenza](#)
- [Proprietà di dipendenza personalizzate](#)
- [Applicazione di stili e modelli](#)

Ereditarietà del valore della proprietà

23/10/2019 • 9 minutes to read • [Edit Online](#)

L'ereditarietà del valore della proprietà è una funzionalità del sistema di proprietà Windows Presentation Foundation (WPF). L'ereditarietà del valore della proprietà consente agli elementi figlio in un albero di elementi di ottenere il valore di una determinata proprietà dagli elementi padre, ereditando tale valore con le impostazioni specificate in un punto qualsiasi dell'elemento padre più vicino. Anche l'elemento padre potrebbe aver ottenuto il valore tramite l'ereditarietà del valore della proprietà, pertanto il sistema potrebbe procedere in modo ricorsivo fino alla radice della pagina. L'ereditarietà del valore della proprietà non è il comportamento predefinito del sistema di proprietà. È necessario stabilire una particolare impostazione dei metadati di una proprietà per fare in modo che quest'ultima attivi l'ereditarietà del valore per gli elementi figlio.

L'ereditarietà del valore della proprietà è un'ereditarietà di contenimento

Il termine "ereditarietà" usato in questa sede non coincide perfettamente con il concetto di ereditarietà nel contesto dei tipi e in generale della programmazione orientata a oggetti, in cui le classi derivate ereditano le definizioni dei membri dalle rispettive classi di base. Questo significato di ereditarietà è valido anche in WPF: le proprietà definite nelle varie classi di base sono esposte come attributi per le classi XAML derivate, se usate come elementi ed esposte come membri per il codice. L'ereditarietà del valore della proprietà riguarda in particolare il modo in cui i valori delle proprietà possono ereditare da un elemento a un altro in base alle relazioni padre-figlio all'interno di un albero di elementi. L'albero di elementi è visibile più direttamente quando si annidano elementi all'interno di altri elementi in fase di definizione delle applicazioni nel markup XAML. Gli alberi di oggetti possono essere creati anche a livello di codice, aggiungendo oggetti a raccolte designate di altri oggetti. In questo caso la modalità di funzionamento dell'ereditarietà del valore della proprietà sarà la stessa nell'albero completato in fase di esecuzione.

Applicazioni pratiche dell'ereditarietà del valore della proprietà

Le WPF API includono diverse proprietà con ereditarietà della proprietà abilitata. In genere, lo scenario per queste proprietà prevede l'uso di una proprietà che possa essere impostata una sola volta per pagina, ma che sia anche un membro di una delle classi dell'elemento di base e quindi esista anche nella maggior parte degli elementi figlio. Ad esempio, la [FlowDirection](#) proprietà controlla la direzione in cui il contenuto propagato deve essere presentato e disposto nella pagina. Nella maggior parte dei casi si preferisce che il concetto di flusso del testo sia gestito coerentemente in tutti gli elementi figlio. Se per qualche motivo la direzione del flusso è stata reimpostata in un livello dell'albero di elementi dall'utente o dall'ambiente, la reimpostazione deve essere in genere eseguita per tutta la struttura. Quando si [FlowDirection](#) crea la proprietà per ereditare, il valore deve essere impostato o reimpostato una sola volta al livello nell'albero degli elementi che comprende le esigenze di presentazione di ogni pagina dell'applicazione. Anche il valore predefinito iniziale userà l'ereditarietà nello stesso modo. Il modello di ereditarietà del valore della proprietà consente in ogni caso ai singoli elementi di reimpostare il valore nelle rare occasioni in cui si sceglie intenzionalmente di usare una combinazione di direzioni di flusso.

Rendere ereditabile una proprietà personalizzata

Modificando i metadati di una proprietà personalizzata è possibile rendere ereditabili anche le proprietà di questo tipo. Si noti, tuttavia, che quando si definisce una proprietà come ereditabile è necessario fare alcune considerazioni sulle prestazioni. Nei casi in cui quella proprietà non dispone di un valore locale stabilito oppure di un valore ottenuto tramite stili, modelli o data binding, una proprietà ereditabile fornisce i relativi valori di

proprietà assegnati a tutti gli elementi figlio dell'albero logico.

Per fare in modo che una proprietà partecipi all'ereditarietà del valore, creare una proprietà associata personalizzata, come descritto in [Registrare una proprietà associata](#). Registrare la proprietà con i metadati `FrameworkPropertyMetadata()` e specificare l'opzione "Inherits" nelle impostazioni delle opzioni all'interno dei metadati. Verificare quindi che la proprietà abbia un valore predefinito stabilito, perché quel valore sarà ereditato. Anche se la proprietà è stata registrata come associata, può essere necessario creare un "wrapper" della proprietà per ottenere o impostare l'accesso sul tipo di proprietario, come si farebbe per una proprietà di dipendenza non associata. Al termine di questa operazione, è possibile impostare la proprietà ereditabile usando il wrapper della proprietà Direct sul tipo di proprietario o sui tipi derivati oppure è possibile impostarla usando la sintassi della proprietà `DependencyObject` associata per qualsiasi.

Le proprietà associate sono concettualmente simili alle proprietà globali. È possibile verificare il valore in qualsiasi `DependencyObject` e ottenere un risultato valido. Lo scenario tipico per le proprietà associate è quello di impostare i valori delle proprietà negli elementi figlio e tale scenario è più efficace se la proprietà in questione è una proprietà associata che è sempre presente in modo implicito come proprietà associata in ogni elemento (`DependencyObject`) nell'albero.

NOTE

Anche se può sembrare che l'ereditarietà del valore della proprietà funzioni per le proprietà di dipendenza non associate, il comportamento di ereditarietà per una proprietà non associata tramite certi limiti di elementi nell'albero della fase di esecuzione non è definito. Usare `RegisterAttached` sempre per registrare le proprietà specificate `Inherits` nei metadati.

Eredità dei valori di proprietà attraverso i limiti dell'albero

L'ereditarietà delle proprietà funziona mediante il passaggio attraverso un albero di elementi. Quest'albero spesso è parallelo all'albero logico. Tuttavia, ogni volta che si include un oggetto a livello di core WPF nel markup che definisce un albero degli elementi, `Brush` ad esempio, è stato creato un albero logico discontinuo. Un vero albero logico non si estende concettualmente attraverso `Brush`, perché l'albero logico è un concetto a livello di Framework WPF. È possibile osservare questo riflesso nei risultati quando si utilizzano i metodi di `LogicalTreeHelper`. Tuttavia, l'ereditarietà del valore della proprietà può colmare questo gap nell'albero logico e può comunque passare i valori ereditati fino a quando la proprietà ereditabile è stata registrata come proprietà associata e nessun limite di blocco dell'ereditarietà intenzionale (ad esempio, `Frame`) rilevato.

Vedere anche

- [Metadati delle proprietà di dipendenza](#)
- [Cenni preliminari sulle proprietà associate](#)
- [Precedenza del valore della proprietà di dipendenza](#)

Sicurezza delle proprietà di dipendenza

23/10/2019 • 5 minutes to read • [Edit Online](#)

Le proprietà di dipendenza in genere devono essere considerate come proprietà pubbliche. A causa della natura stessa del sistema di proprietà di Windows Presentation Foundation (WPF), è impossibile avere delle garanzie di sicurezza sul valore di una proprietà di dipendenza.

Accesso e sicurezza delle proprietà wrapper e di dipendenza

In genere, le proprietà di dipendenza vengono implementate insieme alle proprietà "wrapper" Common Language Runtime (CLR) che semplificano il recupero o l'impostazione della proprietà da un'istanza di. Tuttavia, i wrapper sono solo metodi pratici che implementano le chiamate `GetValue` statiche `SetValue` e sottostanti utilizzate quando si interagisce con le proprietà di dipendenza. Considerandola in un altro modo, le proprietà vengono esposte come proprietà Common Language Runtime (CLR) che vengono supportate da una proprietà di dipendenza anziché da un campo privato. I meccanismi di sicurezza applicati ai wrapper non sono paralleli al comportamento del sistema di proprietà e all'accesso della proprietà di dipendenza sottostante. L'inserimento di una richiesta di sicurezza sul wrapper impedisce solo l'utilizzo del metodo convenience, ma non impedisce le chiamate `GetValue` a `SetValue`. Analogamente, inserendo un livello di accesso protetto o privato sui wrapper non viene garantita alcuna sicurezza effettiva.

Se si scrivono proprietà di dipendenza personalizzate, è necessario dichiarare i wrapper e il `DependencyProperty` campo identificatore come membri pubblici, in modo che i chiamanti non ottengano informazioni fuorvianti sul livello di accesso reale della proprietà (a causa dell'archivio implementata come proprietà di dipendenza).

Per una proprietà di dipendenza personalizzata, è possibile registrare la proprietà come proprietà di dipendenza di sola lettura e questo fornisce un metodo efficace per impedire che una proprietà venga impostata da chiunque che non contenga un riferimento a per `DependencyPropertyKey` la proprietà. Per altre informazioni, vedere [Proprietà di dipendenza di sola lettura](#).

NOTE

La dichiarazione di `DependencyProperty` un campo identificatore privato non è consentita e può essere usata per ridurre lo spazio dei nomi esposto immediatamente di una classe personalizzata, ma tale proprietà non deve essere considerata "privata" nello stesso senso del linguaggio comune le definizioni del linguaggio Runtime (CLR) definiscono il livello di accesso per i motivi descritti nella sezione successiva.

Esposizione di proprietà di dipendenza del sistema di proprietà

Non è in genere utile ed è potenzialmente fuorviante dichiarare come un `DependencyProperty` livello di accesso diverso da Public. L'impostazione di questo livello di accesso impedisce solo che qualcuno possa ottenere un riferimento all'istanza dalla classe dichiarante. Esistono tuttavia diversi aspetti del sistema di proprietà che restituiscono un oggetto `DependencyProperty` come mezzo per identificare una particolare proprietà esistente in un'istanza di una classe o un'istanza della classe derivata e questo identificatore è ancora utilizzabile in una `SetValue` chiamata anche Se l'identificatore statico originale viene dichiarato come non pubblico. Inoltre, `OnPropertyChanged` i metodi virtuali ricevono informazioni di qualsiasi proprietà di dipendenza esistente che ha modificato il valore. Inoltre, il metodo `GetLocalValueEnumerator` restituisce gli identificatori per qualsiasi proprietà nelle istanze con un valore impostato localmente.

Convalida e sicurezza

Se si applica una richiesta `ValidateValueCallback` a un oggetto e si prevede che l'errore di convalida in caso di

errore della richiesta per impedire l'impostazione di una proprietà non è un meccanismo di sicurezza adeguato. L'invalidamento set-value applicato tramite [ValidateValueCallback](#) può anche essere eliminato da chiamanti malintenzionati, se tali chiamanti operano all'interno del dominio applicazione.

Vedere anche

- [Proprietà di dipendenza personalizzate](#)

Modelli di costruttore sicuri per DependencyObject

25/11/2019 • 10 minutes to read • [Edit Online](#)

In genere, i costruttori di classe non devono chiamare callback come metodi virtuali o delegati, in quanto possono essere chiamati come inizializzazione di base di costruttori per una classe derivata. L'uso di elementi virtuali può avvenire in un stato incompleto dell'inizializzazione di qualsiasi dato oggetto. Il sistema di proprietà stesso, tuttavia, chiama ed espone internamente i callback come parte del sistema di proprietà di dipendenza. Come operazione semplice, l'impostazione di un valore della proprietà di dipendenza con [SetValue](#) chiamata può includere potenzialmente un callback in un punto qualsiasi della determinazione. Per questa ragione, occorre prestare attenzione quando si impostano i valori delle proprietà di dipendenza all'interno del corpo di un costruttore, perché l'operazione può divenire problematica se il tipo viene usato come classe di base. È disponibile un modello specifico per l'implementazione di costruttori di [DependencyObject](#) che consente di evitare problemi specifici con gli Stati delle proprietà di dipendenza e i callback inerenti, documentati qui.

Metodi virtuali del sistema di proprietà

I metodi virtuali o i callback seguenti vengono potenzialmente chiamati durante i calcoli della chiamata [SetValue](#) che imposta un valore della proprietà di dipendenza: [ValidateValueCallback](#), [PropertyChangedCallback](#), [CoerceValueCallbackOnPropertyChanged](#). Ognuno di questi metodi virtuali o callback serve a uno scopo particolare nell'espansione della versatilità del sistema di proprietà Windows Presentation Foundation (WPF) e delle proprietà di dipendenza. Per altre informazioni su come usare questi elementi virtuali per personalizzare la determinazione del valore della proprietà, vedere [Callback e convalida delle proprietà di dipendenza](#).

Confronto tra imposizione di regole FXCop e virtuali del sistema di proprietà

Se si usa lo strumento Microsoft FXCop come parte del processo di compilazione e si esegue la derivazione da determinate classi del framework WPF che chiamano il costruttore di base oppure si implementano proprietà di dipendenza personalizzate sulle classi derivate, può verificarsi la violazione di una particolare regola FXCop. La stringa del nome di questa violazione è:

`DoNotCallOverridableMethodsInConstructors`

Si tratta di una regola che fa parte del set di regole pubblico predefinito per FXCop. È possibile che questa regola segnali una traccia tramite il sistema di proprietà di dipendenza che infine chiama il metodo virtuale di un sistema di proprietà di dipendenza. La violazione di questa regola potrebbe continuare ad apparire anche dopo avere seguito i modelli del costruttore consigliati illustrati in questo argomento, pertanto potrebbe essere necessario disabilitare o eliminare la regola nella configurazione del set di regole FXCop.

La maggior parte dei problemi è causata dalla derivazione delle classi e dal mancato uso delle classi esistenti

I problemi segnalati da questa regola si verificano quando una classe implementata con metodi virtuali nella relativa sequenza di costruzione viene successivamente derivata. Se la classe viene contrassegnata come sealed oppure si sa o si decide che la classe non sarà derivata, le considerazioni qui illustrate e i problemi che hanno motivato la regola FXCop non si applicano a questa situazione. Se tuttavia se si stanno creando classi in modo che vengano usate come classi di base, ad esempio se si stanno creando modelli o un set di librerie di controlli espandibile, è consigliabile seguire i modelli raccomandati in questo argomento per i costruttori.

I costruttori predefiniti devono inizializzare tutti i valori richiesti dai callback

Tutti i membri di istanza usati dalla classe esegue l'override o i callback (i callback dall'elenco nella sezione relativa alle virtuali del sistema di proprietà) devono essere inizializzati nel costruttore senza parametri della classe, anche se alcuni di questi valori sono riempiti da valori "reali" tramite parametri dei costruttori senza parametri.

L'esempio di codice seguente e i successivi sono esempi di pseudo codice C# in cui questa regola viene violata e

in cui viene illustrato il problema:

```
public class MyClass : DependencyObject
{
    public MyClass() {}
    public MyClass(object toSetWobble)
        : this()
    {
        Wobble = toSetWobble; //this is backed by a DependencyProperty
        _myList = new ArrayList(); // this line should be in the default ctor
    }
    public static readonly DependencyProperty WobbleProperty =
        DependencyProperty.Register("Wobble", typeof(object), typeof(MyClass));
    public object Wobble
    {
        get { return GetValue(WobbleProperty); }
        set { SetValue(WobbleProperty, value); }
    }
    protected override void OnPropertyChanged(DependencyPropertyChangedEventArgs e)
    {
        int count = _myList.Count; // null-reference exception
    }
    private ArrayList _myList;
}
```

Quando il codice dell'applicazione chiama `new MyClass(objectvalue)`, chiama il costruttore senza parametri e i costruttori della classe base. Imposta quindi `Property1 = object1`, che chiama il metodo virtuale `OnPropertyChanged` sul `MyClass` proprietario `DependencyObject`. L'override fa riferimento a `_myList`, che non è stato ancora inizializzato.

Un modo per evitare questi problemi consiste nel verificare che i callback usino solo altre proprietà di dipendenza e che ognuna di esse abbia un valore predefinito stabilito come parte dei relativi metadati registrati.

Modelli di costruttore sicuri

Per evitare i rischi di un'inizializzazione incompleta nel caso in cui la classe sia usata come classe di base, seguire questi modelli:

Costruttori senza parametri che chiamano l'inizializzazione di base

Implementare questi costruttori che chiamano l'impostazione predefinita di base:

```
public MyClass : SomeBaseClass {
    public MyClass() : base() {
        // ALL class initialization, including initial defaults for
        // possible values that other ctors specify or that callbacks need.
    }
}
```

Costruttori non predefiniti (di comodo), che non corrispondono ad alcuna firma di base

Se questi costruttori usano i parametri per impostare le proprietà di dipendenza nell'inizializzazione, chiamare prima il costruttore senza parametri della classe per l'inizializzazione e quindi usare i parametri per impostare le proprietà di dipendenza. Queste ultime potrebbero essere proprietà di dipendenza definite dalla classe o proprietà di dipendenza ereditate dalle classi di base. In entrambi i casi, tuttavia, usare il modello seguente:

```
public MyClass : SomeBaseClass {  
    public MyClass(object toSetProperty1) : this() {  
        // Class initialization NOT done by default.  
        // Then, set properties to values as passed in ctor parameters.  
        Property1 = toSetProperty1;  
    }  
}
```

Costruttori non predefiniti (di comodo), che corrispondono alle firme di base

Anziché chiamare il costruttore di base con la stessa parametrizzazione, chiamare di nuovo il costruttore senza parametri della propria classe. Non chiamare l'inizializzatore di base, bensì chiamare `this()`. A questo punto, riprodurre il comportamento del costruttore originale usando i parametri passati come valori per l'impostazione delle proprietà pertinenti. Per informazioni sulla determinazione delle proprietà da impostare tramite determinati parametri, usare la documentazione originale del costruttore di base:

```
public MyClass : SomeBaseClass {  
    public MyClass(object toSetProperty1) : this() {  
        // Class initialization NOT done by default.  
        // Then, set properties to values as passed in ctor parameters.  
        Property1 = toSetProperty1;  
    }  
}
```

Deve corrispondere a tutte le firme

Per i casi in cui il tipo di base ha più firme, è necessario associare intenzionalmente tutte le firme possibili con un'implementazione del costruttore personalizzata che usa il modello consigliato per chiamare il costruttore senza parametri della classe prima di impostare ulteriormente Proprietà.

Impostazione delle proprietà di dipendenza con SetValue

Questi stessi modelli si applicano se si imposta una proprietà che non dispone di un wrapper per l'impostazione della proprietà praticità e si impostano i valori con `SetValue`. Le chiamate a `SetValue` che passano attraverso i parametri del costruttore devono chiamare anche il costruttore senza parametri della classe per l'inizializzazione.

Vedere anche

- [Proprietà di dipendenza personalizzate](#)
- [Panoramica sulle proprietà di dipendenza](#)
- [Sicurezza delle proprietà di dipendenza](#)

Proprietà di dipendenza di tipo raccolta

25/11/2019 • 7 minutes to read • [Edit Online](#)

Questo argomento include linee guida e modelli consigliati per l'implementazione di una proprietà di dipendenza in cui la proprietà è di tipo raccolta.

Implementazione di una proprietà di dipendenza di tipo raccolta

Per una proprietà di dipendenza in generale, il modello di implementazione che segue è la definizione di un wrapper della proprietà CLR, in cui tale proprietà è supportata da un identificatore di [DependencyProperty](#) anziché da un campo o da un altro costrutto. Questo stesso modello viene seguito quando si implementa una proprietà di tipo raccolta. Tuttavia, una proprietà di tipo raccolta introduce una certa complessità al modello ogni volta che il tipo contenuto nella raccolta è a sua volta un [DependencyObject](#) o [Freezable](#) classe derivata.

Inizializzazione della raccolta oltre il valore predefinito

Quando si crea una proprietà di dipendenza, non si specifica il valore predefinito della proprietà come valore iniziale del campo. Al contrario, viene specificato il valore predefinito tramite i metadati della proprietà di dipendenza. Se la proprietà è un tipo di riferimento, il valore predefinito specificato nei metadati della proprietà di dipendenza non è un valore predefinito per istanza, ma un valore predefinito che viene applicato a tutte le istanze del tipo. Pertanto, è necessario prestare attenzione a non usare la singola raccolta statica definita dai metadati della proprietà della raccolta come valore di lavoro predefinito per le istanze del tipo appena create. È invece necessario assicurarsi di impostare deliberatamente il valore della raccolta su un'unica raccolta (istanza), come parte della logica del costruttore di classi. In caso contrario, verrà creata una classe Singleton non intenzionale.

Si osservi l'esempio riportato di seguito. Nella sezione seguente dell'esempio viene illustrata la definizione di una classe `Aquarium`, che contiene un difetto con il valore predefinito. La classe definisce la proprietà di dipendenza del tipo di raccolta `AquariumObjects`, che usa il tipo di `List<T>` generico con un vincolo di tipo `FrameworkElement`. Nella chiamata `Register(String, Type, Type, PropertyMetadata)` per la proprietà di dipendenza, i metadati stabiliscono il valore predefinito come nuovo `List<T>` generico.

WARNING

Il codice seguente non si comporta correttamente.

```

public class Fish : FrameworkElement { }
public class Aquarium : DependencyObject {
    private static readonly DependencyPropertyKey AquariumContentsPropertyKey =
        DependencyProperty.RegisterReadOnly(
            "AquariumContents",
            typeof(List<FrameworkElement>),
            typeof(Aquarium),
            new FrameworkPropertyMetadata(new List<FrameworkElement>())
        );
    public static readonly DependencyProperty AquariumContentsProperty =
        AquariumContentsPropertyKey.DependencyProperty;

    public List<FrameworkElement> AquariumContents
    {
        get { return (List<FrameworkElement>)GetValue(AquariumContentsProperty); }
    }

    // ...
}

```

```

Public Class Fish
    Inherits FrameworkElement
End Class
Public Class Aquarium
    Inherits DependencyObject
    Private Shared ReadOnly AquariumContentsPropertyKey As DependencyProperty =
        DependencyProperty.RegisterReadOnly("AquariumContents", GetType(List(Of FrameworkElement)), GetType(Aquarium),
        New FrameworkPropertyMetadata(New List(Of FrameworkElement)()))
    Public Shared ReadOnly AquariumContentsProperty As DependencyProperty =
        AquariumContentsPropertyKey.DependencyProperty

    Public ReadOnly Property AquariumContents() As List(Of FrameworkElement)
        Get
            Return CType(GetValue(AquariumContentsProperty), List(Of FrameworkElement))
        End Get
    End Property
    ' ...
End Class

```

Tuttavia, se si lascia inalterato il codice, quel singolo valore predefinito dell'elenco viene condiviso per tutte le istanze di `Aquarium`. Se si esegue il codice di test riportato di seguito, destinato a mostrare la modalità di creazione di due istanze di `Aquarium` separate e si aggiunge un singolo oggetto `Fish` diverso a ciascuna di esse, il risultato è sorprendente:

```

Aquarium myAq1 = new Aquarium();
Aquarium myAq2 = new Aquarium();
Fish f1 = new Fish();
Fish f2 = new Fish();
myAq1.AquariumContents.Add(f1);
myAq2.AquariumContents.Add(f2);
MessageBox.Show("aq1 contains " + myAq1.AquariumContents.Count.ToString() + " things");
MessageBox.Show("aq2 contains " + myAq2.AquariumContents.Count.ToString() + " things");

```

```

Dim myAq1 As New Aquarium()
Dim myAq2 As New Aquarium()
Dim f1 As New Fish()
Dim f2 As New Fish()
myAq1.AquariumContents.Add(f1)
myAq2.AquariumContents.Add(f2)
MessageBox.Show("aq1 contains " & myAq1.AquariumContents.Count.ToString() & " things")
MessageBox.Show("aq2 contains " & myAq2.AquariumContents.Count.ToString() & " things")

```

Invece di contenere un solo elemento, ogni raccolta ne contiene due. Questa situazione si verifica in quanto ciascun oggetto `Aquarium` ha aggiunto il proprio oggetto `Fish` alla raccolta dei valori predefiniti, generata da una singola chiamata al costruttore nei metadati e pertanto condivisa tra tutte le istanze. Si tratta di una situazione quasi mai auspicabile.

Per risolvere questo problema, è necessario reimpostare il valore della proprietà di dipendenza della raccolta su un'unica istanza, come parte della chiamata al costruttore di classe. Poiché la proprietà è una proprietà di dipendenza di sola lettura, si usa il metodo `SetValue(DependencyPropertyKey, Object)` per impostarla, usando il `DependencyPropertyKey` che è accessibile solo all'interno della classe.

```

public Aquarium() : base()
{
    SetValue(AquariumContentsPropertyKey, new List<FrameworkElement>());
}

```

```

Public Sub New()
    MyBase.New()
    SetValue(AquariumContentsPropertyKey, New List(Of FrameworkElement)())
End Sub

```

A questo punto, se si esegue nuovamente questo stesso codice di test, è possibile ottenere i risultati previsti, in cui ciascun oggetto `Aquarium` supporta la propria raccolta univoca.

Se si sceglie una proprietà della raccolta con possibilità di accesso in lettura/scrittura, questo modello presenta una piccola variazione. In tal caso, è possibile chiamare la funzione di accesso set pubblica dal costruttore per eseguire l'inizializzazione, che chiamerebbe comunque la firma colonne di `SetValue(DependencyProperty, Object)` all'interno del wrapper del set, usando un identificatore di `DependencyProperty` pubblico.

Segnalazione di modifiche dei valori di binding nelle proprietà della raccolta

Una proprietà della raccolta che costituisce essa stessa una proprietà di dipendenza non segnala automaticamente le modifiche per le relative sottoproprietà. La creazione di binding all'interno di una raccolta può impedire al binding di segnalare le modifiche, invalidando in tal modo alcuni scenari di data binding. Tuttavia, se si usa il tipo di raccolta `FreezableCollection<T>` come tipo di raccolta, le modifiche alle sottoproprietà degli elementi contenuti nella raccolta vengono segnalate correttamente e il binding funziona come previsto.

Per abilitare l'associazione di sottoproprietà in una raccolta di oggetti di dipendenza, creare la proprietà della raccolta come tipo `FreezableCollection<T>`, con un vincolo di tipo per tale raccolta a qualsiasi `DependencyObject` classe derivata.

Vedere anche

- [FreezableCollection<T>](#)
- [Classi XAML e personalizzate per WPF](#)

- Panoramica sul data binding
- Panoramica sulle proprietà di dipendenza
- Proprietà di dipendenza personalizzate
- Metadati delle proprietà di dipendenza

Caricamento XAML e proprietà di dipendenza

04/11/2019 • 5 minutes to read • [Edit Online](#)

L'implementazione WPF corrente del relativo processore XAML dipende implicitamente dalle proprietà di dipendenza. Il processore WPF XAML usa metodi del sistema di proprietà per le proprietà di dipendenza al momento del caricamento del codice XAML binario e dell'elaborazione di attributi che sono proprietà di dipendenza. Ciò consente di ignorare in modo efficace i wrapper della proprietà. Quando si implementano proprietà di dipendenza personalizzate, è necessario tenere conto di questo comportamento ed evitare di inserire altro codice nel wrapper della proprietà, oltre ai metodi del sistema di proprietà [GetValue](#) e [SetValue](#).

Prerequisites

Questo argomento presuppone la conoscenza delle proprietà di dipendenza sia in qualità di consumer, sia in qualità di autore oltre alla lettura di [Cenni preliminari sulle proprietà di dipendenza](#) e [Proprietà Dependency personalizzate](#). È inoltre necessario aver letto [Cenni preliminari su XAML \(WPF\)](#) e [Descrizione dettagliata della sintassi XAML](#).

Implementazione del caricatore XAML WPF e prestazioni

Per motivi di implementazione, è meno costoso dal punto di vista del calcolo identificare una proprietà come proprietà di dipendenza e accedere al sistema di proprietà [SetValue](#) metodo per impostarlo, anziché usare il wrapper della proprietà e il relativo setter. Ciò si verifica in quanto un processore XAML deve dedurre l'intero modello a oggetti del codice di supporto esclusivamente in base alla conoscenza delle relazioni tra tipi e membri indicate dalla struttura del markup e da varie stringhe.

Il tipo viene ricercato tramite una combinazione di attributi xmlns e di assembly, ma l'identificazione dei membri, la determinazione di un supporto da impostare come attributo e la risoluzione dei tipi supportati dai valori delle proprietà potrebbero altrimenti richiedere un'ampia Reflection utilizzando [PropertyInfo](#). Poiché le proprietà di dipendenza di un determinato tipo sono accessibili come una tabella di archiviazione tramite il sistema di proprietà, l'implementazione WPF del relativo processore XAML usa questa tabella e deduce che qualsiasi proprietà ABC può essere impostata in modo più efficiente da la chiamata di [SetValue](#) nell'oggetto che contiene [DependencyObject](#) tipo derivato, usando l'identificatore della proprietà di dipendenza [ABCProperty](#).

Implicazioni per le proprietà di dipendenza personalizzate

Poiché l'implementazione corrente WPF del comportamento del processore XAML per l'impostazione delle proprietà ignora completamente i wrapper, non è opportuno inserire logica aggiuntiva nelle definizioni stabilita del wrapper per la proprietà di dipendenza personalizzata. Se si inserisce tale logica nella definizione stabilita, questa non sarà eseguita quando la proprietà viene impostata in XAML piuttosto che nel codice.

Analogamente, anche altri aspetti del processore XAML che ottengono i valori delle proprietà dall'elaborazione XAML utilizzano [GetValue](#) anziché utilizzare il wrapper. Pertanto, è consigliabile evitare qualsiasi implementazione aggiuntiva nella definizione del `get` oltre la chiamata [GetValue](#).

L'esempio seguente è una definizione di una proprietà di dipendenza consigliata con wrapper, in cui l'identificatore di proprietà è archiviato come campo `public static readonly` e le definizioni `get` e `set` non contengono codice oltre ai metodi del sistema di proprietà necessari che definiscono il supporto della proprietà di dipendenza.

```

public static readonly DependencyProperty AquariumGraphicProperty = DependencyProperty.Register(
    "AquariumGraphic",
    typeof(Uri),
    typeof(AquariumObject),
    new FrameworkPropertyMetadata(null,
        FrameworkPropertyMetadataOptions.AffectsRender,
        new PropertyChangedCallback(OnUriChanged)
    )
);
public Uri AquariumGraphic
{
    get { return (Uri)GetValue(AquariumGraphicProperty); }
    set { SetValue(AquariumGraphicProperty, value); }
}

```

```

Public Shared ReadOnly AquariumGraphicProperty As DependencyProperty =
DependencyProperty.Register("AquariumGraphic", GetType(Uri), GetType(AquariumObject), New
FrameworkPropertyMetadata(Nothing, FrameworkPropertyMetadataOptions.AffectsRender, New
PropertyChangedCallback(AddressOf OnUriChanged)))
Public Property AquariumGraphic() As Uri
    Get
        Return CType(GetValue(AquariumGraphicProperty), Uri)
    End Get
    Set(ByVal value As Uri)
        SetValue(AquariumGraphicProperty, value)
    End Set
End Property

```

Vedere anche

- [Panoramica sulle proprietà di dipendenza](#)
- [Cenni preliminari su XAML \(WPF\)](#)
- [Metadati delle proprietà di dipendenza](#)
- [Proprietà di dipendenza di tipo raccolta](#)
- [Sicurezza delle proprietà di dipendenza](#)
- [Modelli di costruttore sicuri per DependencyObject](#)

Procedure relative alle proprietà

23/10/2019 • 2 minutes to read • [Edit Online](#)

In questa sezione

[Implementare una proprietà di dipendenza](#)

[Aggiungere un tipo di proprietario per una proprietà di dipendenza](#)

[Registrare una proprietà associata](#)

[Eseguire l'override dei metadati per una proprietà di dipendenza](#)

Riferimenti

[DependencyProperty](#)

[PropertyMetadata](#)

[FrameworkPropertyMetadata](#)

[DependencyObject](#)

Sezioni correlate

[Proprietà](#)

Procedura: Implementare una proprietà di dipendenza

23/10/2019 • 3 minutes to read • [Edit Online](#)

In questo esempio viene illustrato come eseguire il backup di una proprietà Common Language Runtime (DependencyProperty CLR) con un campo, definendo in tal modo una proprietà di dipendenza. Quando si definiscono proprietà personalizzate e si vuole che supportino molti aspetti della funzionalità Windows Presentation Foundation (WPF), inclusi stili, data binding, ereditarietà, animazione e valori predefiniti, è necessario implementarle come proprietà di dipendenza.

Esempio

Nell'esempio seguente viene innanzitutto registrata una proprietà di dipendenza chiamando `Register` il metodo. Il nome del campo dell'identificatore utilizzato per archiviare il nome e le caratteristiche della proprietà di dipendenza deve essere l'oggetto `Name` scelto per la proprietà di dipendenza come parte `Register` della chiamata, accodato dalla stringa `Property` letterale. Ad esempio, se si registra una proprietà di dipendenza con `Name` un `Location` valore di, il campo identificatore definito per la proprietà di dipendenza deve essere denominato `LocationProperty`.

In questo esempio, il nome della proprietà di dipendenza e la relativa funzione di `State` accesso CLR è; il `StateProperty` campo identificatore è, il tipo della `Boolean` proprietà è e il tipo che regista la proprietà di `MyStateControl` dipendenza è.

Se non si riesce a seguire questo criterio di denominazione, la proprietà potrebbe non essere segnalata correttamente nelle finestre di progettazione e alcuni aspetti dell'applicazione di stili del sistema di proprietà potrebbero non funzionare come previsto.

È anche possibile specificare metadati predefiniti per una proprietà di dipendenza. In questo esempio viene registrato il valore predefinito della proprietà di dipendenza `State` in modo che sia `false`.

```
public class MyStateControl : ButtonBase
{
    public MyStateControl() : base() { }
    public Boolean State
    {
        get { return (Boolean)this.GetValue(StateProperty); }
        set { this.SetValue(StateProperty, value); }
    }
    public static readonly DependencyProperty StateProperty = DependencyProperty.Register(
        "State", typeof(Boolean), typeof(MyStateControl), new PropertyMetadata(false));
}
```

```
Public Class MyStateControl
    Inherits ButtonBase
    Public Sub New()
        MyBase.New()
    End Sub
    Public Property State() As Boolean
        Get
            Return CType(Me.GetValue(StateProperty), Boolean)
        End Get
        Set(ByVal value As Boolean)
            Me.SetValue(StateProperty, value)
        End Set
    End Property
    Public Shared ReadOnly StateProperty As DependencyProperty = DependencyProperty.Register("State",
        GetType(Boolean), GetType(MyStateControl), New PropertyMetadata(False))
End Class
```

Per ulteriori informazioni su come e perché implementare una proprietà di dipendenza, anziché semplicemente eseguire il backup di una proprietà CLR con un campo privato, vedere [Cenni preliminari sulle proprietà di dipendenza](#).

Vedere anche

- [Panoramica sulle proprietà di dipendenza](#)
- [Procedure relative alle proprietà](#)

Procedura: Aggiungere un tipo di proprietario per una proprietà di dipendenza

23/10/2019 • 3 minutes to read • [Edit Online](#)

In questo esempio viene illustrato come aggiungere una classe come proprietario di una proprietà di dipendenza registrata per un tipo diverso. In questo modo, il WPF Reader e il XAML sistema di proprietà sono in grado di riconoscere la classe come proprietario aggiuntivo della proprietà. Se si aggiunge come proprietario, facoltativamente, la classe aggiuntiva consente di fornire metadati specifici del tipo.

Nell'esempio `StateProperty` seguente è una proprietà registrata `MyStateControl` dalla classe. La classe `UnrelatedStateControl` si aggiunge come proprietario dell'oggetto `StateProperty` utilizzando il `AddOwner` metodo, specificando in particolare la firma che consente la presenza di nuovi metadati per la proprietà di dipendenza esistente nel tipo di aggiunta. Si noti che è necessario fornire le funzioni di accesso Common Language Runtime (CLR) per la proprietà in modo simile all'esempio illustrato nell'esempio [implementare una proprietà di dipendenza](#), nonché esporre nuovamente l'identificatore della proprietà di dipendenza nella classe aggiunta come proprietario.

Senza wrapper, la proprietà di dipendenza continuerà a funzionare dalla prospettiva di accesso a livello `GetValue` di `SetValue` codice tramite o. Tuttavia, in genere si desidera eseguire in parallelo questo comportamento del sistema di proprietà con i wrapper della proprietà CLR. I wrapper facilitano l'impostazione della proprietà di dipendenza a livello di codice e consentono di impostare le proprietà come XAML attributi.

Per informazioni su come eseguire l'override dei metadati predefiniti, vedere [eseguire l'override dei metadati per una proprietà di dipendenza](#).

Esempio

```
public class MyStateControl : ButtonBase
{
    public MyStateControl() : base() { }
    public Boolean State
    {
        get { return (Boolean)this.GetValue(StateProperty); }
        set { this.SetValue(StateProperty, value); }
    }
    public static readonly DependencyProperty StateProperty = DependencyProperty.Register(
        "State", typeof(Boolean), typeof(MyStateControl), new PropertyMetadata(false));
}
```

```

Public Class MyStateControl
    Inherits ButtonBase
    Public Sub New()
        MyBase.New()
    End Sub
    Public Property State() As Boolean
        Get
            Return CType(Me.GetValue(StateProperty), Boolean)
        End Get
        Set(ByVal value As Boolean)
            Me.SetValue(StateProperty, value)
        End Set
    End Property
    Public Shared ReadOnly StateProperty As DependencyProperty = DependencyProperty.Register("State",
        GetType(Boolean), GetType(MyStateControl), New PropertyMetadata(False))
End Class

```

```

public class UnrelatedStateControl : Control
{
    public UnrelatedStateControl() { }
    public static readonly DependencyProperty StateProperty =
MyStateControl.StateProperty.AddOwner(typeof(UnrelatedStateControl), new PropertyMetadata(true));
    public Boolean State
    {
        get { return (Boolean)this.GetValue(StateProperty); }
        set { this.SetValue(StateProperty, value); }
    }
}

```

```

Public Class UnrelatedStateControl
    Inherits Control
    Public Sub New()
    End Sub
    Public Shared ReadOnly StateProperty As DependencyProperty =
MyStateControl.StateProperty.AddOwner(GetType(UnrelatedStateControl), New PropertyMetadata(True))
    Public Property State() As Boolean
        Get
            Return CType(Me.GetValue(StateProperty), Boolean)
        End Get
        Set(ByVal value As Boolean)
            Me.SetValue(StateProperty, value)
        End Set
    End Property
End Class

```

Vedere anche

- [Proprietà di dipendenza personalizzate](#)
- [Panoramica sulle proprietà di dipendenza](#)

Procedura: Registrare una proprietà associata

23/10/2019 • 2 minutes to read • [Edit Online](#)

Questo esempio mostra come registrare una proprietà associata e fornire funzioni di accesso pubbliche, in modo da poter usare la proprietà sia in Extensible Application Markup Language (XAML) che nel codice. Le proprietà associate rappresentano un concetto della sintassi definito da Extensible Application Markup Language (XAML). La maggior parte delle proprietà associate per i tipi WPF viene anche implementata come proprietà di dipendenza. È possibile usare le proprietà di dipendenza in qualsiasi [DependencyObject](#) tipi.

Esempio

Nell'esempio seguente viene illustrato come registrare una proprietà associata come proprietà di dipendenza, usando il [RegisterAttached](#) (metodo). La classe del provider può fornire metadati predefiniti per la proprietà applicabile quando la proprietà viene usata in un'altra classe, a meno che tale classe non esegua l'override dei metadati. In questo esempio il valore predefinito della proprietà `IsBubbleSource` viene impostato su `false`.

La classe del provider per una proprietà associata (anche se non è registrata come proprietà di dipendenza) deve fornire funzioni di accesso `get` e `set` statiche basate sulla convenzione di denominazione `Set [NomeProprietàAssociata]` e `Get [NomeProprietàAssociata]`. Queste funzioni di accesso sono necessarie per consentire al lettore XAML in funzione di riconoscere la proprietà come attributo in XAML e risolvere i tipi appropriati.

```
public static readonly DependencyProperty IsBubbleSourceProperty = DependencyProperty.RegisterAttached(
    "IsBubbleSource",
    typeof(Boolean),
    typeof(AquariumObject),
    new FrameworkPropertyMetadata(false, FrameworkPropertyMetadataOptions.AffectsRender)
);
public static void SetIsBubbleSource(UIElement element, Boolean value)
{
    element.SetValue(IsBubbleSourceProperty, value);
}
public static Boolean GetIsBubbleSource(UIElement element)
{
    return (Boolean)element.GetValue(IsBubbleSourceProperty);
}
```

```
Public Shared ReadOnly IsBubbleSourceProperty As DependencyProperty =
DependencyProperty.RegisterAttached("IsBubbleSource", GetType(Boolean), GetType(AquariumObject), New
FrameworkPropertyMetadata(False, FrameworkPropertyMetadataOptions.AffectsRender))
Public Shared Sub SetIsBubbleSource(ByVal element As UIElement, ByVal value As Boolean)
    element.SetValue(IsBubbleSourceProperty, value)
End Sub
Public Shared Function GetIsBubbleSource(ByVal element As UIElement) As Boolean
    Return CType(element.GetValue(IsBubbleSourceProperty), Boolean)
End Function
```

Vedere anche

- [DependencyProperty](#)
- [Panoramica sulle proprietà di dipendenza](#)
- [Proprietà di dipendenza personalizzate](#)

- Procedure relative alle proprietà

Procedura: Eseguire l'override dei metadati per una proprietà di dipendenza

23/10/2019 • 3 minutes to read • [Edit Online](#)

Questo esempio illustra come eseguire l'override di metadati di proprietà di dipendenza predefinito che derivano da una classe ereditata, chiamando il [OverrideMetadata](#) (metodo) e fornendo i metadati specifici del tipo.

Esempio

Con la definizione relativa [PropertyMetadata](#), una classe possa definire i comportamenti della proprietà di dipendenza, ad esempio il callback del sistema predefinita valore e proprietà. Molte classi della proprietà di dipendenza dispongono già di metadati predefiniti stabiliti nell'ambito del processo di registrazione. Ciò include le proprietà di dipendenza che fanno parte di WPF API. Una classe che eredita la proprietà di dipendenza tramite l'ereditarietà di classe può eseguire l'override dei metadati originali in modo che le caratteristiche della proprietà che è possibile modificare tramite i metadati soddisfino qualsiasi requisito specifico della sottoclassificazione.

L'override dei metadati di una proprietà di dipendenza deve essere eseguito prima che la proprietà venga resa utilizzabile dal sistema di proprietà, vale a dire nel momento in cui vengono create istanze specifiche degli oggetti che registrano la proprietà. Le chiamate a [OverrideMetadata](#) devono essere eseguite all'interno dei costruttori statici del tipo che fornisce se stesso come il `forType` parametro di [OverrideMetadata](#). Il tentativo di modificare i metadati dopo la creazione delle istanze del tipo di proprietario non genererà eccezioni, ma darà come risultato comportamenti incoerenti nel sistema di proprietà. Inoltre, l'override dei metadati può essere eseguito solo una volta per tipo. I tentativi successivi di eseguire l'override dei metadati sullo stesso tipo genereranno un'eccezione.

Nell'esempio seguente la classe personalizzata `MyAdvancedStateControl` esegue l'override dei metadati forniti per `StateProperty` da `MyAdvancedStateControl` con i nuovi metadati delle proprietà. Ad esempio, il valore predefinito di `StateProperty` sarà `true` quando viene eseguita una query sulla proprietà in un'istanza `MyAdvancedStateControl` appena creata.

```
public class MyStateControl : ButtonBase
{
    public MyStateControl() : base() { }

    public Boolean State
    {
        get { return (Boolean)this.GetValue(StateProperty); }
        set { this.SetValue(StateProperty, value); }
    }

    public static readonly DependencyProperty StateProperty = DependencyProperty.Register(
        "State", typeof(Boolean), typeof(MyStateControl), new PropertyMetadata(false));
}
```

```

Public Class MyStateControl
    Inherits ButtonBase
    Public Sub New()
        MyBase.New()
    End Sub
    Public Property State() As Boolean
        Get
            Return CType(Me.GetValue(StateProperty), Boolean)
        End Get
        Set(ByVal value As Boolean)
            Me.SetValue(StateProperty, value)
        End Set
    End Property
    Public Shared ReadOnly StateProperty As DependencyProperty = DependencyProperty.Register("State",
        GetType(Boolean), GetType(MyStateControl), New PropertyMetadata(False))
End Class

```

```

public class MyAdvancedStateControl : MyStateControl
{
    public MyAdvancedStateControl() : base() { }
    static MyAdvancedStateControl()
    {
        MyStateControl.StateProperty.OverrideMetadata(typeof(MyAdvancedStateControl), new
        PropertyMetadata(true));
    }
}

```

```

Public Class MyAdvancedStateControl
    Inherits MyStateControl
    Public Sub New()
        MyBase.New()
    End Sub
    Shared Sub New()
        MyStateControl.StateProperty.OverrideMetadata(GetType(MyAdvancedStateControl), New
        PropertyMetadata(True))
    End Sub
End Class

```

Vedere anche

- [DependencyProperty](#)
- [Panoramica sulle proprietà di dipendenza](#)
- [Proprietà di dipendenza personalizzate](#)
- [Procedure relative alle proprietà](#)

Eventi (WPF)

03/02/2020 • 2 minutes to read • [Edit Online](#)

In Windows Presentation Foundation (WPF) vengono introdotti gli eventi indirizzati che possono richiamare i gestori esistenti su vari listener nell'albero degli elementi di un'applicazione.

Contenuto della sezione

[Cenni preliminari sugli eventi indirizzati](#)
[Cenni preliminari sugli eventi associati](#)
[Eventi di durata degli oggetti](#)
[Contrassegno degli eventi indirizzati come gestiti e gestione delle classi](#)
[Eventi di anteprima](#)
[Eventi di modifica delle proprietà](#)
[Visual Basic e la gestione degli eventi WPF](#)
[Modelli di eventi deboli](#)
[Procedure relative alla struttura ad albero e alla serializzazione degli elementi](#)

Riferimento

[RoutedEventArgs](#)
[EventManager](#)
[RoutingStrategy](#)

Sezioni correlate

[Architettura WPF](#)
[XAML in WPF](#)
[Elementi di base](#)
[Albero degli elementi e serializzazione](#)
[Proprietà](#)
[Input](#)
[Risorse](#)
[Applicazione di stili e modelli](#)
[Modello di contenuto WPF](#)
[Modello di threading](#)

Cenni preliminari sugli eventi indirizzati

31/01/2020 • 45 minutes to read • [Edit Online](#)

Questo argomento descrive il concetto di eventi indirizzati in Windows Presentation Foundation (WPF). L'argomento definisce la terminologia correlata agli eventi indirizzati, descrive in che modo questi eventi sono indirizzati lungo un albero di elementi, riepiloga le modalità di gestione degli eventi indirizzati e spiega come creare eventi indirizzati personalizzati.

Prerequisiti

In questo argomento si presuppone che l'utente abbia una conoscenza di base del Common Language Runtime (CLR) e della programmazione orientata a oggetti, nonché il concetto di come le relazioni tra WPF elementi possano essere concettuali come albero. Per seguire gli esempi illustrati in questo argomento, è anche necessario conoscere Extensible Application Markup Language (XAML) e saper scrivere applicazioni o pagine WPF di base. Per ulteriori informazioni, vedere [procedura dettagliata: applicazione desktop WPF](#) e [Cenni preliminari su XAML \(WPF\)](#).

Definizione di evento indirizzato

È possibile considerare gli eventi indirizzati da un punto di vista funzionale o da una prospettiva di implementazione. In questo argomento sono illustrati entrambi i concetti, perché alcuni utenti trovano più utile il primo e altri il secondo.

Definizione funzionale: un evento indirizzato è un tipo di evento che può richiamare gestori su più listener in un albero degli elementi invece che solo sull'oggetto che ha generato l'evento.

Definizione di implementazione: un evento indirizzato è un evento CLR supportato da un'istanza della classe [RoutedEventArgs](#) e viene elaborato dal sistema di Windows Presentation Foundation (WPF) eventi.

Un'applicazione WPF tipica contiene molti elementi. Indipendentemente dal fatto che vengano creati nel codice o dichiarati in XAML, questi elementi sono legati tra loro da una relazione di albero degli elementi. La route dell'evento può procedere in una di due direzioni, in base alla definizione dell'evento, ma generalmente parte dall'elemento di origine e sale (bubbling) lungo l'albero degli elementi fino a raggiungere la radice dell'albero (in genere una pagina o una finestra). Questo concetto di bubbling può risultare familiare a chi in precedenza ha già usato il modello a oggetti DHTML.

Considerare il semplice albero degli elementi seguente:

```
<Border Height="50" Width="300" BorderBrush="Gray" BorderThickness="1">
  <StackPanel Background="LightGray" Orientation="Horizontal" Button.Click="CommonClickHandler">
    <Button Name="YesButton" Width="Auto" >Yes</Button>
    <Button Name="NoButton" Width="Auto" >No</Button>
    <Button Name="CancelButton" Width="Auto" >Cancel</Button>
  </StackPanel>
</Border>
```

Questo albero degli elementi produce un risultato analogo al seguente:



In questo albero degli elementi semplificato, l'origine di un evento [Click](#) è uno degli elementi [Button](#) e

qualsiasi [Button](#) fatto clic sia il primo elemento che ha l'opportunità di gestire l'evento. Tuttavia, se nessun gestore associato al [Button](#) agisce sull'evento, l'evento verrà ribolle verso l'alto nel [Button](#) padre nell'albero degli elementi, che è il [StackPanel](#). Potenzialmente, l'evento bolle per [Border](#) quindi oltre alla radice della pagina dell'albero degli elementi (non illustrato).

In altre parole, la route dell'evento per questo evento [Click](#) è:

Button-->StackPanel-->Border-->...

Scenari principali per gli eventi indirizzati

Di seguito è riportato un breve riepilogo degli scenari in cui è stato motivato il concetto di evento indirizzato e il motivo per cui un tipico evento CLR non era adatto a questi scenari:

Composizione e incapsulamento di controlli: diversi controlli di WPF hanno un modello di contenuto avanzato. Ad esempio, è possibile inserire un'immagine all'interno di un [Button](#), che estende efficacemente la struttura ad albero visuale del pulsante. Tuttavia, l'immagine aggiunta non deve interrompere il comportamento di hit testing che fa in modo che un pulsante risponda a una [Click](#) del contenuto, anche se l'utente fa clic su pixel che fanno tecnicamente parte dell'immagine.

Punti di collegamento del gestore singolari: In Windows Forms, è necessario allegare più volte lo stesso gestore per elaborare gli eventi che possono essere generati da più elementi. Gli eventi indirizzati consentono di associare il gestore una sola volta, come illustrato nell'esempio precedente, e di usare la logica del gestore per determinare da dove proviene l'evento, se necessario. Ad esempio, questo potrebbe essere il gestore per il codice XAML illustrato in precedenza:

```
private void CommonClickHandler(object sender, RoutedEventArgs e)
{
    FrameworkElement feSource = e.Source as FrameworkElement;
    switch (feSource.Name)
    {
        case "YesButton":
            // do something here ...
            break;
        case "NoButton":
            // do something ...
            break;
        case "CancelButton":
            // do something ...
            break;
    }
    e.Handled=true;
}
```

```
Private Sub CommonClickHandler(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Dim feSource As FrameworkElement = TryCast(e.Source, FrameworkElement)
    Select Case feSource.Name
        Case "YesButton"
            ' do something here ...
        Case "NoButton"
            ' do something ...
        Case "CancelButton"
            ' do something ...
    End Select
    e.Handled=True
End Sub
```

Gestione delle classi: gli eventi indirizzati consentono un gestore statico definito dalla classe. Questo gestore di classi ha la possibilità di gestire un evento prima di qualsiasi gestore di istanze associato.

Riferimento a un evento senza reflection: alcune tecniche di codice e di markup richiedono un modo

per identificare un evento specifico. Un evento indirizzato crea un campo **RoutedEvent** come identificatore, che fornisce una tecnica di identificazione degli eventi affidabile che non richiede la reflection statica o in fase di esecuzione.

Modalità di implementazione degli eventi indirizzati

Un evento indirizzato è un evento CLR supportato da un'istanza della classe **RoutedEvent** e registrato con il sistema di eventi di WPF. L'istanza **RoutedEvent** ottenuta dalla registrazione viene in genere mantenuta come `public static readonly` membro del campo della classe che esegue la registrazione e quindi "è proprietario" dell'evento indirizzato. La connessione all'evento CLR con nome identico (a volte definito evento "wrapper") viene eseguita eseguendo l'override delle implementazioni `add` e `remove` per l'evento CLR. Normalmente, `add` e `remove` vengono lasciati come impostazione predefinita implicita che usa la sintassi di evento specifica del linguaggio per aggiungere e rimuovere i gestori dell'evento. Il meccanismo di backup e di connessione degli eventi indirizzati è concettualmente simile al modo in cui una proprietà di dipendenza è una proprietà CLR supportata dalla classe **DependencyProperty** e registrata con il sistema di proprietà WPF.

Nell'esempio seguente viene illustrata la dichiarazione per un evento indirizzato `Tap` personalizzato, inclusa la registrazione e l'esposizione del campo identificatore **RoutedEvent** e le implementazioni di `add` e `remove` per l'evento CLR `Tap`.

```
public static readonly RoutedEvent TapEvent = EventManager.RegisterRoutedEvent(
    "Tap", RoutingStrategy.Bubble, typeof(RoutedEventHandler), typeof(MyButtonSimple));

// Provide CLR accessors for the event
public event RoutedEventHandler Tap
{
    add { AddHandler(TapEvent, value); }
    remove { RemoveHandler(TapEvent, value); }
}
```

```
Public Shared ReadOnly TapEvent As RoutedEvent = EventManager.RegisterRoutedEvent("Tap",
    RoutingStrategy.Bubble, GetType(RoutedEventHandler), GetType(MyButtonSimple))

    ' Provide CLR accessors for the event
    Public Custom Event Tap As RoutedEventHandler
        AddHandler(ByVal value As RoutedEventHandler)
            Me.AddHandler(TapEvent, value)
        End AddHandler

        RemoveHandler(ByVal value As RoutedEventHandler)
            Me.RemoveHandler(TapEvent, value)
        End RemoveHandler

        RaiseEvent(ByVal sender As Object, ByVal e As RoutedEventArgs)
            Me.RaiseEvent(e)
        End RaiseEvent
    End Event
```

Gestori di eventi indirizzati e XAML

Per aggiungere un gestore per un evento usando XAML, il nome dell'evento deve essere dichiarato come attributo dell'elemento che è un listener di eventi. Il valore dell'attributo è il nome del metodo del gestore implementato, che deve trovarsi nella classe parziale del file code-behind.

```
<Button Click="b1SetColor">button</Button>
```

La sintassi XAML per l'aggiunta di gestori eventi CLR standard è la stessa per l'aggiunta di gestori di eventi

indirizzati, perché si aggiungono effettivamente gestori al wrapper di eventi CLR, che include un'implementazione di evento indirizzato sottostante. Per altre informazioni sull'aggiunta di gestori eventi in XAML, vedere [Cenni preliminari su XAML \(WPF\)](#).

Strategie di routing

Gli eventi indirizzati usano una delle tre strategie di routing illustrate di seguito:

- **Bubbling:** vengono richiamati i gestori eventi sull'origine dell'evento. L'evento indirizzato viene quindi indirizzato agli elementi padre successivi fino a raggiungere la radice dell'albero degli elementi. La maggior parte degli eventi indirizzati usa la strategia di bubbling. Gli eventi indirizzati di bubbling vengono in genere usati per segnalare l'input o le modifiche dello stato da controlli distinti o altri elementi dell'interfaccia utente.
- **Diretto:** solo l'elemento di origine può richiamare i gestori in risposta. Questo è analogo al "routing" usato Windows Forms per gli eventi. Tuttavia, a differenza di un evento CLR standard, gli eventi indirizzati diretti supportano la gestione delle classi (la gestione delle classi viene illustrata in una sezione imminente) e può essere usata da [EventSetter](#) e [EventTrigger](#).
- **Tunneling:** inizialmente vengono richiamati i gestori eventi alla radice dell'albero degli elementi. L'evento indirizzato percorre quindi una route attraverso elementi figlio successivi, verso l'elemento nodo che rappresenta l'origine dell'evento indirizzato (l'elemento che ha generato l'evento indirizzato). Gli eventi indirizzati di tunneling vengono spesso usati o gestiti come parte della composizione di un controllo, in modo che gli eventi di parti composite possano essere deliberatamente eliminati o sostituiti da eventi specifici del controllo completo. Gli eventi di input forniti in WPF vengono spesso implementati come coppia di tunneling/bubbling. Gli eventi di tunneling sono talvolta anche detti eventi di anteprima, per una convenzione di denominazione usata per le coppie.

Vantaggi offerti dall'uso degli eventi indirizzati

Per gli sviluppatori di applicazioni non è sempre necessario o importante sapere se l'evento gestito è implementato come evento indirizzato. Gli eventi indirizzati hanno un comportamento speciale, che però è per lo più invisibile se l'evento viene gestito sull'elemento in cui è stato generato.

Gli eventi indirizzati diventano particolarmente efficaci se si usa uno degli scenari suggeriti: definizione di gestori comuni in una radice comune, composizione di un controllo personalizzato o definizione di una classe di controlli personalizzata.

Non è necessario che i listener e le origini degli eventi indirizzati condividano un evento comune nella gerarchia. Qualsiasi [UIElement](#) o [ContentElement](#) può essere un listener di eventi per qualsiasi evento indirizzato. È quindi possibile usare il set completo di eventi indirizzati disponibili nell'API di lavoro come "interfaccia" concettuale, in cui gli elementi diversi dell'applicazione possono scambiare informazioni sugli eventi. Questo concetto di "interfaccia" per gli eventi indirizzati è applicabile in particolar modo agli eventi di input.

Gli eventi indirizzati possono anche essere usati per comunicare nell'albero degli elementi, poiché i dati di un evento vengono trasmessi a ogni elemento nella route. Se un elemento modifica in qualche modo i dati dell'evento, tale modifica risulta disponibile per l'elemento successivo nella route.

Oltre all'aspetto di routing, è possibile che qualsiasi evento WPF venga implementato come un evento indirizzato anziché come evento CLR standard. Se si implementano eventi personalizzati, considerare anche questi principi:

- Alcune funzionalità di applicazione di stili e modelli di WPF come [EventSetter](#) e [EventTrigger](#) richiedono che l'evento a cui si fa riferimento sia un evento indirizzato. Si tratta dello scenario di

identificatore dell'evento citato in precedenza.

- Gli eventi indirizzati supportano un meccanismo di gestione delle classi in base al quale la classe può specificare metodi statici che hanno la possibilità di gestire gli eventi indirizzati prima che qualsiasi gestore di istanze registrato possa accedervi. Questa funzionalità è molto utile nella progettazione di controlli, perché la classe può imporre comportamenti di classe basati su eventi che non possono essere eliminati accidentalmente mediante la gestione di un evento su un'istanza.

Ognuna delle considerazioni precedenti viene discussa in una sezione separata di questo argomento.

Aggiunta e implementazione di un gestore eventi per un evento indirizzato

Per aggiungere un gestore eventi in XAML, è sufficiente aggiungere il nome dell'evento a un elemento come attributo e impostare il valore dell'attributo come nome del gestore eventi che implementa un delegato appropriato, come nell'esempio seguente.

```
<Button Click="b1SetColor">button</Button>
```

`b1SetColor` è il nome del gestore implementato che contiene il codice che gestisce l'evento di [Click](#). `b1SetColor` deve avere la stessa firma del delegato [RoutedEventHandler](#), che è il delegato del gestore eventi per l'evento [Click](#). Il primo parametro di tutti i delegati dei gestori di eventi indirizzati specifica l'elemento a cui viene aggiunto il gestore eventi, mentre il secondo parametro specifica i dati per l'evento.

```
void b1SetColor(object sender, RoutedEventArgs args)
{
    //logic to handle the Click event
}
```

```
Private Sub b1SetColor(ByVal sender As Object, ByVal args As RoutedEventArgs)
    'logic to handle the Click event
End Sub
```

[RoutedEventHandler](#) è il delegato del gestore eventi indirizzati di base. Per gli eventi indirizzati specializzati per determinati controlli o scenari, i delegati da usare per i gestori di eventi indirizzati possono anche essere più specializzati, in modo da poter trasmettere dati di eventi specializzati. In uno scenario di input comune, ad esempio, è possibile gestire un evento indirizzato [DragEnter](#). Il gestore deve implementare il delegato [DragEventHandler](#). Usando il delegato più specifico, è possibile elaborare il [DragEventArgs](#) nel gestore e leggere la proprietà [Data](#), che contiene il payload degli Appunti dell'operazione di trascinamento.

Per un esempio completo di come aggiungere un gestore eventi a un elemento usando XAML, vedere [Gestire un evento indirizzato](#).

L'aggiunta di un gestore per un evento indirizzato in un'applicazione creata nel codice è semplice. I gestori di eventi indirizzati possono sempre essere aggiunti tramite un metodo helper [AddHandler](#) (ovvero lo stesso metodo che il supporto esistente chiama per `add`). Tuttavia, gli eventi indirizzati WPF esistenti in genere dispongono di implementazioni di supporto di `add` e logica di `remove` che consentono di aggiungere i gestori per gli eventi indirizzati tramite una sintassi di evento specifica del linguaggio, che è una sintassi più intuitiva rispetto al metodo helper. Di seguito è illustrato un esempio di utilizzo del metodo helper:

```

void MakeButton()
{
    Button b2 = new Button();
    b2.AddHandler(Button.ClickEvent, new RoutedEventHandler(Onb2Click));
}
void Onb2Click(object sender, RoutedEventArgs e)
{
    //logic to handle the Click event
}

```

```

Private Sub MakeButton()
    Dim b2 As New Button()
    b2.AddHandler(Button.ClickEvent, New RoutedEventHandler(AddressOf Onb2Click))
End Sub
Private Sub Onb2Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
    'logic to handle the Click event
End Sub

```

Nell'esempio seguente viene illustrata la C# sintassi dell'operatore (Visual Basic presenta una sintassi di operatore leggermente diversa a causa della relativa gestione della dereferenziazione):

```

void MakeButton2()
{
    Button b2 = new Button();
    b2.Click += new RoutedEventHandler(Onb2Click2);
}
void Onb2Click2(object sender, RoutedEventArgs e)
{
    //logic to handle the Click event
}

```

```

Private Sub MakeButton2()
    Dim b2 As New Button()
    AddHandler b2.Click, AddressOf Onb2Click2
End Sub
Private Sub Onb2Click2(ByVal sender As Object, ByVal e As RoutedEventArgs)
    'logic to handle the Click event
End Sub

```

Per un esempio di come aggiungere un gestore eventi nel codice, vedere [Aggiungere un gestore eventi mediante codice](#).

Se si utilizza Visual Basic, è inoltre possibile utilizzare la parola chiave `Handles` per aggiungere gestori come parte delle dichiarazioni del gestore. Per altre informazioni, vedere [Visual Basic e la gestione degli eventi WPF](#).

Concetto di gestito

Tutti gli eventi indirizzati condividono una classe di base dei dati di evento comuni, `RoutedEventArgs`. `RoutedEventArgs` definisce la proprietà `Handled`, che accetta un valore booleano. Lo scopo della proprietà `Handled` consiste nell'abilitare qualsiasi gestore eventi lungo la route per contrassegnare l'evento indirizzato come *gestito*, impostando il valore di `Handled` su `true`. Dopo essere stati elaborati dal gestore in corrispondenza di un elemento lungo la route, i dati di evento condivisi vengono nuovamente segnalati a ogni listener lungo la route.

Il valore di `Handled` influiscono sul modo in cui un evento indirizzato viene segnalato o elaborato mentre si sposta ulteriormente lungo la route. Se `Handled` viene `true` nei dati dell'evento per un evento indirizzato, i gestori che restano in attesa di tale evento indirizzato su altri elementi in genere non vengono più

richiamati per quella particolare istanza di evento. Ciò vale sia per i gestori associati in XAML che per i gestori aggiunti da sintassi di associazione dei gestori eventi specifiche del linguaggio, come `+ =` o `Handles`. Per gli scenari di gestione più comuni, contrassegnare un evento come gestito impostando `Handled` su `true` arresterà il routing per una route di tunneling o una route di bubbling e anche per qualsiasi evento gestito in un punto della route da un gestore di classe.

Tuttavia, esiste un meccanismo "handledEventsToo" che consente ai listener di eseguire ancora i gestori in risposta a eventi indirizzati in cui `Handled` viene `true` nei dati dell'evento. In altre parole, la route dell'evento non viene effettivamente interrotta contrassegnando i dati di evento come gestiti. È possibile usare il meccanismo `handledEventsToo` solo nel codice o in un `EventSetter`:

- Nel codice, anziché usare una sintassi di evento specifica del linguaggio che funzioni per gli eventi CLR generali, chiamare il metodo WPF `AddHandler(RoutedEvent, Delegate, Boolean)` per aggiungere il gestore. Specificare il valore di `handledEventsToo` come `true`.
- In una `EventSetter` impostare l'attributo `HandledEventsToo` da `true`.

Oltre al comportamento che `Handled` stato produce negli eventi indirizzati, il concetto di `Handled` presenta implicazioni relative alla progettazione dell'applicazione e alla scrittura del codice del gestore eventi. È possibile concettualizzare `Handled` come un semplice protocollo esposto dagli eventi indirizzati. Il modo in cui si usa questo protocollo dipende dall'utente, ma la progettazione concettuale per la modalità di utilizzo del valore di `Handled` è la seguente:

- Se un evento indirizzato è contrassegnato come gestito, non è necessario che venga gestito nuovamente da altri elementi lungo la route.
- Se un evento indirizzato non è contrassegnato come gestito, altri listener precedenti lungo la Route hanno scelto di non registrare un gestore o i gestori registrati hanno scelto di non modificare i dati dell'evento e impostare `Handled` su `true`. In alternativa, è ovviamente possibile che il listener corrente sia il primo punto della route. I gestori del listener corrente hanno ora tre possibili corsi d'azione:
 - Non eseguire alcuna azione. L'evento rimane non gestito e viene indirizzato al listener successivo.
 - Eseguire il codice in risposta all'evento, ma stabilire che l'azione eseguita non è sufficiente per contrassegnare l'evento come gestito. L'evento viene indirizzato al listener successivo.
 - Eseguire il codice in risposta all'evento. Contrassegnare l'evento come gestito nei dati di evento passati al gestore, perché l'azione eseguita viene ritenuta sufficiente per contrassegnare l'evento come gestito. L'evento viene ancora indirizzato al listener successivo, ma con `Handled = true` nei dati degli eventi, quindi solo i listener `handledEventsToo` hanno la possibilità di richiamare altri gestori.

Questa progettazione concettuale è rafforzata dal comportamento del routing indicato in precedenza: è più difficile (anche se ancora possibile nel codice o negli stili) allineare i gestori per gli eventi indirizzati richiamati anche se un gestore precedente lungo la route ha già impostato `Handled` `true`.

Per ulteriori informazioni su `Handled`, sulla gestione delle classi degli eventi indirizzati e sui suggerimenti relativi al momento in cui è opportuno contrassegnare un evento indirizzato come `Handled`, vedere [contrassegno degli eventi indirizzati come gestiti e gestione delle classi](#).

Nelle applicazioni, è piuttosto comune gestire un evento indirizzato di bubbling solo sull'oggetto che lo ha generato, senza preoccuparsi delle caratteristiche di routing dell'evento. È tuttavia comunque consigliabile contrassegnare l'evento indirizzato come gestito nei dati di evento, per evitare effetti collaterali imprevisti nel caso in cui un elemento più in alto nell'albero degli elementi abbia un gestore associato per lo stesso evento indirizzato.

Gestori di classi

Se si definisce una classe che deriva in qualche modo da [DependencyObject](#), è anche possibile definire e allineare un gestore di classi per un evento indirizzato che è un membro di evento dichiarato o ereditato della classe. I gestori di classi vengono richiamati prima dei gestori di listener di istanze associati a un'istanza di tale classe, ogni volta che un evento indirizzato raggiunge un'istanza dell'elemento nella relativa route.

Alcuni controlli WPF hanno una gestione intrinseca delle classi per determinati eventi indirizzati. In questo modo potrebbe sembrare che l'evento indirizzato non venga mai generato, ma in realtà viene gestito tramite classi e, usando determinate tecniche, può essere ancora gestito dai gestori di istanze. Molti controlli e classi di base, inoltre, espongono metodi virtuali che possono essere usati per eseguire l'override del comportamento di gestione delle classi. Per altre informazioni su come evitare un comportamento di gestione delle classi non desiderato e su come definire la gestione delle classi in una classe personalizzata, vedere [Contrassegno degli eventi indirizzati come gestiti e gestione delle classi](#).

Eventi associati in WPF

Il linguaggio XAML definisce anche un tipo di evento speciale noto come *evento associato*. Un evento associato consente di aggiungere un gestore per un determinato evento a un elemento arbitrario.

L'elemento che gestisce l'evento non deve necessariamente definire o ereditare l'evento associato, né l'oggetto che genera potenzialmente l'evento o l'istanza di gestione di destinazione deve necessariamente definire o "possedere" in altro modo tale evento come membro della classe.

Il sistema di input di WPF fa largo uso degli eventi associati. Tuttavia, quasi tutti questi eventi associati vengono inoltrati tramite elementi di base. Gli eventi di input appaiono quindi come eventi indirizzati non associati equivalenti, membri della classe di elementi di base. Ad esempio, l'evento associato sottostante [Mouse.MouseDown](#) può essere gestito più facilmente in qualsiasi [UIElement](#) specificato utilizzando [MouseDown](#) su tale [UIElement](#) anziché occuparsi della sintassi dell'evento associato in XAML o nel codice.

Per altre informazioni sugli eventi associati in WPF, vedere [Cenni preliminari sugli eventi associati](#).

Nomi di evento completi in XAML

Un altro utilizzo della sintassi simile alla sintassi dell'evento associato *nometipo.nomeevento*, che però non è esattamente un utilizzo di un evento associato, è rappresentato dall'associazione di gestori per eventi indirizzati generati da elementi figlio. I gestori vengono associati a un elemento padre comune, per sfruttare il routing degli eventi, anche se l'evento indirizzato rilevante potrebbe non essere membro dell'elemento padre comune. Si consideri di nuovo questo esempio:

```
<Border Height="50" Width="300" BorderBrush="Gray" BorderThickness="1">
    <StackPanel Background="LightGray" Orientation="Horizontal" Button.Click="CommonClickHandler">
        <Button Name="YesButton" Width="Auto" >Yes</Button>
        <Button Name="NoButton" Width="Auto" >No</Button>
        <Button Name="CancelButton" Width="Auto" >Cancel</Button>
    </StackPanel>
</Border>
```

In questo caso, il listener dell'elemento padre in cui viene aggiunto il gestore è un [StackPanel](#). Tuttavia, viene aggiunto un gestore per un evento indirizzato che è stato dichiarato e che verrà generato dalla classe [Button](#) ([ButtonBase](#) effettivamente, ma disponibile per [Button](#) tramite ereditarietà). [Button](#) "proprietario" dell'evento, ma il sistema di eventi indirizzati consente ai gestori per qualsiasi evento indirizzato di essere collegato a qualsiasi listener di istanza [UIElement](#) o [ContentElement](#) che altrimenti potrebbe collegare listener per un evento Common Language Runtime (CLR). Lo spazio dei nomi `xmlns` predefinito per questi nomi completi di attributo di evento è in genere lo spazio dei nomi `xmlns` WPF predefinito, ma è anche

possibile specificare spazi dei nomi con prefisso per eventi indirizzati personalizzati. Per altre informazioni su xmlns, vedere [Spazi dei nomi XAML e mapping dello spazio dei nomi per XAML WPF](#).

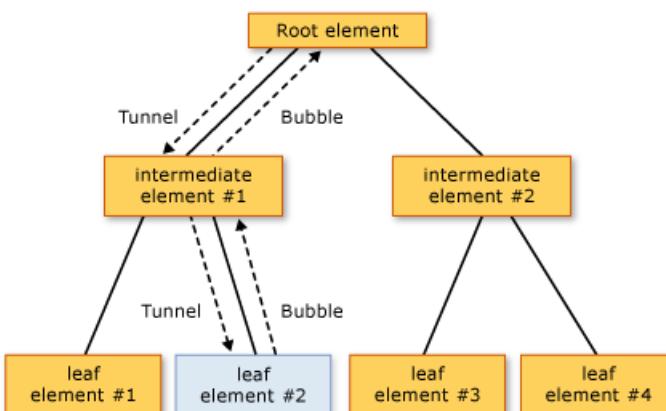
Eventi di input WPF

Un'applicazione frequente degli eventi indirizzati nella piattaforma WPF riguarda gli eventi di input. In WPF i nomi degli eventi indirizzati di tunneling sono preceduti dalla parola "Preview" per convenzione. Gli eventi di input sono spesso in coppia, con un evento che rappresenta l'evento di bubbling e l'altro quello di tunneling. Ad esempio, l'evento `KeyDown` e l'evento `PreviewKeyDown` hanno la stessa firma, il primo è l'evento di input di bubbling e il secondo è l'evento di input di tunneling. In alcuni casi, gli eventi di input hanno solo una versione di bubbling o solo una versione indirizzata diretta. Nella documentazione, gli argomenti relativi agli eventi indirizzati contengono riferimenti incrociati a eventi indirizzati simili con strategie di routing alternative, se disponibili, e le sezioni nelle pagine di riferimenti gestiti chiariscono la strategia di routing per ogni evento indirizzato.

Gli eventi di input WPF in coppia vengono implementati in modo che una singola azione di input dell'utente, ad esempio la pressione di un pulsante del mouse, generi entrambi gli eventi indirizzati della coppia in sequenza. Per prima cosa, viene generato l'evento di tunneling, che avanza nella route. Successivamente, viene generato l'evento di bubbling, che avanza nella route. I due eventi condividono letteralmente la stessa istanza di dati evento, perché la chiamata al metodo `RaiseEvent` nella classe di implementazione che genera l'evento di bubbling resta in attesa dei dati dell'evento dell'evento di tunneling e la riutilizza nel nuovo evento generato. I listener con gestori per l'evento di tunneling hanno l'opportunità di contrassegnare per primi l'evento indirizzato come gestito (prima i gestori di classi, quindi i gestori di istanze). Se un elemento nella route di tunneling ha contrassegnato l'evento indirizzato come gestito, i dati dell'evento già gestito vengono inviati per l'evento di bubbling e i gestori tipici associati per gli eventi di input di bubbling equivalenti non vengono richiamati. Dall'esterno, potrebbe sembrare che l'evento di bubbling gestito non sia mai stato generato. Questo comportamento di gestione è utile per la composizione di controlli in cui si vuole che tutti gli eventi di input basati su hit test o gli eventi di input basati su stato attivo vengano segnalati dal controllo finale, invece che dalle relative parti composite. L'elemento del controllo finale è più vicino alla radice nella composizione e quindi ha la possibilità di gestire tramite classi prima l'evento di tunneling ed eventualmente di "sostituire" tale evento indirizzato con un evento più specifico del controllo, come parte del codice sottostante la classe del controllo.

Come dimostrazione del funzionamento dell'elaborazione degli eventi di input, considerare l'esempio di evento di input seguente. Nell'illustrazione dell'albero seguente `leaf element #2` è l'origine di un

`PreviewMouseDown` e quindi di un evento `MouseDown`:



L'ordine di elaborazione degli eventi è il seguente:

1. `PreviewMouseDown` (tunneling) sull'elemento radice.
2. `PreviewMouseDown` (tunneling) sull'elemento intermedio 1.

3. `PreviewMouseDown` (tunneling) sull'elemento di origine 2.
4. `MouseDown` (bubbling) sull'elemento di origine 2.
5. `MouseDown` (bubbling) sull'elemento intermedio 1.
6. `MouseDown` (bubbling) sull'elemento radice.

Un delegato di un gestore di eventi indirizzato fornisce riferimenti a due oggetti: l'oggetto che ha generato l'evento e quello su cui è stato richiamato il gestore. L'oggetto in cui è stato richiamato il gestore è quello indicato dal parametro `sender`. L'oggetto in cui l'evento è stato generato per la prima volta viene segnalato dalla proprietà `Source` nei dati dell'evento. Un evento indirizzato può comunque essere generato e gestito dallo stesso oggetto, nel qual caso `sender` e `Source` sono identici (questo è il caso dei passaggi 3 e 4 nell'elenco di esempio di elaborazione degli eventi).

A causa del tunneling e del bubbling, gli elementi padre ricevono gli eventi di input in cui il `Source` è uno dei relativi elementi figlio. Quando è importante conoscere l'elemento di origine, è possibile identificare l'elemento di origine accedendo alla proprietà `Source`.

In genere, dopo che l'evento di input è stato contrassegnato come `Handled`, altri gestori non vengono richiamati. Solitamente è consigliabile contrassegnare gli eventi di input come gestiti non appena viene richiamato un gestore che applica la gestione logica specifica dell'applicazione al significato dell'evento di input.

L'eccezione a questa istruzione generale sullo stato `Handled` è che i gestori di eventi di input registrati per ignorare intenzionalmente `Handled` stato dei dati dell'evento verrebbero comunque richiamati lungo entrambe le route. Per altre informazioni, vedere [Eventi di anteprima](#) e [Contrassegno degli eventi indirizzati come gestiti e gestione delle classi](#).

Il modello di dati di evento condivisi tra eventi di tunneling e di bubbling e la generazione sequenziale di eventi prima di tunneling e quindi di bubbling non sono concetti generalmente veri per tutti gli eventi indirizzati. Questo comportamento viene implementato specificamente dal modo in cui i dispositivi di input WPF scelgono di generare e connettere le coppie di eventi di input. L'implementazione di eventi di input personalizzati è un scenario avanzato, ma si può scegliere di seguire tale modello anche per gli eventi di input personalizzati.

Alcune classi scelgono di gestire tramite classi determinati eventi di input, di solito con lo scopo di ridefinire il significato di un determinato evento di input generato dall'utente all'interno del controllo e di generare un nuovo evento. Per altre informazioni, vedere [Contrassegno degli eventi indirizzati come gestiti e gestione delle classi](#).

Per altre informazioni sull'input e su come input ed eventi interagiscono in scenari di applicazioni tipici, vedere [Cenni preliminari sull'input](#).

EventSetters ed EventTriggers

Negli stili è possibile includere alcuni XAML sintassi pre-dichiarata di gestione degli eventi nel markup usando una `EventSetter`. Quando lo stile viene applicato, il gestore a cui si fa riferimento viene aggiunto all'istanza a cui è stato applicato lo stile. È possibile dichiarare un `EventSetter` solo per un evento indirizzato. Di seguito è riportato un esempio. Si noti che il metodo `b1SetColor` a cui si fa riferimento qui si trova in un file code-behind.

```

<StackPanel
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SDKSample.EventOvw2"
    Name="dpanel2"
    Initialized="PrimeHandledToo"
>
    <StackPanel.Resources>
        <Style TargetType="{x:Type Button}">
            <EventSetter Event="Click" Handler="b1SetColor"/>
        </Style>
    </StackPanel.Resources>
    <Button>Click me</Button>
    <Button Name="ThisButton" Click="HandleThis">
        Raise event, handle it, use handled=true handler to get it anyway.
    </Button>
</StackPanel>

```

Il vantaggio ottenuto è che è probabile che lo stile contenga una grande quantità di informazioni che possono essere applicate a qualsiasi pulsante nell'applicazione e che il [EventSetter](#) faccia parte di tale stile promuova il riutilizzo del codice anche a livello di markup. Inoltre, un [EventSetter](#) estrae i nomi di metodo per i gestori un passaggio più lontano dal markup generale dell'applicazione e della pagina.

Un'altra sintassi specializzata che combina gli eventi indirizzati e le funzionalità di animazione di WPF è un [EventTrigger](#). Come con [EventSetter](#), solo gli eventi indirizzati possono essere usati per un [EventTrigger](#). In genere, una [EventTrigger](#) viene dichiarata come parte di uno stile, ma una [EventTrigger](#) può essere dichiarata anche in elementi a livello di pagina come parte della raccolta di [Triggers](#) o in una [ControlTemplate](#). Un [EventTrigger](#) consente di specificare un [Storyboard](#) che viene eseguito ogni volta che un evento indirizzato raggiunge un elemento nella propria route che dichiara un [EventTrigger](#) per quell'evento. Il vantaggio di un [EventTrigger](#) rispetto alla semplice gestione dell'evento e l'avvio di uno storyboard esistente consiste nel fatto che una [EventTrigger](#) fornisce un migliore controllo sullo storyboard e sul comportamento in fase di esecuzione. Per altre informazioni, vedere [Usare i trigger di evento per controllare uno storyboard dopo il relativo avvio](#).

Altre informazioni sugli eventi indirizzati

Questo argomento illustra gli eventi indirizzati principalmente con lo scopo di descrivere i concetti di base e di fornire istruzioni su come e quando rispondere agli eventi indirizzati già presenti nei vari controlli ed elementi di base. È tuttavia possibile creare un evento indirizzato personalizzato in una classe personalizzata insieme a tutto il supporto necessario, come delegati e classi di dati di evento specializzati. Il proprietario dell'evento indirizzato può essere qualsiasi classe, ma gli eventi indirizzati devono essere generati da e gestiti da [UIElement](#) o [ContentElement](#) classi derivate per essere utili. Per altre informazioni sugli eventi personalizzati, vedere [Creare un evento indirizzato personalizzato](#).

Vedere anche

- [EventManager](#)
- [RoutedEventArgs](#)
- [RoutedEventArgs](#)
- [Contrassegno degli eventi indirizzati come gestiti e gestione delle classi](#)
- [Cenni preliminari sull'input](#)
- [Panoramica sull'esecuzione di comandi](#)
- [Proprietà di dipendenza personalizzate](#)
- [Strutture ad albero in WPF](#)
- [Modelli di eventi deboli](#)

Cenni preliminari sugli eventi associati

04/11/2019 • 12 minutes to read • [Edit Online](#)

Extensible Application Markup Language (XAML) definisce un componente del linguaggio e un tipo di evento chiamato *evento associato*. Il concetto di evento associato consente di aggiungere un gestore per un determinato evento a un elemento arbitrario, anziché a un elemento che definisce o eredita effettivamente l'evento. In questo caso, né l'oggetto che genera potenzialmente l'evento, né l'istanza di gestione di destinazione definisce o possiede in altro modo l'evento.

Prerequisites

Questo argomento si presuppone di aver letto [Cenni preliminari sugli eventi indirizzati](#) e [Cenni preliminari su XAML \(WPF\)](#).

Sintassi per gli eventi associati

Gli eventi associati hanno una sintassi XAML e uno schema di codifica che deve essere usato dal codice di supporto per supportare l'utilizzo dell'evento associato.

Nella sintassi XAML, l'evento associato viene specificato non solo dal nome dell'evento, ma dal tipo proprietario più il nome dell'evento, separato da un punto (.). Dato che il nome di evento è qualificato con il nome del tipo proprietario, la sintassi dell'evento associato consente l'associazione di tale evento a qualsiasi elemento di cui è possibile creare un'istanza.

Ad esempio, di seguito è riportata la sintassi XAML per collegare un gestore per un evento associato

`NeedsCleaning` personalizzato:

```
<aqua: Aquarium Name="theAquarium" Height="600" Width="800" aqua: AquariumFilter. NeedsCleaning="WashMe" />
```

Si noti il prefisso `aqua:`, necessario in questo caso perché l'evento associato è un evento personalizzato tratto da un xmlns mappato personalizzato.

Modalità di implementazione degli eventi associati in WPF

In WPF gli eventi associati sono supportati da un campo `RoutedEventArgs` e vengono instradati attraverso la struttura ad albero dopo la generazione. In genere, l'origine dell'evento associato (oggetto che genera l'evento) è un'origine di sistema o servizio e l'oggetto che esegue il codice che genera l'evento non è pertanto parte diretta dell'albero degli elementi.

Scenari per gli eventi associati

In WPF gli eventi associati sono presenti in determinate aree di funzionalità in cui è presente un'astrazione a livello di servizio, ad esempio per gli eventi abilitati dalla classe `Mouse` statica o dalla classe `Validation`. Le classi che interagiscono con il servizio o lo usano possono usare l'evento nella sintassi dell'evento associato o scegliere di usarlo come un evento indirizzato che fa parte del modo in cui la classe integra le funzionalità del servizio.

Sebbene WPF definisca un certo numero di eventi associati, gli scenari in cui si utilizzerà o si gestirà direttamente l'evento associato sono molto limitati. In genere, l'evento associato serve a scopo di architettura, ma viene quindi inoltrato a un evento indirizzato non associato (supportato con un evento CLR "wrapper").

Ad esempio, l'evento associato sottostante `Mouse.MouseDown` può essere gestito più facilmente in qualsiasi `UIElement` specificato utilizzando `MouseDown` su tale `UIElement` anziché gestire la sintassi degli eventi associati in XAML o nel codice. L'evento associato assolve a uno scopo nell'architettura perché consente l'espansione futura dei dispositivi di input. Il dispositivo ipotetico deve solo generare `Mouse.MouseDown` per simulare l'input del mouse e non deve derivare da `Mouse` a tale scopo. Tuttavia, questo scenario implica la gestione del codice degli eventi e la gestione XAML dell'evento associato non è pertinente per questo scenario.

Gestione di un evento associato in WPF

Il processo di gestione di un evento associato e il codice del gestore che verrà scritto è fondamentalmente lo stesso necessario per un evento indirizzato.

In generale, un evento associato WPF non è molto diverso da un evento indirizzato WPF. Le differenze sono il modo in cui l'evento viene originato e il modo in cui viene esposto da una classe come membro (che influiscono anche sulla sintassi del gestore XAML).

Tuttavia, come indicato in precedenza, gli eventi associati WPF esistenti non sono destinati in modo specifico alla gestione in WPF. Più spesso, lo scopo dell'evento è quello di consentire la segnalazione di uno stato da parte di un elemento composto a un elemento padre nella composizione, nel qual caso l'evento viene in genere generato nel codice e si basa inoltre sulla gestione della classe nella classe padre rilevante. Ad esempio, è previsto che gli elementi all'interno di un `Selector` generino l'evento `Attached Selected`, che è quindi gestito dalla classe `Selector` e quindi potenzialmente convertito dalla classe `Selector` in un evento indirizzato diverso, `SelectionChanged`. Per altre informazioni sugli eventi indirizzati e sulla gestione delle classi, vedere [Contrassegno degli eventi indirizzati come gestiti e gestione delle classi](#).

Definizione di eventi associati personalizzati come eventi indirizzati

Se si esegue la derivazione da classi base WPF comuni, è possibile implementare eventi allegati personalizzati includendo determinati metodi di modello nella classe e usando metodi di utilità già presenti nelle classi di base.

Il modello è il seguente:

- Metodo che **aggiunge il gestore `eventName`** con due parametri. Il primo parametro è l'istanza di a cui viene aggiunto il gestore eventi. Il secondo parametro è il gestore eventi da aggiungere. Il metodo deve essere `public` e `static`, senza alcun valore restituito.
- Metodo **`RemoveEventNamehandler`** con due parametri. Il primo parametro è l'istanza da cui viene rimosso il gestore eventi. Il secondo parametro è il gestore eventi da rimuovere. Il metodo deve essere `public` e `static`, senza alcun valore restituito.

Il metodo della funzione di accesso del **gestore `AddEventName`** facilita l'elaborazione XAML quando gli attributi del gestore eventi associati sono dichiarati su un elemento. Anche i metodi **`AddEventNamehandler`** e **`RemoveEventNamehandler`** abilitano l'accesso al codice all'archivio del gestore eventi per l'evento associato.

Questo modello generale non è ancora abbastanza preciso per l'implementazione pratica in un Framework, perché qualsiasi implementazione del reader XAML potrebbe avere schemi diversi per l'identificazione degli eventi sottostanti nel linguaggio e nell'architettura di supporto. Questo è uno dei motivi per cui WPF implementa gli eventi associati come eventi indirizzati. L'identificatore da utilizzare per un evento (`RoutedEventArgs`) è già definito dal sistema di eventi WPF. Inoltre, il routing di un evento è un'estensione di implementazione naturale nel concetto a livello di linguaggio XAML di un evento associato.

L'implementazione dell'**aggiunta del gestore `eventName`** per un evento associato WPF consiste nella chiamata di `AddHandler` con l'evento indirizzato e il gestore come argomenti.

Questa strategia di implementazione e il sistema di eventi indirizzati in generale limitano la gestione degli eventi associati a `UIElement` classi derivate o `ContentElement` classi derivate, perché solo le classi dispongono di

implementazioni di [AddHandler](#).

Il codice seguente, ad esempio, definisce il `NeedsCleaning` evento associato sulla classe Owner `Aquarium`, usando la strategia di evento associato WPF di dichiarazione dell'evento associato come un evento indirizzato.

```
public static readonly RoutedEvent NeedsCleaningEvent = EventManager.RegisterRoutedEvent("NeedsCleaning",
    RoutingStrategy.Bubble, typeof(RoutedEventHandler), typeof(AquariumFilter));
public static void AddNeedsCleaningHandler(DependencyObject d, RoutedEventHandler handler)
{
    UIElement uie = d as UIElement;
    if (uie != null)
    {
        uie.AddHandler(AquariumFilter.NeedsCleaningEvent, handler);
    }
}
public static void RemoveNeedsCleaningHandler(DependencyObject d, RoutedEventHandler handler)
{
    UIElement uie = d as UIElement;
    if (uie != null)
    {
        uie.RemoveHandler(AquariumFilter.NeedsCleaningEvent, handler);
    }
}
```

```
Public Shared ReadOnly NeedsCleaningEvent As RoutedEvent = EventManager.RegisterRoutedEvent("NeedsCleaning",
    RoutingStrategy.Bubble, GetType(RoutedEventHandler), GetType(AquariumFilter))
Public Shared Sub AddNeedsCleaningHandler(ByVal d As DependencyObject, ByVal handler As RoutedEventHandler)
    Dim uie As UIElement = TryCast(d, UIElement)
    If uie IsNot Nothing Then
        uie.AddHandler(AquariumFilter.NeedsCleaningEvent, handler)
    End If
End Sub
Public Shared Sub RemoveNeedsCleaningHandler(ByVal d As DependencyObject, ByVal handler As RoutedEventHandler)
    Dim uie As UIElement = TryCast(d, UIElement)
    If uie IsNot Nothing Then
        uie.RemoveHandler(AquariumFilter.NeedsCleaningEvent, handler)
    End If
End Sub
```

Si noti che il metodo usato per stabilire il campo dell'identificatore di evento associato, [RegisterRoutedEvent](#), è in realtà lo stesso metodo usato per registrare un evento indirizzato non collegato. Gli eventi associati e gli eventi indirizzati vengono tutti registrati in un archivio interno centralizzato. Questa implementazione dell'archivio degli eventi consente la considerazione del concetto di "eventi come interfaccia" presentata in [Cenni preliminari sugli eventi indirizzati](#).

Generazione di un evento associato WPF

Non è in genere necessario generare eventi allegati esistenti definiti da WPF dal codice. Questi eventi seguono il modello concettuale generale "Service" e le classi di servizio come [InputManager](#) sono responsabili della generazione degli eventi.

Tuttavia, se si definisce un evento associato personalizzato basato sul modello WPF di eventi associati basandoli in [RoutedEventArgs](#), è possibile usare [RaiseEvent](#) per generare un evento associato da qualsiasi [UIElement](#) o [ContentElement](#). Per generare un evento indirizzato (collegato o meno), è necessario dichiarare un particolare elemento nell'albero degli elementi come origine evento; tale origine viene segnalata come il chiamante del [RaiseEvent](#). È responsabilità del servizio determinare quale elemento viene riportato come origine nell'albero.

Vedere anche

- [Cenni preliminari sugli eventi indirizzati](#)
- [Descrizione dettagliata della sintassi XAML](#)
- [Classi XAML e personalizzate per WPF](#)

Eventi di durata degli oggetti

04/11/2019 • 8 minutes to read • [Edit Online](#)

In questo argomento vengono descritti gli eventi WPF specifici che denotano le fasi della durata di un oggetto in termini di creazione, uso e distruzione.

Prerequisites

In questo argomento si presuppongono la conoscenza delle proprietà di dipendenza dal punto di vista di un consumer di proprietà di dipendenza esistenti nelle classi Windows Presentation Foundation (WPF), nonché la lettura dell'argomento [Panoramica sulle proprietà di dipendenza](#). Per seguire gli esempi illustrati in questo argomento, è anche necessario conoscere Extensible Application Markup Language (XAML) (vedere [Cenni preliminari su XAML \(WPF\)](#)) e saper scrivere applicazioni WPF.

Eventi di durata degli oggetti

Tutti gli oggetti nel codice gestito di Microsoft .NET Framework passano attraverso un insieme simile di fasi di vita, creazione, utilizzo e distruzione. Per molti oggetti la fase di finalizzazione della vita si verifica nell'ambito della fase di distruzione. Gli oggetti WPF più specificamente gli oggetti visivi che WPF identifica come elementi, hanno una serie comune di fasi di vita dell'oggetto. I modelli di programmazione e applicazione WPF espongono queste fasi come una serie di eventi. Esistono quattro tipi principali di oggetti in WPF in relazione agli eventi di durata: gli elementi in generale, gli elementi finestra, gli host di navigazione e gli oggetti applicazione. Le finestre e gli host di navigazione fanno parte anche del più ampio raggruppamento di oggetti visivi (elementi). In questo argomento vengono descritti gli eventi di durata che sono comuni a tutti gli elementi; vengono quindi introdotti quelli più specifici che si applicano alle definizioni dell'applicazione, alle finestre o agli host di navigazione.

Eventi di durata comuni degli elementi

Qualsiasi elemento a livello di Framework WPF, ovvero gli oggetti che derivano da [FrameworkElement](#) o [FrameworkContentElement](#), presenta tre eventi di durata comuni: [Initialized](#), [Loaded](#) e [Unloaded](#).

Inizializzato

[Initialized](#) viene generato per primo e approssimativamente corrisponde all'inizializzazione dell'oggetto da parte della chiamata al relativo costruttore. Poiché l'evento si verifica in seguito all'inizializzazione, tutte le proprietà dell'oggetto sono sicuramente impostate. Un'eccezione è costituita da utilizzi di espressioni come le risorse dinamiche o l'associazione. Si tratta di espressioni non valutate. Come conseguenza del requisito per il quale vengono impostate tutte le proprietà, la sequenza di [Initialized](#) generati da elementi annidati definiti nel markup sembra essere eseguita prima dell'ordine degli elementi più profondi nell'albero degli elementi, quindi gli elementi padre verso la radice. Questo ordine è determinato dal fatto che le relazioni padre-figlio e il contenimento sono proprietà e pertanto l'elemento padre non può segnalare l'inizializzazione finché tutti gli elementi figlio che riempiono la proprietà non sono stati completamente inizializzati.

Quando si scrivono i gestori in risposta all'evento [Initialized](#), è necessario tenere presente che non vi è alcuna garanzia che tutti gli altri elementi nella struttura ad albero dell'elemento (albero logico o albero visuale) in cui è collegato il gestore siano stati creati, in particolare l'elemento padre elementi. Le variabili membro potrebbero essere Null oppure le origini dati potrebbero non essere ancora popolate dall'associazione sottostante (anche a livello di espressione).

Caricato

Il [Loaded](#) viene generato successivamente. L'evento [Loaded](#) viene generato prima del rendering finale, ma dopo

che il sistema di layout ha calcolato tutti i valori necessari per il rendering. [Loaded](#) comporta che l'albero logico in cui è contenuto un elemento sia completato e si connette a un'origine della presentazione che fornisce l'HWND e la superficie di rendering. Le data binding standard (associazione a origini locali, ad esempio altre proprietà o origini dati definite direttamente), si sono verificate prima di [Loaded](#). È possibile che il data binding asincrono (a origini esterne o dinamiche) si sia verificato, ma a causa della sua natura asincrona non è possibile averne la certezza.

Il meccanismo in base al quale viene generato l'evento [Loaded](#) è diverso rispetto a [Initialized](#). L'evento [Initialized](#) viene generato elemento per elemento, senza un coordinamento diretto da parte di un albero degli elementi completato. Al contrario, l'evento [Loaded](#) viene generato come sforzo coordinato nell'intero albero degli elementi (in particolare, l'albero logico). Quando tutti gli elementi dell'albero sono in uno stato in cui sono considerati caricati, l'evento [Loaded](#) viene innanzitutto generato sull'elemento radice. L'evento [Loaded](#) viene quindi generato successivamente in ogni elemento figlio.

NOTE

Questo comportamento potrebbe sembrare a prima vista analogo al tunneling per un evento indirizzato. Tuttavia, le informazioni non vengono passate da evento a evento. Ogni elemento ha sempre la possibilità di gestire l'evento [Loaded](#) e contrassegnare i dati dell'evento come gestiti non ha alcun effetto oltre tale elemento.

Scaricato

[Unloaded](#) viene generato per ultimo e viene avviato dall'origine della presentazione o dalla rimozione del padre visivo. Quando [Unloaded](#) viene generato e gestito, l'elemento che rappresenta l'origine dell'evento padre (come determinato dalla proprietà [Parent](#)) o qualsiasi elemento specificato verso l'alto negli alberi logici o visivi potrebbe essere già stato annullato, vale a dire che data binding, riferimenti alle risorse e stili non può essere impostato sul valore di runtime normale o finale noto.

Elementi del modello di applicazione di eventi di durata

La compilazione sugli eventi di durata comuni per gli elementi sono gli elementi del modello di applicazione seguenti: [Application](#), [Window](#), [Page](#), [NavigationWindow](#) e [Frame](#). Questi estendono gli eventi di durata comuni con eventi aggiuntivi, pertinenti al relativo scopo specifico e vengono descritti in dettaglio nelle sezioni seguenti:

- [Application](#): Cenni preliminari sulla gestione delle applicazioni.
- [Window](#): Cenni preliminari sulle finestre WPF.
- [Page](#), [NavigationWindow](#) e [Frame](#): Cenni preliminari sulla navigazione.

Vedere anche

- [Precedenza del valore della proprietà di dipendenza](#)
- [Cenni preliminari sugli eventi indirizzati](#)

Impostazione degli eventi indirizzati come gestiti e gestione delle classi

23/10/2019 • 30 minutes to read • [Edit Online](#)

I gestori per un evento indirizzato possono contrassegnare l'evento come gestito all'interno dei dati dell'evento. La gestione dell'evento abbrevia efficacemente la route. La gestione delle classi è un concetto di programmazione supportato dagli eventi indirizzati. Un gestore classi ha l'opportunità di gestire un evento indirizzato specifico a livello di classe con un gestore richiamato prima di qualsiasi gestore istanze in qualsiasi istanza di classe.

Prerequisiti

Questo argomento elabora i concetti introdotti in [Cenni preliminari sugli eventi indirizzati](#).

Quando contrassegnare eventi come gestiti

Quando si imposta il valore della `Handled` proprietà su `true` nei dati dell'evento per un evento indirizzato, questo viene definito "contrassegno dell'evento gestito". Non esiste una regola assoluta riguardo a quando contrassegnare gli eventi indirizzati come gestiti, né per un autore di applicazioni né per un autore di controlli che risponde a eventi indirizzati esistenti o implementa nuovi eventi indirizzati. Nella maggior parte dei casi, il concetto di "gestito", come avviene nei dati degli eventi dell'evento indirizzato, deve essere usato come protocollo limitato per le risposte dell'applicazione ai vari eventi indirizzati esposti nelle WPF API, nonché per qualsiasi evento indirizzato personalizzato. Un altro modo di considerare il concetto di "gestito" è che in genere è necessario contrassegnare un evento indirizzato come gestito se il codice ha risposto all'evento indirizzato in modo significativo e relativamente completo. In genere, deve essere presente una sola risposta significativa che richiede implementazioni del gestore separate per qualsiasi singola occorrenza di un evento indirizzato. Se sono necessarie più risposte, il codice richiesto deve essere implementato tramite logica dell'applicazione concatenata a un singolo gestore, anziché usare il sistema degli eventi indirizzati per l'inoltro. Anche il concetto di "significativo" è soggettivo e dipende dall'applicazione o dal codice. Come indicazione generale, ecco alcuni esempi di "risposta significativa": impostazione dello stato attivo, modifica dello stato pubblico, impostazione delle proprietà che influiscono sulla rappresentazione visiva e generazione di altri nuovi eventi. Ecco inoltre alcuni esempi di risposte non significative: modifica dello stato privato (senza alcun impatto visivo o sulla rappresentazione programmatica), registrazione di eventi o esame degli argomenti di un evento e scelta di non rispondervi.

Il comportamento del sistema di eventi indirizzati rafforza questo modello di "risposta significativa" per l'uso dello stato gestito di un evento indirizzato, perché XAML i gestori aggiunti in o `AddHandler` la firma comune di non vengono richiamati in risposta a un evento indirizzato in cui l'evento i dati sono già contrassegnati come gestiti. È necessario eseguire il lavoro aggiuntivo di aggiunta di un gestore con la `handledEventsToo` versione del parametro `AddHandler(RoutedEvent, Delegate, Boolean)()` per gestire gli eventi indirizzati contrassegnati come gestiti da partecipanti precedenti nella route dell'evento.

In alcuni casi, i controlli stessi contrassegnano determinati eventi indirizzati come gestiti. Un evento indirizzato gestito rappresenta una decisione da parte degli autori di controlli WPF in base alla quale le azioni del controllo in risposta all'evento indirizzato sono significative o complete come parte dell'implementazione del controllo e che l'evento non richiede gestione aggiuntiva. Questo avviene in genere aggiungendo un gestore classi per un evento oppure eseguendo l'override di uno dei metodi virtuali del gestore classi presenti in una classe base. Se necessario, è comunque possibile trovare soluzioni alternative per questa gestione degli eventi. Vedere [Soluzioni alternative all'eliminazione di eventi da parte dei controlli](#) più avanti in questo argomento.

Confronto tra eventi (di tunneling) "di anteprima" ed eventi di bubbling e gestione degli eventi

Gli eventi indirizzati di anteprima sono eventi che seguono una route di tunneling attraverso l'albero degli elementi. Il termine "anteprima" usato nella convenzione di denominazione è indicativo del principio generale per gli eventi di input in base al quale gli eventi indirizzati (di tunneling) di anteprima vengono generati prima dell'evento indirizzato di bubbling equivalente. Inoltre, gli eventi indirizzati di input dotati di una coppia di tunneling e bubbling hanno una logica di gestione distinta. Se l'evento indirizzato di tunneling/anteprima viene contrassegnato come gestito da un listener di eventi, l'evento indirizzato di bubbling verrà contrassegnato come gestito anche prima che qualsiasi listener dell'evento indirizzato di bubbling lo riceva. Gli eventi indirizzati di tunneling e bubbling sono tecnicamente eventi separati, ma condividono intenzionalmente la stessa istanza dei dati degli eventi per permettere questo comportamento.

La connessione tra gli eventi indirizzati di tunneling e bubbling viene eseguita tramite l'implementazione interna del modo in cui ogni classe WPF specifica genera i propri eventi indirizzati dichiarati e questo avviene per gli eventi indirizzati di input associati. Tuttavia, se questa implementazione a livello di classe non esiste, non vi è alcuna connessione tra un evento indirizzato di tunneling e un evento indirizzato di bubbling che condividono lo schema di denominazione: senza questa implementazione, i due eventi indirizzati sono completamente separati e non vengono generati in sequenza né condividono i dati degli eventi.

Per altre informazioni su come implementare coppie di eventi indirizzati di input di tunneling/bubbling in una classe personalizzata, vedere [Creare un evento indirizzato personalizzato](#).

Gestori classi e gestori istanze

Gli eventi indirizzati considerano due tipi diversi di listener per l'evento: listener di classi e listener di istanze. I listener di classi sono disponibili perché i tipi hanno [EventManager](#) chiamato [unaRegisterClassHandler](#) particolare API, nel relativo costruttore statico o hanno eseguito l'override di un metodo virtuale del gestore classi da una classe di base dell'elemento. I listener di istanze sono istanze di classe/elementi particolari in cui uno o più gestori sono stati collegati per l'evento indirizzato da una [AddHandler](#) chiamata a. Gli WPF eventi indirizzati esistenti effettuano [AddHandler](#) chiamate a come parte delle implementazioni di aggiunta{} e rimozione{} del wrapper di eventi Common Language Runtime (CLR), che è anche il modo XAML in cui il meccanismo semplice di fissaggio di gestori eventi tramite una sintassi di attributo è abilitato. Pertanto, anche l' XAML utilizzo semplice equivale in definitiva a una [AddHandler](#) chiamata.

Gli elementi all'interno dell'albero visuale vengono controllati per individuare le eventuali implementazioni di gestori registrati. I gestori vengono potenzialmente richiamati lungo la route, nell'ordine ereditato nel tipo della strategia di routing per l'evento indirizzato specifico. Ad esempio, gli eventi indirizzati di bubbling richiameranno prima di tutto i gestori collegati allo stesso elemento che ha generato l'evento indirizzato. L'evento indirizzato viene propagato al successivo elemento padre e così via fino a raggiungere l'elemento radice dell'applicazione.

Dal punto di vista dell'elemento radice in una route di bubbling, se la gestione delle classi o qualsiasi elemento più vicino all'origine dell'evento indirizzato richama gestori che contrassegnano gli argomenti dell'evento come gestiti, i gestori negli elementi radice non vengono richiamati e la route dell'evento viene abbreviata in modo efficace prima di raggiungere l'elemento radice. Tuttavia, la route non viene completamente interrotta, perché è possibile aggiungere gestori con una speciale condizione in base alla quale devono comunque essere richiamati, anche se un gestore classi o un gestore istanze ha contrassegnato l'evento indirizzato come gestito. Questo comportamento viene descritto in [Aggiunta di gestori istanze generati anche se gli eventi sono contrassegnati come gestiti](#) più avanti in questo argomento.

A un livello più profondo rispetto a quello della route degli eventi esistono potenzialmente anche più gestori classi che agiscono su qualsiasi istanza specifica di una classe. Questo avviene perché il modello di gestione

delle classi per eventi indirizzati permette a tutte le possibili classi in una gerarchia di classi di registrare ciascuna il proprio gestore classi per ogni evento indirizzato. Ogni gestore classi viene aggiunto a un archivio interno e quando viene creata la route degli eventi per un'applicazione, i gestori classi vengono tutti aggiunti alla route. I gestori classi vengono aggiunti alla route in modo che venga richiamato per primo il gestore della classe più derivata, richiamando quindi i gestori classi da ogni classe base successiva. In genere i gestori classi non sono registrati in modo da rispondere anche a eventi indirizzati già contrassegnati come gestiti. Di conseguenza, questo meccanismo di gestione delle classi permette di scegliere tra due opzioni:

- Le classi derivate possono completare la gestione delle classi ereditata dalla classe base aggiungendo un gestore che non contrassegna l'evento indirizzato come gestito, perché il gestore della classe base verrà richiamato in un momento successivo al gestore delle classi derivate.
- Le classi derivate possono sostituire la gestione delle classi dalla classe base tramite laggiunta di un gestore classi che contrassegna l'evento indirizzato come gestito. È necessario usare questo approccio con cautela, perché potrebbe modificare la progettazione dei controlli di base desiderata, ad esempio in aree come l'aspetto visivo, la logica di stato, la gestione degli input e la gestione dei comandi.

Gestione delle classi degli eventi indirizzati tramite classi di base dei controlli

Nel nodo di ogni elemento specifico in una route di eventi i listener di classi possono rispondere all'evento indirizzato prima di qualsiasi listener di istanze per l'elemento. Per questo motivo, i gestori classi vengono usati talvolta per eliminare gli eventi indirizzati che l'implementazione di una determinata classe di controlli non desidera propagare ulteriormente oppure per fornire una gestione speciale dell'evento indirizzato che è una caratteristica della classe. Ad esempio, una classe potrebbe generare il proprio evento specifico della classe che contiene più specifiche sul significato di una certa condizione di input utente nel contesto di una determinata classe. L'implementazione della classe potrebbe quindi contrassegnare come gestito l'evento indirizzato più generale. I gestori di classi vengono in genere aggiunti in modo che non vengano richiamati per gli eventi indirizzati in cui i dati degli eventi condivisi erano già contrassegnati come gestiti [RegisterClassHandler\(Type, RoutedEvent, Delegate, Boolean\)](#), ma per i casi atipici è presente anche una firma che regista i gestori di classi da richiamare anche quando gli eventi indirizzati sono Contrassegnato come gestito.

Metodi virtuali dei gestori classi

Alcuni elementi, in particolare gli elementi [UIElement](#) di base come, espongono metodi virtuali "on * Event" e "OnPreview*Event" vuoti che corrispondono al relativo elenco di eventi indirizzati pubblici. È possibile eseguire l'override di questi metodi virtuali per implementare un gestore classi per l'evento indirizzato. Le classi degli elementi di base registrano questi metodi virtuali come gestore di classe per ogni evento indirizzato usando [RegisterClassHandler\(Type, RoutedEvent, Delegate, Boolean\)](#) come descritto in precedenza. I metodi virtuali On*Event semplificano notevolmente l'implementazione della gestione delle classi per gli eventi indirizzati pertinenti, senza richiedere attività di inizializzazione speciali nei costruttori statici per ogni tipo. È ad esempio possibile aggiungere la gestione delle classi per [DragEnter](#) l'evento in [UIElement](#) qualsiasi classe derivata eseguendo l'override [OnDragEnter](#) del metodo virtuale. All'interno dell'override è possibile gestire l'evento indirizzato, generare altri eventi, avviare una logica specifica della classe che può modificare le proprietà degli elementi nelle istanze o scegliere qualsiasi combinazione di queste azioni. È in genere consigliabile chiamare l'implementazione base in questi override, anche se si contrassegna l'evento come gestito. La chiamata dell'implementazione base è fortemente consigliata perché il metodo virtuale è incluso nella classe base. Il modello virtuale protetto standard costituito dalla chiamata delle implementazioni base da ogni metodo virtuale essenzialmente sostituisce ed eguaglia un meccanismo simile che è nativo per la gestione delle classi degli eventi indirizzati, in cui i gestori classi per tutte le classi in una gerarchia di classi vengono chiamati in qualsiasi istanza specifica, a partire dal gestore della classe più derivata e continuando fino al gestore della classe base. È necessario omettere la chiamata dell'implementazione base solo se la classe prevede un requisito intenzionale relativo alla modifica della logica di gestione delle classi base. Se si chiamerà l'implementazione base prima o dopo l'override del codice dipende dalla natura dell'implementazione.

Gestione delle classi degli eventi di input

I metodi virtuali dei gestori classi vengono tutti registrati in modo da essere richiamati solo in presenza di dati degli eventi condivisi che non sono già stati contrassegnati come gestiti. Inoltre, solo per gli eventi di input, le versioni di tunneling e bubbling vengono normalmente generate in sequenza e condividono i dati degli eventi. Di conseguenza, per una specifica coppia di gestori classi di eventi di input in cui uno corrisponde alla versione di tunneling e l'altro alla versione di bubbling, si potrebbe non voler contrassegnare immediatamente l'evento come gestito. Se si implementa il metodo virtuale della gestione delle classi di tunneling per contrassegnare l'evento come gestito, si impedirà al gestore classi di bubbling di essere richiamato, oltre a impedire la chiamata dei gestori istanze normalmente registrati per l'evento di tunneling o di bubbling.

Quando la gestione delle classi in un nodo è completa, vengono presi in considerazione i listener di istanze.

Aggiunta di gestori istanze generati anche se gli eventi sono contrassegnati come gestiti

Il [AddHandler](#) metodo fornisce un overload specifico che consente di aggiungere gestori che verranno richiamati dal sistema di eventi ogni volta che un evento raggiunge l'elemento di gestione nella route, anche se un altro gestore ha già modificato i dati dell'evento per contrassegnare evento gestito. Questo non è il comportamento più comune. In genere, i gestori possono essere scritti in modo da modificare tutte le aree del codice dell'applicazione che potrebbero essere influenzate da un evento, indipendentemente dal punto in cui questo è stato gestito nell'albero degli elementi, anche nei casi in cui si desiderano più risultati finali. Inoltre, in genere, un solo elemento deve rispondere all'evento e la logica dell'applicazione appropriata è già stata applicata. Tuttavia, è disponibile l'overload `handledEventsToo` per i casi eccezionali in cui un altro elemento in un albero di elementi o in una composizione di controlli ha già contrassegnato un evento come gestito, ma altri elementi in posizione superiore o inferiore nell'albero degli elementi (a seconda della route) vogliono comunque che i rispettivi gestori vengano richiamati.

Quando contrassegnare eventi gestiti come non gestiti

In genere, gli eventi indirizzati contrassegnati come gestiti non devono essere contrassegnati come non gestiti [Handled](#) (reimpostati `false`) anche da gestori che agiscono su `handledEventsToo`. Tuttavia, alcuni eventi di input hanno rappresentazioni di eventi di alto livello e di livello inferiore che possono sovrapporsi quando l'evento di alto livello viene visualizzato in una posizione nell'albero e quello di basso livello viene visualizzato in un'altra posizione. Si consideri, ad esempio, il caso in cui un elemento figlio è in ascolto di un evento [TextInput](#) chiave di alto livello, ad esempio mentre un elemento padre è in ascolto [KeyDown](#) di un evento di basso livello, ad esempio. Se l'elemento padre gestisce l'evento di basso livello, l'evento di livello superiore può essere eliminato anche nell'elemento figlio che dovrebbe avere la prima opportunità di gestire l'evento.

In queste situazioni può essere necessario aggiungere gestori sia agli elementi padre sia agli elementi figlio per l'evento di basso livello. L'implementazione del gestore dell'elemento figlio può contrassegnare l'evento di basso livello come gestito, ma l'implementazione del gestore dell'elemento padre lo imposterebbe di nuovo come non gestito per permettere ad altri elementi di livello superiore nell'albero (nonché all'evento di alto livello) di rispondere. Questa situazione dovrebbe essere piuttosto rara.

Eliminazione intenzionale di eventi di input per la composizione dei controlli

Lo scenario principale in cui viene usata la gestione delle classi per eventi indirizzati riguarda gli eventi di input e i controlli composti. Un controllo composto è per definizione composto da più controlli pratici o classi base di controlli. Spesso l'autore del controllo desidera comporre in modo uniforme tutti i possibili eventi di input che possono essere generati da ognuno dei sottocomponenti, per segnalare l'intero controllo come singola origine evento. In alcuni casi, l'autore del controllo potrebbe desiderare di eliminare interamente gli eventi dai componenti oppure sostituire un evento definito da un componente che contiene più informazioni o implica un comportamento più specifico. L'esempio canonico che è immediatamente visibile a qualsiasi autore di

componenti è il Windows Presentation Foundation (WPF) modo in cui un [Button](#) gestisce qualsiasi evento del mouse che alla fine verrà risolto nell'evento intuitivo [Click](#) che tutti i pulsanti hanno: un evento.

La [Button](#) classe di base [ButtonBase\(\)](#) deriva da [Control](#), che a sua volta deriva [FrameworkElement](#) da [UIElement](#) e la maggior parte dell'infrastruttura di eventi necessaria per l' [UIElement](#) elaborazione dell'input del controllo è disponibile a livello di. In particolare, [UIElement](#) elabora eventi [Mouse](#) generali che gestiscono l'hit testing per il cursore del mouse all'interno dei limiti e fornisce eventi distinti per le azioni dei [MouseLeftButtonDown](#) pulsanti più comuni, ad esempio. [UIElement](#) fornisce inoltre un oggetto virtuale [OnMouseLeftButtonDown](#) vuoto come gestore della classe pre-registrato [MouseLeftButtonDown](#) per ed [ButtonBase](#) esegue l'override. Analogamente [ButtonBase](#), utilizza i gestori di [MouseLeftButtonUp](#) classi per. Nelle sostituzioni, a cui vengono passati i dati dell'evento, le implementazioni [RoutedEventArgs](#) contrassegnano l'istanza come [Handled](#) gestita [true](#) impostando su e gli stessi dati di evento continuano lungo il resto della route ad altri gestori di classi e anche ai gestori di istanze o ai setter di eventi. Inoltre, l' [OnMouseLeftButtonUp](#) override di genererà l' [Click](#) evento. Il risultato finale per la maggior parte dei listener sarà che [MouseLeftButtonDown](#) gli [MouseLeftButtonUp](#) eventi e "scompaiono" e vengono sostituiti [Click](#) da, un evento che ha un significato maggiore perché è noto che questo evento ha avuto origine da un pulsante true e non da alcuna parte composta del pulsante o da un altro elemento interamente.

Soluzioni alternative all'eliminazione di eventi da parte dei controlli

A volte questo comportamento di eliminazione di eventi all'interno di singoli controlli può interferire con alcune intenzioni più generali della logica di gestione degli eventi per l'applicazione. Ad esempio, se per qualche motivo l'applicazione dispone di un gestore [MouseLeftButtonDown](#) per individuato nell'elemento radice dell'applicazione, si noterà che qualsiasi clic del mouse su un pulsante non [MouseLeftButtonDown](#) richiama [MouseLeftButtonUp](#) o gestori a livello radice. L'evento stesso è stato effettivamente propagato. Come già detto, le route degli eventi non vengono davvero completate, ma il sistema degli eventi indirizzati ne modifica il comportamento di chiamata del gestore dopo che gli eventi sono stati contrassegnati come gestiti. Quando l'evento indirizzato ha raggiunto il pulsante, [ButtonBase](#) la gestione della classe [MouseLeftButtonDown](#) contrassegnata come gestita perché desiderava sostituire [Click](#) l'evento con un significato maggiore. Pertanto, non viene [MouseLeftButtonDown](#) richiamato alcun gestore standard più alto della route. Esistono due tecniche che è possibile usare per garantire che i gestori vengano richiamati in questo caso.

La prima tecnica consiste nell'aggiungere intenzionalmente il gestore usando la [handledEventsToo](#) firma di [AddHandler\(RoutedEvent, Delegate, Boolean\)](#). Una limitazione di questo approccio è che la tecnica di collegamento di un gestore eventi è possibile solo dal codice, non dal markup. La semplice sintassi per specificare il nome del gestore eventi come valore di attributo dell'evento tramite Extensible Application Markup Language (XAML) non permette questo comportamento.

La seconda tecnica può essere usata solo per gli eventi di input, le cui versioni di tunneling e bubbling dell'evento indirizzato sono associate. Per questi eventi indirizzati, è invece possibile aggiungere gestori all'evento indirizzato di anteprima/tunneling equivalente. Poiché l'evento indirizzato percorre tramite tunneling la route a partire dalla radice, il codice di gestione delle classi del pulsante non lo intercetta, presumendo che il gestore di anteprima sia stato collegato a livello di un elemento predecessore nell'albero degli elementi dell'applicazione. Se si usa questo approccio, contrassegnare con cautela qualsiasi evento di anteprima come gestito. Per l'esempio fornito con [PreviewMouseLeftButtonDown](#) la gestione nell'elemento radice, se l'evento è stato contrassegnato come [Handled](#) nell'implementazione del gestore, si eliminerà effettivamente l' [Click](#) evento. Questo non è un comportamento consigliato.

Vedere anche

- [EventManager](#)
- [Eventi di anteprima](#)
- [Creare un evento indirizzato personalizzato](#)

- Cenni preliminari sugli eventi indirizzati

Eventi di anteprima

23/10/2019 • 6 minutes to read • [Edit Online](#)

Eventi di anteprima, noti anche come eventi di tunneling, sono eventi indirizzati in cui la direzione della route viene trasferito dalla radice dell'applicazione verso l'elemento che ha generato l'evento e viene indicato come origine di dati dell'evento. Non tutti gli scenari di evento supportano o richiedono eventi di anteprima. Questo argomento descrive le situazioni in cui gli eventi di anteprima sono presenti, come le applicazioni o componenti devono essere gestiti, e i casi in cui la creazione di eventi di anteprima in componenti personalizzati o classi potrebbe essere appropriata.

Input e gli eventi di anteprima

Quando si gestisce gli eventi in generale, è necessario prestare attenzione all'anteprima contrassegnare gli eventi gestiti nell'evento dati. Gestisce un evento di anteprima in qualsiasi elemento diverso dall'elemento che lo ha generato (l'elemento che viene segnalato come origine nei dati dell'evento) ha l'effetto di non fornire la possibilità di gestire l'evento che ha avuto origine un elemento. In alcuni casi questo è il risultato desiderato, in particolare se sono presenti gli elementi in questione nelle relazioni all'interno della composizione di un controllo.

Per gli eventi di input in particolare, gli eventi di anteprima anche condividono le istanze dei dati di evento con l'evento di bubbling equivalente. Se si usa un gestore di classi di eventi di anteprima per contrassegnare l'evento di input gestita, il gestore di classi di evento di input bubbling non essere richiamato. In alternativa, se si usa un gestore di istanze evento anteprima per contrassegnare l'evento come gestito, i gestori per l'evento di bubbling non saranno in genere essere richiamati. I gestori classi o gestori istanze possono essere registrati o collegati con la possibilità di essere richiamati anche se l'evento è contrassegnato come gestito, ma tale tecnica non viene usata comunemente.

Per altre informazioni sulla gestione delle classi e la sua relazione con eventi di anteprima, vedere [contrassegno degli eventi indirizzati come gestiti e gestione delle classi](#).

Soluzioni alternative all'eliminazione di eventi da parte dei controlli

Uno scenario in cui gli eventi di anteprima vengono comunemente usati è per la gestione di controllo composito di eventi di input. In alcuni casi, l'autore del controllo evita la visualizzazione di un determinato evento originato dai loro controllo, ad esempio per sostituire un evento definito dal componente che contiene più informazioni o implica un comportamento più specifico. Ad esempio, un Windows Presentation Foundation (WPF) [Button](#) evita [MouseLeftButtonDown](#) e [MouseRightButtonDown](#) bubbling gli eventi generati dal [Button](#) o sui suoi elementi compositi a favore lo stato mouse capture e la generazione di un [Click](#) evento che viene sempre generato dal [Button](#) stesso. L'evento e i relativi dati comunque continuare lungo la route, tuttavia, poiché il [Button](#) contrassegna i dati dell'evento come [Handled](#), solo i gestori per l'evento che espressamente indicato agiscono nel [handledEventsToo](#) case vengono richiamati. Se gli altri elementi verso la radice dell'applicazione voleva comunque la possibilità di gestire un evento eliminato di controllo, in alternativa è possibile associare i gestori nel codice con [handledEventsToo](#) specificato come [true](#). Ma spesso una tecnica più semplice consiste nel modificare la direzione di routing è gestire per l'equivalente di anteprima di un evento di input. Ad esempio, se un controllo sopprime [MouseLeftButtonDown](#), provare a collegare un gestore per [PreviewMouseLeftButtonDown](#) invece. Questa tecnica funziona solo per gli eventi di input di elementi di base, ad esempio [MouseLeftButtonDown](#). Questi eventi di input usano coppie di tunneling/bubbling, generano entrambi gli eventi e condividono i dati dell'evento.

Ognuna di queste tecniche ha effetti collaterali o le limitazioni. L'effetto collaterale di gestione dell'evento di anteprima è che la gestione dell'evento a questo punto può disattivare i gestori che prevedono di gestire l'evento di bubbling e pertanto la limitazione è che in genere non è una buona idea per contrassegnare l'evento come gestito mentre è ancora attivata la Previ parte uova della route. La limitazione del [handledEventsToo](#) tecnica è che

non è possibile specificare un `handledEventsToo` gestore in XAML come un attributo, è necessario registrare il gestore dell'evento nel codice dopo aver ottenuto un riferimento all'oggetto per l'elemento in cui il gestore di è da collegare.

Vedere anche

- [Contrassegno degli eventi indirizzati come gestiti e gestione delle classi](#)
- [Cenni preliminari sugli eventi indirizzati](#)

Eventi di modifica delle proprietà

04/11/2019 • 11 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) definisce diversi eventi generati in risposta a una modifica nel valore di una proprietà. Spesso la proprietà è una proprietà di dipendenza. L'evento stesso è talvolta un evento indirizzato ed è talvolta un evento standard Common Language Runtime (CLR). La definizione dell'evento varia in base allo scenario, poiché alcune modifiche delle proprietà sono inviate in modo più appropriato tramite un albero di elementi, mentre altre in genere sono di interesse solo per l'oggetto in cui la proprietà è stata modificata.

Identificazione di un evento di modifica delle proprietà

Non tutti gli eventi che segnalano una modifica delle proprietà sono identificati in modo esplicito come eventi di modifica delle proprietà, per mezzo di un pattern di firma o di nome. In genere, la descrizione dell'evento nella documentazione dell'SDK indica se l'evento è associato direttamente alla modifica del valore di una proprietà e fornisce riferimenti incrociati tra la proprietà e l'evento.

Eventi RoutedPropertyChanged

Alcuni eventi usano un tipo di dati di evento e un delegato che vengono usati in modo esplicito per gli eventi di modifica delle proprietà. Il tipo di dati dell'evento è [RoutedPropertyChangedEventArgs<T>](#) e il delegato è [RoutedPropertyChangedEventHandler<T>](#). I dati di evento e il delegato dispongono entrambi di un parametro di tipo generico, usato per specificare il tipo effettivo della proprietà modificata quando si definisce il gestore. I dati dell'evento contengono due proprietà, [OldValue](#) e [NewValue](#), che vengono entrambe passate come argomento di tipo nei dati dell'evento.

La parte "Routed" del nome indica che l'evento di modifica proprietà è registrato come evento indirizzato. Il vantaggio che si ottiene nell'indirizzare un evento di modifica proprietà è dato dal fatto che il livello superiore di un controllo può ricevere eventi di modifica proprietà se vengono modificati i valori delle proprietà sugli elementi figlio (parti composite del controllo). Ad esempio, è possibile creare un controllo che incorpora un controllo [RangeBase](#), ad esempio una [Slider](#). Se il valore della proprietà [Value](#) viene modificato sulla parte del dispositivo di scorrimento, potrebbe essere necessario gestire tale modifica nel controllo padre anziché nella parte.

Poiché si hanno un valore vecchio e un valore nuovo, può sembrare opportuno usare questo gestore eventi come validator per il valore della proprietà. Quest'uso, tuttavia, non rientra negli intenti di progettazione della maggior parte degli eventi di modifica proprietà. In genere i valori vengono forniti per consentire di eseguire azioni su di essi in altre aree logiche del codice, ma la modifica dei valori dall'interno del gestore eventi non è consigliabile e può causare ricorsione non intenzionale a seconda della modalità di implementazione del gestore.

Se la proprietà è una proprietà di dipendenza personalizzata o se si utilizza una classe derivata in cui è stato definito il codice di creazione dell'istanza, è disponibile un meccanismo molto migliore per tenere traccia delle modifiche delle proprietà compilate nel sistema di proprietà WPF: la proprietà callback di sistema [CoerceValueCallback](#) e [PropertyChangedCallback](#). Per informazioni dettagliate su come usare il sistema di proprietà WPF per la convalida e la coercizione, vedere [Callback e convalida delle proprietà di dipendenza](#) e [Proprietà di dipendenza personalizzate](#).

Eventi DependencyPropertyChanged

Un'altra coppia di tipi che fanno parte di uno scenario di evento modificato da proprietà è [DependencyPropertyChangedEventArgs](#) e [DependencyPropertyChangedEventHandler](#). Gli eventi per queste modifiche delle proprietà non vengono indirizzati; si tratta di eventi CLR standard.

[DependencyPropertyChangedEventArgs](#) è un tipo di segnalazione dati evento insolito perché non deriva da [EventArgs](#); [DependencyPropertyChangedEventArgs](#) è una struttura e non una classe.

Gli eventi che usano `DependencyPropertyChangedEventArgs` e `DependencyPropertyChangedEventHandler` sono leggermente più comuni degli eventi di `RoutedPropertyChanged`. Un esempio di evento che usa questi tipi è `IsMouseCapturedChanged`.

Come `RoutedPropertyChangedEventArgs<T>`, `DependencyPropertyChangedEventArgs` segnala anche un valore precedente e nuovo per la proprietà. Inoltre, a questo evento si applicano le stesse considerazioni relative all'uso dei valori. In genere è sconsigliabile provare a modificare nuovamente i valori sul mittente in risposta all'evento.

Trigger di proprietà

Un concetto correlato strettamente a un evento di modifica proprietà è quello di trigger di proprietà. Un trigger di proprietà viene creato all'interno di uno stile o di un modello e consente di creare un comportamento condizionale basato sul valore della proprietà a cui il trigger è assegnato.

La proprietà per un trigger di proprietà deve essere una proprietà di dipendenza. Può trattarsi, come spesso accade, di una proprietà di dipendenza di sola lettura. Un buon indicatore del fatto che una proprietà di dipendenza esposta da un controllo sia progettata almeno parzialmente come trigger di proprietà è la presenza del termine "Is" nel nome della proprietà. Le proprietà con questo nome sono spesso proprietà di dipendenza booleane di sola lettura il cui scenario primario è la segnalazione dello stato di un controllo che può avere conseguenze sull'interfaccia utente in tempo reale, pertanto sono un buon candidato ad essere trigger di proprietà.

Alcune di queste proprietà hanno anche un evento di modifica proprietà dedicato. Ad esempio, la proprietà `IsMouseCaptured` dispone di un evento di modifica proprietà `IsMouseCapturedChanged`. La proprietà stessa è di sola lettura, con il relativo valore regolato dal sistema di input e il sistema di input genera `IsMouseCapturedChanged` per ogni modifica in tempo reale.

Rispetto a un vero evento di modifica proprietà, l'uso di un trigger per gestire una modifica di proprietà presenta alcune limitazioni.

I trigger di proprietà funzionano in base a una logica di corrispondenza esatta. Specificare una proprietà e un valore che indichi il valore specifico per il quale agirà il trigger. Ad esempio:

`<Setter Property="IsMouseCaptured" Value="true"> ... </Setter>`. A causa di questa limitazione, nella maggior parte dei casi i trigger di proprietà vengono usati per proprietà booleane, o proprietà che accettano un valore di enumerazione dedicato, in cui l'intervallo di valori possibili è sufficientemente gestibile da consentire la definizione di un trigger per ogni caso. In alternativa, potrebbero esistere trigger di proprietà solo per valori speciali, ad esempio nel caso che un conteggio di elementi raggiunga lo zero, senza trigger che tengano conto dei casi in cui il valore della proprietà viene di nuovo modificato in un numero diverso da zero (anziché disporre di trigger per tutti i casi, in questo scenario potrebbe essere necessario un gestore eventi di codice o un comportamento predefinito per ripristinare lo stato del trigger quando il valore è diverso da zero).

La sintassi del trigger di proprietà è analoga a un'istruzione "if" nella programmazione. Se la condizione del trigger è true, viene eseguito il corpo del trigger di proprietà. Il corpo di un trigger di proprietà non è codice, ma markup. Il markup è limitato all'uso di uno o più elementi di `Setter` per impostare altre proprietà dell'oggetto in cui viene applicato lo stile o il modello.

Per compensare la condizione "if" di un trigger di proprietà con un'ampia gamma di valori possibili, è in genere consigliabile impostare lo stesso valore di proprietà su un valore predefinito utilizzando un `Setter`. In questo modo, il Setter contenuto `Trigger` avrà la precedenza quando la condizione del trigger è true e il `Setter` che non si trova all'interno di un `Trigger` avrà la precedenza ogni volta che la condizione del trigger è false.

I trigger di proprietà sono generalmente adatti negli scenari in cui una o più proprietà di aspetto devono essere modificate, in base allo stato di un'altra proprietà sullo stesso elemento.

Per altre informazioni sui trigger di proprietà, vedere [Applicazione di stili e modelli](#).

Vedere anche

- Cenni preliminari sugli eventi indirizzati
- Panoramica sulle proprietà di dipendenza

Visual Basic e la gestione degli eventi WPF

03/02/2020 • 7 minutes to read • [Edit Online](#)

Per il linguaggio Microsoft Visual Basic .NET in particolare, è possibile usare la parola chiave `Handles` specifica del linguaggio per associare i gestori eventi alle istanze, anziché collegare i gestori eventi agli attributi o usare il metodo `AddHandler`. Tuttavia, la tecnica `Handles` per collegare gestori alle istanze presenta alcuni limiti, in quanto la sintassi `Handles` non è in grado di supportare alcune delle funzioni degli eventi indirizzati specifiche del sistema di eventi WPF.

Uso di "Handles" in un'applicazione WPF

I gestori eventi connessi a istanze ed eventi tramite `Handles` devono essere definiti tutti all'interno della dichiarazione di classe parziale dell'istanza. Ciò vale anche per i gestori eventi assegnati tramite valori di attributo negli elementi. È possibile specificare solo `Handles` per un elemento nella pagina che ha un valore di proprietà `Name` (o [direttiva x:Name dichiarata](#)). Ciò è dovuto al fatto che il `Name` in XAML crea il riferimento all'istanza necessario per supportare il formato di riferimento dell'*istanza.evento* richiesto dalla sintassi `Handles`. L'unico elemento che può essere utilizzato per `Handles` senza un riferimento `Name` è l'istanza dell'elemento radice che definisce la classe parziale.

È possibile assegnare lo stesso gestore a più elementi separando i riferimenti *Istanza.Evento* dopo `Handles` con delle virgole.

È possibile usare `Handles` per assegnare più gestori allo stesso riferimento *Istanza.Evento*. Non attribuire alcuna importanza all'ordine in cui vengono forniti i gestori nel riferimento `Handles`. È necessario presupporre che gestori dello stesso evento possono essere richiamati in qualsiasi ordine.

Per rimuovere un gestore aggiunto con `Handles` nella dichiarazione, è possibile chiamare [RemoveHandler](#).

È possibile usare `Handles` per collegare i gestori per gli eventi indirizzati, purché i gestori vengano collegati a istanze che definiscono l'evento gestito nelle rispettive tabelle di membri. Per gli eventi indirizzati, i gestori collegati con `Handles` seguono le stesse regole di routing dei gestori collegati come XAML attributi o con la firma comune di `AddHandler`. Ciò significa che se l'evento è già contrassegnato come gestito (la proprietà `Handled` nei dati dell'evento viene `True`), i gestori collegati con `Handles` non vengono richiamati in risposta a tale istanza di evento. L'evento potrebbe essere contrassegnato come gestito dai gestori dell'istanza in un altro elemento della route o dalla gestione delle classi nell'elemento corrente o negli elementi precedenti della route. Per gli eventi di input che supportano eventi accoppiati di tunneling e di bubbling, è possibile che la route di tunneling abbia contrassegnato la coppia di eventi come gestita. Per altre informazioni sugli eventi indirizzati, vedere [Cenni preliminari sugli eventi indirizzati](#).

Limitazioni di "Handle" per l'aggiunta di gestori

`Handles` non può fare riferimento ai gestori per gli eventi associati. È necessario usare il metodo della funzione di accesso `add` per quello specifico evento associato oppure gli attributi dell'evento `nometipo.nomeevento` in XAML. Per informazioni dettagliate, vedere [Cenni preliminari sugli eventi indirizzati](#).

Per gli eventi indirizzati, è possibile usare `Handles` unicamente per assegnare gestori per le istanze in cui quell'evento è presente nella tabella dei membri dell'istanza. Tuttavia, con gli eventi indirizzati in generale, un elemento padre può essere un listener per un evento degli elementi figlio anche se l'elemento padre non dispone di quell'evento nella relativa tabella dei membri. Nella sintassi dell'attributo, ciò può essere specificato tramite un formato di attributo `nometipo.nomemembro` che qualifica il tipo che definisce effettivamente l'evento che si vuole

gestire. Ad esempio, un `Page` padre (senza `click` evento definito) può restare in ascolto degli eventi `click` del pulsante assegnando un gestore attributi nel formato `Button.Click`. Tuttavia, `Handles` non supporta il formato `nometipo.nomemembro`, in quanto deve supportare un formato `Istanza.Evento` in conflitto. Per informazioni dettagliate, vedere [Cenni preliminari sugli eventi indirizzati](#).

`Handles` non è in grado di associare gestori richiamati per eventi che sono già contrassegnati come gestiti. Al contrario, è necessario usare il codice e chiamare l'overload `handledEventsToo` di `AddHandler(RoutedEvent, Delegate, Boolean)`.

NOTE

Non usare la sintassi `Handles` nel codice Visual Basic quando si specifica un gestore eventi per lo stesso evento in XAML. In questo caso, il gestore eventi viene chiamato due volte.

Come WPF implementa la funzionalità "Handles"

Quando viene compilata una pagina di Extensible Application Markup Language (XAML), il file intermedio dichiara `Friend WithEvents` riferimenti a ogni elemento della pagina che dispone di una `Name` set di proprietà (o [direttiva `x:Name`](#) dichiarata). Ogni istanza denominata è potenzialmente un elemento che può essere assegnato a un gestore tramite `Handles`.

NOTE

In Visual Studio, IntelliSense consente di visualizzare il completamento per cui sono disponibili elementi per un riferimento `Handles` in una pagina. Tuttavia, tale operazione potrebbe richiedere un passaggio di compilazione che consenta al file intermedio di popolare tutti i riferimenti `Friends`.

Vedere anche

- [AddHandler](#)
- [Contrassegno degli eventi indirizzati come gestiti e gestione delle classi](#)
- [Cenni preliminari sugli eventi indirizzati](#)
- [Panoramica di XAML \(WPF\)](#)

Modelli di eventi deboli

31/01/2020 • 12 minutes to read • [Edit Online](#)

Nelle applicazioni è possibile che i gestori collegati alle origini eventi non vengano eliminati in modo coordinato con l'oggetto listener che ha collegato il gestore all'origine. Questa situazione può causare perdite di memoria. Windows Presentation Foundation (WPF) introduce uno schema progettuale che può essere usato per risolvere questo problema, fornendo una classe Manager dedicata per determinati eventi e implementando un'interfaccia nei listener per tale evento. Questo schema progettuale è noto come *modello di eventi deboli*.

Perché implementare il modello di eventi deboli?

L'ascolto degli eventi può causare perdite di memoria. La tecnica tipica per l'ascolto di un evento consiste nell'usare la sintassi specifica del linguaggio che connette un gestore a un evento in un'origine. In C#, ad esempio, la sintassi è: `source.SomeEvent += new SomeEventHandler(MyEventHandler)` .

Questa tecnica crea un riferimento sicuro dall'origine evento al listener di eventi. In genere, il fissaggio di un gestore eventi per un listener fa sì che il listener abbia una durata degli oggetti influenzato dalla durata dell'oggetto dell'origine (a meno che il gestore eventi non venga rimosso in modo esplicito). In alcuni casi, tuttavia, potrebbe essere necessario controllare la durata dell'oggetto del listener da altri fattori, ad esempio se appartiene attualmente alla struttura ad albero visuale dell'applicazione e non alla durata dell'origine. Ogni volta che la durata dell'oggetto di origine supera la durata dell'oggetto del listener, il modello di evento normale causa una perdita di memoria: il listener viene mantenuto attivo più a lungo del previsto.

Il modello di eventi vulnerabili è progettato per risolvere questo problema di perdita di memoria. Il modello di eventi vulnerabili può essere usato ogni volta che un listener deve registrarsi per un evento, ma il listener non sa in modo esplicito quando annullare la registrazione. Il modello di eventi vulnerabili può essere utilizzato anche ogni volta che la durata dell'oggetto dell'origine supera la durata utile dell'oggetto del listener. (In questo caso, l'*utilità* è determinata dall'utente). Il modello di eventi vulnerabili consente al listener di registrarsi e ricevere l'evento senza influire in alcun modo sulle caratteristiche di durata degli oggetti del listener. In effetti, il riferimento隐式的 dall'origine non determina se il listener è idoneo per Garbage Collection. Il riferimento è un riferimento debole, quindi la denominazione del modello di eventi deboli e le API correlate. Il listener può essere sottoposta a Garbage Collection o eliminato in altro modo e l'origine può continuare senza mantenere i riferimenti al gestore non ritirabili a un oggetto distrutto.

Chi deve implementare il modello di eventi deboli?

L'implementazione del modello di eventi vulnerabili è particolarmente interessante per gli autori di controlli. In qualità di autore del controllo, l'utente è in gran parte responsabile del comportamento e del contenimento del controllo e dell'incidenza sulle applicazioni in cui viene inserito. Questo include il comportamento di controllo della durata degli oggetti, in particolare la gestione del problema di perdita di memoria descritto.

Determinati scenari si prestano intrinsecamente all'applicazione del modello di eventi deboli. Uno scenario di questo tipo è data binding. In data binding, è normale che l'oggetto di origine sia completamente indipendente dall'oggetto listener, che è la destinazione di un'associazione. Molti aspetti di WPF data binding hanno già il modello di eventi deboli applicato nella modalità di implementazione degli eventi.

Come implementare il modello di eventi deboli

Esistono tre modi per implementare il modello di eventi deboli. Nella tabella seguente sono elencati i tre approcci e vengono fornite alcune indicazioni per l'utilizzo di ciascuno di essi.

APPROCCIO	QUANDO IMPLEMENTARE
Usa una classe di gestione degli eventi vulnerabili esistente	Se l'evento a cui si desidera effettuare la sottoscrizione ha un WeakEventManager corrispondente, utilizzare il gestore eventi vulnerabile esistente. Per un elenco di gestori di eventi vulnerabili inclusi in WPF, vedere la gerarchia di ereditarietà nella classe WeakEventManager . Poiché i gestori di eventi deboli inclusi sono limitati, probabilmente sarà necessario scegliere uno degli altri approcci.
Usare una classe generica di gestione eventi debole	Usare un WeakEventManager<TEventArgs> generico quando un WeakEventManager esistente non è disponibile, si vuole un modo semplice per implementare e non si è interessati all'efficienza. Il WeakEventManager<TEventArgs> generico è meno efficiente rispetto a un gestore di eventi vulnerabile esistente o personalizzato. La classe generica, ad esempio, esegue più reflection per individuare l'evento in base al nome dell'evento. Inoltre, il codice per registrare l'evento usando il WeakEventManager<TEventArgs> generico è più dettagliato rispetto all'uso di un WeakEventManager personalizzato o esistente.
Creare una classe personalizzata di gestione eventi vulnerabili	Creare un WeakEventManager personalizzato quando un WeakEventManager esistente non è disponibile e si desidera ottenere la migliore efficienza. L'uso di un WeakEventManager personalizzato per sottoscrivere un evento è più efficiente, ma comporta il costo di scrivere altro codice all'inizio.
Usare un gestore eventi vulnerabile di terze parti	NuGet dispone di diversi gestori di eventi vulnerabili e molti framework WPF supportano anche il modello (ad esempio, vedere la documentazione di Prisma sulla sottoscrizione di eventi a regime di controllo libero).

Le sezioni seguenti descrivono come implementare il modello di eventi deboli. Ai fini di questa discussione, l'evento da sottoscrivere presenta le caratteristiche seguenti.

- Il nome dell'evento è `SomeEvent`.
- L'evento viene generato dalla classe `EventSource`.
- Il gestore eventi è di tipo: `SomeEventEventHandler` o `EventHandler<SomeEventArgs>`.
- L'evento passa un parametro di tipo `SomeEventArgs` ai gestori eventi.

Utilizzo di una classe di gestione eventi debole esistente

1. Trovare un gestore eventi vulnerabile esistente.

Per un elenco di gestori di eventi vulnerabili inclusi in WPF, vedere la gerarchia di ereditarietà nella classe [WeakEventManager](#).

2. Utilizzare il nuovo gestore eventi debole anziché il normale collegamento evento.

Ad esempio, se il codice usa il modello seguente per sottoscrivere un evento:

```
source.SomeEvent += new SomeEventEventHandler(OnSomeEvent);
```

Modificare il modello nel modo seguente:

```
SomeEventWeakEventManager.AddHandler(source, OnSomeEvent);
```

Analogamente, se il codice usa il modello seguente per annullare la sottoscrizione di un evento:

```
source.SomeEvent -= new SomeEventEventHandler(OnSomeEvent);
```

Modificare il modello nel modo seguente:

```
SomeEventWeakEventManager.RemoveHandler(source, OnSomeEvent);
```

Uso della classe generica di gestione eventi debole

1. Usare la classe [WeakEventManager<TEventArgs>](#) generica anziché il normale collegamento evento.

Quando si usa [WeakEventManager<TEventArgs>](#) per registrare i listener di eventi, fornire l'origine evento e il tipo di [EventArgs](#) come parametri di tipo alla classe e chiamare [AddHandler](#) come illustrato nel codice seguente:

```
WeakEventManager<EventSource, SomeEventArgs>.AddHandler(source, "SomeEvent", source_SomeEvent);
```

Creazione di una classe personalizzata di gestione eventi vulnerabili

1. Copiare il modello di classe seguente nel progetto.

Questa classe eredita dalla classe [WeakEventManager](#).

```
class SomeEventWeakEventManager : WeakEventManager
{
    private SomeEventWeakEventManager()
    {
    }

    /// <summary>
    /// Add a handler for the given source's event.
    /// </summary>
    public static void AddHandler(EventSource source,
                                  EventHandler<SomeEventArgs> handler)
    {
        if (source == null)
            throw new ArgumentNullException("source");
        if (handler == null)
            throw new ArgumentNullException("handler");

        CurrentManager.ProtectedAddHandler(source, handler);
    }

    /// <summary>
    /// Remove a handler for the given source's event.
    /// </summary>
    public static void RemoveHandler(EventSource source,
                                    EventHandler<SomeEventArgs> handler)
    {
        if (source == null)
            throw new ArgumentNullException("source");
        if (handler == null)
            throw new ArgumentNullException("handler");

        CurrentManager.ProtectedRemoveHandler(source, handler);
    }
}
```

```

}

/// <summary>
/// Get the event manager for the current thread.
/// </summary>
private static SomeEventWeakEventManager CurrentManager
{
    get
    {
        Type managerType = typeof(SomeEventWeakEventManager);
        SomeEventWeakEventManager manager =
            (SomeEventWeakEventManager)GetCurrentManager(managerType);

        // at first use, create and register a new manager
        if (manager == null)
        {
            manager = new SomeEventWeakEventManager();
            SetCurrentManager(managerType, manager);
        }

        return manager;
    }
}

/// <summary>
/// Return a new list to hold listeners to the event.
/// </summary>
protected override ListenerList NewListenerList()
{
    return new ListenerList<SomeEventEventArgs>();
}

/// <summary>
/// Listen to the given source for the event.
/// </summary>
protected override void StartListening(object source)
{
    EventSource typedSource = (EventSource)source;
    typedSource.SomeEvent += new EventHandler<SomeEventEventArgs>(OnSomeEvent);
}

/// <summary>
/// Stop listening to the given source for the event.
/// </summary>
protected override void StopListening(object source)
{
    EventSource typedSource = (EventSource)source;
    typedSource.SomeEvent -= new EventHandler<SomeEventEventArgs>(OnSomeEvent);
}

/// <summary>
/// Event handler for the SomeEvent event.
/// </summary>
void OnSomeEvent(object sender, SomeEventEventArgs e)
{
    DeliverEvent(sender, e);
}
}

```

2. Sostituire il nome del `SomeEventWeakEventManager` con il proprio nome.
3. Sostituire i tre nomi descritti in precedenza con i nomi corrispondenti per l'evento. (`SomeEvent`, `EventSource` e `SomeEventEventArgs`)
4. Impostare la visibilità (pubblica/interna/privata) della classe del gestore eventi debole sulla stessa visibilità dell'evento gestito.

5. Utilizzare il nuovo gestore eventi debole anziché il normale collegamento evento.

Ad esempio, se il codice usa il modello seguente per sottoscrivere un evento:

```
source.SomeEvent += new SomeEventEventHandler(OnSomeEvent);
```

Modificare il modello nel modo seguente:

```
SomeEventWeakEventManager.AddHandler(source, OnSomeEvent);
```

Analogamente, se il codice usa il modello seguente per annullare la sottoscrizione di un evento:

```
source.SomeEvent -= new SomeEventEventHandler(OnSomeEvent);
```

Modificare il modello nel modo seguente:

```
SomeEventWeakEventManager.RemoveHandler(source, OnSomeEvent);
```

Vedere anche

- [WeakEventManager](#)
- [IWeakEventListener](#)
- [Cenni preliminari sugli eventi indirizzati](#)
- [Cenni preliminari sull'associazione dati](#)

Procedure relative agli eventi

23/10/2019 • 2 minutes to read • [Edit Online](#)

Gli argomenti in questa sezione descrivono come usare gli eventi in WPF.

In questa sezione

[Aggiungere un gestore eventi usando il codice](#)

[Gestire un evento indirizzato](#)

[Creare un evento indirizzato personalizzato](#)

[Cercare l'elemento di origine in un gestore eventi](#)

[Aggiungere la gestione di classi per un evento indirizzato](#)

Riferimenti

[RoutedEventArgs](#)

[EventManager](#)

[RoutingStrategy](#)

Sezioni correlate

Procedura: aggiungere un gestore eventi mediante codice

04/11/2019 • 3 minutes to read • [Edit Online](#)

Questo esempio illustra come aggiungere un gestore eventi a un elemento usando il codice.

Se si desidera aggiungere un gestore eventi a un elemento XAML e la pagina di markup che contiene l'elemento è già stata caricata, è necessario aggiungere il gestore utilizzando il codice. In alternativa, se si sta compilando la struttura ad albero dell'elemento per un'applicazione usando interamente il codice e non dichiarando alcun elemento usando XAML, è possibile chiamare metodi specifici per aggiungere gestori eventi all'albero degli elementi costruito.

Esempio

Nell'esempio seguente viene aggiunto un nuovo [Button](#) a una pagina esistente inizialmente definita in XAML. Un file code-behind implementa un metodo del gestore eventi e quindi aggiunge tale metodo come nuovo gestore eventi nel [Button](#).

Nell' C# esempio viene utilizzato l'operatore `+=` per assegnare un gestore a un evento. Si tratta dello stesso operatore utilizzato per assegnare un gestore nel modello di gestione degli eventi Common Language Runtime (CLR). Microsoft Visual Basic non supporta questo operatore come mezzo per l'aggiunta di gestori eventi.

Richiede invece una delle due tecniche seguenti:

- Usare il metodo [AddHandler](#) insieme a un operatore `Addressof` per fare riferimento all'implementazione del gestore eventi.
- Usare la parola chiave `Handles` come parte della definizione del gestore eventi. Questa tecnica non è illustrata di seguito. vedere [Visual Basic e gestione degli eventi WPF](#).

```
<TextBlock Name="text1">Start by clicking the button below</TextBlock>
<Button Name="b1" Click="MakeButton">Make new button and add handler to it</Button>
```

```
public partial class RoutedEventAddRemoveHandler {
    void MakeButton(object sender, RoutedEventArgs e)
    {
        Button b2 = new Button();
        b2.Content = "New Button";
        // Associate event handler to the button. You can remove the event
        // handler using "=-" syntax rather than "+=".
        b2.Click += new RoutedEventHandler(Onb2Click);
        root.Children.Insert(root.Children.Count, b2);
        DockPanel.SetDock(b2, Dock.Top);
        text1.Text = "Now click the second button...";
        b1.IsEnabled = false;
    }
    void Onb2Click(object sender, RoutedEventArgs e)
    {
        text1.Text = "New Button (b2) Was Clicked!!";
    }
}
```

```
Public Partial Class RoutedEventAddRemoveHandler
    Private Sub MakeButton(ByVal sender As Object, ByVal e As RoutedEventArgs)
        Dim b2 As Button = New Button()
        b2.Content = "New Button"
        AddHandler b2.Click, AddressOf Onb2Click
        root.Children.Insert(root.Children.Count, b2)
        DockPanel.SetDock(b2, Dock.Top)
        text1.Text = "Now click the second button..."
        b1.IsEnabled = False
    End Sub
    Private Sub Onb2Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
        text1.Text = "New Button (b2) Was Clicked!!"
    End Sub
```

NOTE

L'aggiunta di un gestore eventi nella pagina XAML analizzata inizialmente è molto più semplice. All'interno dell'elemento oggetto in cui si desidera aggiungere il gestore eventi, aggiungere un attributo che corrisponda al nome dell'evento che si desidera gestire. Specificare quindi il valore di tale attributo come nome del metodo del gestore eventi definito nel file code-behind della pagina XAML. Per altre informazioni, vedere Cenni [preliminari su XAML \(WPF\)](#) o [Cenni preliminari sugli eventi indirizzati](#).

Vedere anche

- [Cenni preliminari sugli eventi indirizzati](#)
- [Procedure relative alle proprietà](#)

Procedura: Gestire un evento indirizzato

23/10/2019 • 4 minutes to read • [Edit Online](#)

Questo esempio illustra il funzionamento degli eventi di bubbling e spiega come scrivere un gestore in grado di elaborare i dati dell'evento indirizzato.

Esempio

In Windows Presentation Foundation (WPF) gli elementi sono disposti in una struttura ad albero degli elementi. L'elemento padre può partecipare alla gestione di eventi inizialmente generati dagli elementi figlio nell'albero degli elementi. Questo è possibile grazie al routing di eventi.

Per gli eventi indirizzati si usa in genere una delle due strategie di routing seguenti: bubbling o tunneling. Questo esempio illustra l'evento di bubbling e Usa il [ButtonBase.Click](#) evento per visualizzarne il funzionamento del routing.

L'esempio seguente crea due [Button](#) consente di controllare e viene utilizzato XAML la sintassi per associare un gestore eventi a un elemento padre comune, ovvero in questo esempio di attributo [StackPanel](#). Anziché associare singoli gestori eventi per ognuno [Button](#) elemento figlio, l'esempio Usa la sintassi degli attributi per associare il gestore eventi per il [StackPanel](#) elemento padre. Questo criterio di gestione degli eventi illustra in che modo usare il routing di eventi come tecnica per ridurre il numero di elementi a cui è associato un gestore. Tutti gli eventi di bubbling per ogni [Button](#) viene instradato attraverso l'elemento padre.

Si noti che nell'elemento padre [StackPanel](#) elemento, il [Click](#) nome dell'evento specificato, come l'attributo è parzialmente qualificato mediante la denominazione di [Button](#) classe. Il [Button](#) classe è una [ButtonBase](#) classe derivata che ha il [Click](#) evento nel relativo elenco dei membri. La tecnica della qualifica parziale per l'associazione di un gestore eventi è necessaria se l'evento che viene gestito non esiste nell'elenco dei membri dell'elemento a cui è associato il gestore dell'evento indirizzato.

```
<StackPanel  
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
    x:Class="SDKSample.RoutedEventHandle"  
    Name="dpanel1"  
    Button.Click="HandleClick">  
>  
<StackPanel.Resources>  
    <Style TargetType="{x:Type Button}">  
        <Setter Property="Height" Value="20"/>  
        <Setter Property="Width" Value="250"/>  
        <Setter Property="HorizontalAlignment" Value="Left"/>  
    </Style>  
</StackPanel.Resources>  
<Button Name="Button1">Item 1</Button>  
<Button Name="Button2">Item 2</Button>  
<TextBlock Name="results"/>  
</StackPanel>
```

L'esempio seguente viene gestito il [Click](#) evento. L'esempio stabilisce quale elemento gestisce l'evento e quale elemento genera l'evento. Il gestore eventi viene eseguito quando l'utente fa clic su uno dei pulsanti.

```

public partial class RoutedEventHandle : StackPanel
{
    StringBuilder eventstr = new StringBuilder();
    void HandleClick(object sender, RoutedEventArgs args)
    {
        // Get the element that handled the event.
        FrameworkElement fe = (FrameworkElement)sender;
        eventstr.Append("Event handled by element named ");
        eventstr.Append(fe.Name);
        eventstr.Append("\n");

        // Get the element that raised the event.
        FrameworkElement fe2 = (FrameworkElement)args.Source;
        eventstr.Append("Event originated from source element of type ");
        eventstr.Append(args.Source.GetType().ToString());
        eventstr.Append(" with Name ");
        eventstr.Append(fe2.Name);
        eventstr.Append("\n");

        // Get the routing strategy.
        eventstr.Append("Event used routing strategy ");
        eventstr.Append(args.RoutedEventArgs.RoutingStrategy);
        eventstr.Append("\n");

        results.Text = eventstr.ToString();
    }
}

```

```

Private eventstr As New Text.StringBuilder()

Private Sub HandleClick(ByVal sender As Object, ByVal args As RoutedEventArgs)
    ' Get the element that handled the event.
    Dim fe As FrameworkElement = DirectCast(sender, FrameworkElement)
    eventstr.Append("Event handled by element named ")
    eventstr.Append(fe.Name)
    eventstr.Append(vbLf)

    ' Get the element that raised the event.
    Dim fe2 As FrameworkElement = DirectCast(args.Source, FrameworkElement)
    eventstr.Append("Event originated from source element of type ")
    eventstr.Append(args.Source.[GetType]().ToString())
    eventstr.Append(" with Name ")
    eventstr.Append(fe2.Name)
    eventstr.Append(vbLf)

    ' Get the routing strategy.
    eventstr.Append("Event used routing strategy ")
    eventstr.Append(args.RoutedEventArgs.RoutingStrategy)
    eventstr.Append(vbLf)

    results.Text = eventstr.ToString()
End Sub

```

Vedere anche

- [RoutedEvent](#)
- [Cenni preliminari sull'input](#)
- [Cenni preliminari sugli eventi indirizzati](#)
- [Procedure relative alle proprietà](#)
- [Descrizione dettagliata della sintassi XAML](#)

Procedura: Creare un evento indirizzato personalizzato

23/10/2019 • 3 minutes to read • [Edit Online](#)

Affinché l'evento personalizzato supporti il routing degli eventi, è necessario registrare un [RoutedEvent](#) usando il [RegisterRoutedEvent](#) metodo. Questo esempio illustra le nozioni di base per la creazione di un evento indirizzato personalizzato.

Esempio

Come illustrato nell'esempio seguente, è necessario innanzitutto registrare un [RoutedEvent](#) oggetto usando [RegisterRoutedEvent](#) il metodo. Per convenzione, il [RoutedEvent](#) nome del campo statico deve terminare con l'**evento**suffisso. In questo esempio, il nome dell'evento è `Tap` e la strategia di routing dell'evento è [Bubble](#). Dopo la chiamata di registrazione, è possibile fornire funzioni di accesso agli eventi Common Language Runtime (CLR) Add-and-Remove per l'evento.

Si noti che anche se l'evento viene generato tramite il metodo virtuale `OnTap` in questo particolare esempio, la modalità di generazione dell'evento o la relativa risposta alle modifiche varierà in base alle esigenze.

Si noti inoltre che in questo esempio viene implementata essenzialmente [Button](#) un'intera sottoclasse di. tale sottoclasse viene compilata come assembly separato e quindi creata come Extensible Application Markup Language (XAML) classe personalizzata in una pagina separata. Ciò dimostra che i controlli sottoclassati possono essere inseriti in alberi composti da altri controlli e che, in una situazione del genere, gli eventi personalizzati in questi controlli dispongono delle stesse funzionalità di routing di qualsiasi elemento Windows Presentation Foundation (WPF) nativo.

```
public class MyButtonSimple: Button
{
    // Create a custom routed event by first registering a RoutedEventID
    // This event uses the bubbling routing strategy
    public static readonly RoutedEvent TapEvent = EventManager.RegisterRoutedEvent(
        "Tap", RoutingStrategy.Bubble, typeof(RoutedEventHandler), typeof(MyButtonSimple));

    // Provide CLR accessors for the event
    public event RoutedEventHandler Tap
    {
        add { AddHandler(TapEvent, value); }
        remove { RemoveHandler(TapEvent, value); }
    }

    // This method raises the Tap event
    void RaiseTapEvent()
    {
        RoutedEventArgs newEventArgs = new RoutedEventArgs(MyButtonSimple.TapEvent);
        RaiseEvent(newEventArgs);
    }
    // For demonstration purposes we raise the event when the MyButtonSimple is clicked
    protected override void OnClick()
    {
        RaiseTapEvent();
    }
}
```

```

Public Class MyButtonSimple
    Inherits Button

    ' Create a custom routed event by first registering a RoutedEventID
    ' This event uses the bubbling routing strategy
    Public Shared ReadOnly TapEvent As RoutedEvent = EventManager.RegisterRoutedEvent("Tap",
        RoutingStrategy.Bubble, GetType(RoutedEventHandler), GetType(MyButtonSimple))

    ' Provide CLR accessors for the event
    Public Custom Event Tap As RoutedEventHandler
        AddHandler(ByVal value As RoutedEventHandler)
            Me.AddHandler(TapEvent, value)
        End AddHandler

        RemoveHandler(ByVal value As RoutedEventHandler)
            Me.RemoveHandler(TapEvent, value)
        End RemoveHandler

        RaiseEvent(ByVal sender As Object, ByVal e As RoutedEventArgs)
            Me.RaiseEvent(e)
        End RaiseEvent
    End Event

    ' This method raises the Tap event
    Private Sub RaiseTapEvent()
        Dim newEventArgs As New RoutedEventArgs(MyButtonSimple.TapEvent)
        MyBase.RaiseEvent(newEventArgs)
    End Sub

    ' For demonstration purposes we raise the event when the MyButtonSimple is clicked
    Protected Overrides Sub OnClick()
        Me.RaiseTapEvent()
    End Sub
End Class

```

```

<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:custom="clr-namespace:SDKSample;assembly=SDKSampleLibrary"
    x:Class="SDKSample.RoutedEventCustomApp"

    >
    <Window.Resources>
        <Style TargetType="{x:Type custom:MyButtonSimple}">
            <Setter Property="Height" Value="20"/>
            <Setter Property="Width" Value="250"/>
            <Setter Property="HorizontalAlignment" Value="Left"/>
            <Setter Property="Background" Value="#808080"/>
        </Style>
    </Window.Resources>
    <StackPanel Background="LightGray">
        <custom:MyButtonSimple Name="mybtnsimple" Tap="TapHandler">Click to see Tap custom event
        work</custom:MyButtonSimple>
    </StackPanel>
</Window>

```

Gli eventi di tunneling vengono creati nello stesso modo, [RoutingStrategy](#) ma con [Tunnel](#) impostato su nella chiamata di registrazione. Per convenzione, gli eventi di tunneling in WPF vengono denominati con il prefisso "Preview".

Per un esempio di funzionamento degli eventi di bubbling, vedere [Gestire un evento indirizzato](#).

Vedere anche

- [Cenni preliminari sugli eventi indirizzati](#)
- [Cenni preliminari sull'input](#)
- [Cenni preliminari sulla modifica di controlli](#)

Procedura: Cercare l'elemento di origine in un gestore eventi

23/10/2019 • 2 minutes to read • [Edit Online](#)

In questo esempio viene illustrato come trovare l'elemento di origine in un gestore eventi.

Esempio

L'esempio seguente mostra un [Click](#) gestore eventi che è dichiarato in un file code-behind. Quando un utente fa clic sul pulsante con il gestore di è collegato a, il gestore di modifica un valore della proprietà. Usa il codice del gestore il [Source](#) proprietà dei dati dell'evento indirizzato che viene segnalati negli argomenti di evento per modificare il [Width](#) sul valore della proprietà di [Source](#) elemento.

```
<Button Click="HandleClick">Button 1</Button>
```

```
void HandleClick(object sender, RoutedEventArgs e)
{
    // You must cast the sender object as a Button element, or at least as FrameworkElement, to set Width
    Button srcButton = e.Source as Button;
    srcButton.Width = 200;
}
```

```
Private Sub HandleClick(ByVal sender As Object, ByVal e As RoutedEventArgs)
    'You must cast the object as a Button element, or at least as FrameworkElement, to set Width
    Dim srcButton As Button
    srcButton = CType(e.Source, Button)
    srcButton.Width = 200
End Sub
```

Vedere anche

- [RoutedEventArgs](#)
- [Cenni preliminari sugli eventi indirizzati](#)
- [Procedure relative alle proprietà](#)

Procedura: Aggiungere la gestione di classi per un evento indirizzato

23/10/2019 • 4 minutes to read • [Edit Online](#)

Gli eventi indirizzati possono essere gestiti mediante i gestori classi o gestori dell'istanza in qualsiasi nodo specificato nella route. I gestori classi vengono richiamati per primi e utilizzabile da implementazioni della classe per eliminare gli eventi dalla gestione di istanze o introdurre altri comportamenti specifici di evento degli eventi che appartengono a classi di base. Questo esempio illustra due tecniche strettamente correlate per l'implementazione di gestori di classi.

Esempio

Questo esempio Usa una classe personalizzata in base il [Canvas](#) pannello. La premessa alla base dell'applicazione è che la classe personalizzata presenta comportamenti nei relativi elementi figlio, tra cui intercettazione fa clic su qualsiasi pulsante del mouse a sinistra e il contrassegno che essi gestiti, prima che verranno richiamate la classe di elemento figlio o qualsiasi gestore di istanze su di esso.

Il [UIElement](#) classe espone un metodo virtuale che consente la gestione della classe di [PreviewMouseLeftButtonDown](#) eventi, sostituendo semplicemente l'evento. Questo è il modo più semplice per implementare la gestione della classe se tale metodo virtuale è disponibile in una posizione nella gerarchia delle classi. Il codice seguente illustra il [OnPreviewMouseLeftButtonDown](#) implementazione in "MyEditContainer" che è derivato da [Canvas](#). L'implementazione contrassegna l'evento come gestito negli argomenti e quindi aggiunge codice per apportare una modifica base visibile l'elemento di origine.

```
protected override void OnPreviewMouseRightButtonDown(System.Windows.Input.MouseButtonEventArgs e)
{
    e.Handled = true; //suppress the click event and other leftmousebuttondown responders
    MyEditContainer ec = (MyEditContainer)e.Source;
    if (ec.EditState)
    { ec.EditState = false; }
    else
    { ec.EditState = true; }
    base.OnPreviewMouseRightButtonDown(e);
}
```

```
Protected Overrides Sub OnPreviewMouseRightButtonDown(ByVal e As System.Windows.Input.MouseButtonEventArgs)
    e.Handled = True 'suppress the click event and other leftmousebuttondown responders
    Dim ec As MyEditContainer = CType(e.Source, MyEditContainer)
    If ec.EditState Then
        ec.EditState = False
    Else
        ec.EditState = True
    End If
    MyBase.OnPreviewMouseRightButtonDown(e)
End Sub
```

Se nessun metodo virtuale è disponibile nelle classi di base o per il metodo specifico, la gestione di classi può essere aggiunti direttamente usando un metodo di utilità del [EventManager](#) classe [RegisterClassHandler](#). Questo metodo deve essere chiamato solo all'interno dell'inizializzazione statica delle classi che aggiunge la gestione di classi. Questo esempio viene aggiunto un altro gestore per [PreviewMouseLeftButtonDown](#), e in questo caso la classe registrata è la classe personalizzata. Al contrario, quando si usano i metodi virtuali, la classe registrata è

davvero il [UIElement](#) classe di base. Nei casi in cui le classi di base e ogni sottoclasse registrare la gestione di classi, i gestori di sottoclasse siano richiamati per primi. Il comportamento in un'applicazione sarebbe che prima di tutto questo gestore visualizza la finestra di messaggio e quindi visualizzerà la modifica visiva nel gestore del metodo virtuale.

```
static MyEditContainer()
{
    EventManager.RegisterClassHandler(typeof(MyEditContainer), PreviewMouseRightButtonDownEvent, new
    RoutedEventHandler(LocalOnMouseRightButtonDown));
}
internal static void LocalOnMouseRightButtonDown(object sender, RoutedEventArgs e)
{
    MessageBox.Show("this is invoked before the On* class handler on UIElement");
    //e.Handled = true; //uncommenting this would cause ONLY the subclass' class handler to respond
}
```

```
Shared Sub New()
    EventManager.RegisterClassHandler(GetType(MyEditContainer), PreviewMouseRightButtonDownEvent, New
    RoutedEventHandler(AddressOf LocalOnMouseRightButtonDown))
End Sub
Friend Shared Sub LocalOnMouseRightButtonDown(ByVal sender As Object, ByVal e As RoutedEventArgs)
    MessageBox.Show("this is invoked before the On* class handler on UIElement")
    'e.Handled = True //uncommenting this would cause ONLY the subclass' class handler to respond
End Sub
```

Vedere anche

- [EventManager](#)
- [Contrassegno degli eventi indirizzati come gestiti e gestione delle classi](#)
- [Gestire un evento indirizzato](#)
- [Cenni preliminari sugli eventi indirizzati](#)

Input (WPF)

03/02/2020 • 2 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) include il supporto per diversi tipi di input. Questo input include testo, tocco, mouse, comandi, messa a fuoco, tocco, trascinamento della selezione e input penna digitale. In questa sezione vengono descritti gli argomenti relativi all'input in WPF.

Contenuto della sezione

[Cenni preliminari sull'input](#)

[Panoramica sull'esecuzione di comandi](#)

[Panoramica sullo stato attivo](#)

[Applicazione di stili per lo stato attivo nei controlli e FocusVisualStyle](#)

[Procedura dettagliata: Creazione della prima applicazione a tocco](#)

[Procedure relative alla struttura ad albero e alla serializzazione degli elementi](#)

[Input penna](#)

Riferimento

[UIElement](#)

[FrameworkElement](#)

[ContentElement](#)

[FrameworkContentElement](#)

[Keyboard](#)

[Mouse](#)

[FocusManager](#)

Sezioni correlate

[Controlli](#)

[Eventi](#)

Cenni preliminari sull'input

23/11/2019 • 48 minutes to read • [Edit Online](#)

Il sottosistema Windows Presentation Foundation (WPF) fornisce un'API potente per ottenere input da un'ampia gamma di dispositivi, tra cui mouse, tastiera, tocco e stilo. Questo argomento descrive i servizi forniti da WPF e illustra l'architettura dei sistemi di input.

API di input

L'esposizione dell'API di input primaria si trova nelle classi degli elementi di base: [UIElement](#), [ContentElement](#), [FrameworkElement](#) e [FrameworkContentElement](#). Per altre informazioni sugli elementi di base, vedere [Cenni preliminari sugli elementi di base](#). Queste classi forniscono funzionalità per gli eventi di input correlati, ad esempio, a pressioni di tasti, pulsanti del mouse, rotellina del mouse, movimento del mouse, gestione dello stato attivo, stato mouse capture e altro. Inserendo l'API di input sugli elementi di base, invece di considerare tutti gli eventi di input come un servizio, l'architettura di input consente di originare gli eventi di input da un determinato oggetto nell'interfaccia utente e di supportare uno schema di routing degli eventi in base al quale più di un elemento dispone di un'opportunità per la gestione di un evento di input. A molti eventi di input è associata una coppia di eventi. Ad esempio, l'evento di riduzione della chiave è associato agli eventi [KeyDown](#) e [PreviewKeyDown](#). La differenza tra questi eventi consiste nel modo in cui ne viene eseguito il routing all'elemento di destinazione. Gli eventi di anteprima scendono lungo l'albero degli elementi (tunneling), dall'elemento radice all'elemento di destinazione. Gli eventi di bubbling invece salgono dall'elemento di destinazione all'elemento radice. Il routing degli eventi in WPF è illustrato in modo più dettagliato più avanti in questo articolo e nell'articolo [Cenni preliminari sugli eventi indirizzati](#).

Classi Keyboard e Mouse

Oltre all'API di input sulle classi degli elementi di base, le classi [Keyboard](#) Class e [Mouse](#) forniscono un'API aggiuntiva per lavorare con l'input della tastiera e del mouse.

Esempi di API di input nella classe [Keyboard](#) sono la proprietà [Modifiers](#), che restituisce il [ModifierKeys](#) attualmente premuto e il metodo [IsKeyDown](#), che determina se viene premuto un tasto specificato.

Nell'esempio seguente viene usato il metodo [GetKeyStates](#) per determinare se una [Key](#) è nello stato di inattività.

```
// Uses the Keyboard.GetKeyStates to determine if a key is down.  
// A bitwise AND operation is used in the comparison.  
// e is an instance of KeyEventArgs.  
if ((Keyboard.GetKeyStates(Key.Return) & KeyStates.Down) > 0)  
{  
    btnNone.Background = Brushes.Red;  
}
```

```
' Uses the Keyboard.GetKeyStates to determine if a key is down.  
' A bitwise AND operation is used in the comparison.  
' e is an instance of KeyEventArgs.  
If (Keyboard.GetKeyStates(Key.Return) And KeyStates.Down) > 0 Then  
    btnNone.Background = Brushes.Red
```

Gli esempi di API di input nella classe [Mouse](#) sono [MiddleButton](#), che ottiene lo stato del pulsante centrale del mouse e [DirectlyOver](#), che ottiene l'elemento sul quale si trova attualmente il puntatore del mouse.

Nell'esempio seguente viene determinato se il [LeftButton](#) sul mouse si trova nello stato [Pressed](#).

```
if (Mouse.LeftButton == MouseButtonState.Pressed)
{
    UpdateSampleResults("Left Button Pressed");
}
```

```
If Mouse.LeftButton = MouseButtonState.Pressed Then
    UpdateSampleResults("Left Button Pressed")
End If
```

Le classi [Mouse](#) e [Keyboard](#) sono descritte in dettaglio in questa panoramica.

Input dello stilo

WPF dispone del supporto integrato per l'[Stylus](#). Il [Stylus](#) è un input penna reso popolare dal Tablet PC. WPF applicazioni possono trattare lo stilo come mouse tramite l'API del mouse, ma WPF espone anche un'astrazione del dispositivo stilo che utilizza un modello simile a quello della tastiera e del mouse. Tutte le API correlate allo stilo contengono la parola "stilo".

Dal momento che lo stilo può funzionare come un mouse, le applicazioni che supportano solo l'input del mouse possono comunque ottenere automaticamente un certo livello di supporto dello stilo. Quando lo stilo viene usato in questo modo, l'applicazione può gestire l'evento dello stilo appropriato gestendo l'evento del mouse corrispondente. Inoltre, tramite l'astrazione del dispositivo stilo, sono disponibili anche servizi di livello superiore, come l'input penna. Per altre informazioni sull'input penna, vedere [Nozioni di base sull'input penna](#).

Routing di eventi

Un [FrameworkElement](#) può contenere altri elementi come elementi figlio nel relativo modello di contenuto, formando una struttura ad albero di elementi. In WPF l'elemento padre può partecipare all'input diretto verso i relativi elementi figlio o agli altri discendenti mediante la gestione degli eventi. Ciò risulta particolarmente utile per la creazione di controlli a partire da controlli più piccoli, processo noto come "composizione di controlli" o "composizione". Per altre informazioni sugli alberi degli elementi e sulla loro relazione con le route degli eventi, vedere [Strutture ad albero in WPF](#).

Il routing degli eventi è il processo di inoltro degli eventi a più elementi, in modo che un oggetto o un elemento specifico lungo la route possa offrire una risposta significativa (tramite la gestione) a un evento che potrebbe essere stato originato da un elemento diverso. Gli eventi indirizzati usano uno dei tre meccanismi di routing indicati di seguito: diretto, bubbling e tunneling. Nel routing diretto, l'elemento di origine è l'unico elemento che riceve la notifica e l'evento non viene indirizzato a nessun altro elemento. Tuttavia, l'evento indirizzato diretto offre ancora alcune funzionalità aggiuntive che sono presenti solo per gli eventi indirizzati anziché per gli eventi CLR standard. Il bubbling viene eseguito procedendo verso l'alto nell'albero degli elementi, notificando innanzitutto l'elemento che ha originato l'evento, quindi l'elemento padre e così via. Il tunneling parte dalla radice dell'albero degli elementi e procede verso il basso, terminando con l'elemento di origine. Per altre informazioni sugli eventi indirizzati, vedere [Cenni preliminari sugli eventi indirizzati](#).

Gli eventi di input WPF sono in genere a coppie, costituite da un evento di tunneling e un evento di bubbling. Gli eventi di tunneling si distinguono da quelli di bubbling tramite il prefisso "Preview". Ad esempio, [PreviewMouseMove](#) è la versione di tunneling di un evento di spostamento del mouse e [MouseMove](#) è la versione di bubbling di questo evento. Questa coppia di eventi è una convenzione implementata a livello di elemento e non è una funzionalità intrinseca del sistema di eventi WPF. Per informazioni dettagliate, vedere la sezione Eventi di input WPF in [Cenni preliminari sugli eventi indirizzati](#).

Gestione degli eventi di input

Per ricevere input su un elemento, un gestore eventi deve essere associato a quel particolare evento. In XAML tale associazione è semplice: è sufficiente fare riferimento al nome dell'evento come attributo dell'elemento che rimarrà in ascolto di questo evento. Si imposta quindi il valore dell'attributo sul nome del gestore eventi definito, in base a un delegato. Il gestore eventi deve essere scritto in codice, ad C# esempio e, può essere incluso in un file code-behind.

Gli eventi della tastiera si verificano quando il sistema operativo segnala azioni dei tasti eseguite quando lo stato attivo della tastiera è su un elemento. Gli eventi del mouse e dello stilo rientrano ciascuno in due categorie: eventi che segnalano modifiche nella posizione del puntatore rispetto all'elemento ed eventi che segnalano modifiche nello stato dei pulsanti del dispositivo.

Esempio di evento di input da tastiera

L'esempio seguente illustra una situazione di ascolto della pressione del tasto freccia SINISTRA. Viene creato un **StackPanel** con una **Button**. Un gestore eventi per restare in attesa della pressione del tasto freccia sinistra viene collegato all'istanza di **Button**.

Nella prima sezione dell'esempio vengono creati il **StackPanel** e il **Button** e viene collegato il gestore eventi per il **KeyDown**.

```
<StackPanel>
    <Button Background="AliceBlue"
        KeyDown="OnButtonKeyDown"
        Content="Button1"/>
</StackPanel>
```

```
// Create the UI elements.
StackPanel keyboardStackPanel = new StackPanel();
Button keyboardButton1 = new Button();

// Set properties on Buttons.
keyboardButton1.Background = Brushes.AliceBlue;
keyboardButton1.Content = "Button 1";

// Attach Buttons to StackPanel.
keyboardStackPanel.Children.Add(keyboardButton1);

// Attach event handler.
keyboardButton1.KeyDown += new KeyEventHandler(OnButtonKeyDown);
```

```
' Create the UI elements.
Dim keyboardStackPanel As New StackPanel()
Dim keyboardButton1 As New Button()

' Set properties on Buttons.
keyboardButton1.Background = Brushes.AliceBlue
keyboardButton1.Content = "Button 1"

' Attach Buttons to StackPanel.
keyboardStackPanel.Children.Add(keyboardButton1)

' Attach event handler.
AddHandler keyboardButton1.KeyDown, AddressOf OnButtonKeyDown
```

La seconda sezione è scritta nel codice e definisce il gestore eventi. Quando viene premuto il tasto freccia sinistra e il **Button** dispone dello stato attivo della tastiera, il gestore viene eseguito e il colore **Background** del **Button** viene modificato. Se il tasto è premuto, ma non è il tasto freccia sinistra, il colore **Background** della **Button** viene nuovamente modificato sul colore iniziale.

```

private void OnButtonKeyDown(object sender, KeyEventArgs e)
{
    Button source = e.Source as Button;
    if (source != null)
    {
        if (e.Key == Key.Left)
        {
            source.Background = Brushes.LemonChiffon;
        }
        else
        {
            source.Background = Brushes.AliceBlue;
        }
    }
}

```

```

Private Sub OnButtonKeyDown(ByVal sender As Object, ByVal e As KeyEventArgs)
    Dim source As Button = TryCast(e.Source, Button)
    If source IsNot Nothing Then
        If e.Key = Key.Left Then
            source.Background = Brushes.LemonChiffon
        Else
            source.Background = Brushes.AliceBlue
        End If
    End If
End Sub

```

Esempio di evento di input del mouse

Nell'esempio seguente, il colore **Background** di un **Button** viene modificato quando il puntatore del mouse entra nell'**Button**. Il colore **Background** viene ripristinato quando il mouse esce dall'**Button**.

La prima sezione dell'esempio crea il **StackPanel** e il controllo **Button** e connette i gestori eventi per gli eventi **MouseEnter** e **MouseLeave** al **Button**.

```

<StackPanel>
    <Button Background="AliceBlue"
        MouseEnter="OnMouseExampleMouseEnter"
        MouseLeave="OnMosueExampleMouseLeave">Button

    </Button>
</StackPanel>

```

```

// Create the UI elements.
StackPanel mouseMoveStackPanel = new StackPanel();
Button mouseMoveButton = new Button();

// Set properties on Button.
mouseMoveButton.Background = Brushes.AliceBlue;
mouseMoveButton.Content = "Button";

// Attach Buttons to StackPanel.
mouseMoveStackPanel.Children.Add(mouseMoveButton);

// Attach event handler.
mouseMoveButton.MouseEnter += new MouseEventHandler(OnMouseExampleMouseEnter);
mouseMoveButton.MouseLeave += new MouseEventHandler(OnMosueExampleMouseLeave);

```

```

' Create the UI elements.
Dim mouseMoveStackPanel As New StackPanel()
Dim mouseMoveButton As New Button()

' Set properties on Button.
mouseMoveButton.Background = Brushes.AliceBlue
mouseMoveButton.Content = "Button"

' Attach Buttons to StackPanel.
mouseMoveStackPanel.Children.Add(mouseMoveButton)

' Attach event handler.
AddHandler mouseMoveButton.MouseEnter, AddressOf OnMouseExampleMouseEnter
AddHandler mouseMoveButton.MouseLeave, AddressOf OnMosueExampleMouseLeave

```

La seconda sezione dell'esempio è scritta nel codice e definisce i gestori eventi. Quando il mouse entra nel [Button](#), il colore [Background](#) del [Button](#) viene modificato in [SlateGray](#). Quando il mouse esce dalla [Button](#), il colore [Background](#) del [Button](#) viene nuovamente modificato in [AliceBlue](#).

```

private void OnMouseExampleMouseEnter(object sender, MouseEventArgs e)
{
    // Cast the source of the event to a Button.
    Button source = e.Source as Button;

    // If source is a Button.
    if (source != null)
    {
        source.Background = Brushes.SlateGray;
    }
}

```

```

Private Sub OnMouseExampleMouseEnter(ByVal sender As Object, ByVal e As MouseEventArgs)
    ' Cast the source of the event to a Button.
    Dim source As Button = TryCast(e.Source, Button)

    ' If source is a Button.
    If source IsNot Nothing Then
        source.Background = Brushes.SlateGray
    End If
End Sub

```

```

private void OnMosueExampleMouseLeave(object sender, MouseEventArgs e)
{
    // Cast the source of the event to a Button.
    Button source = e.Source as Button;

    // If source is a Button.
    if (source != null)
    {
        source.Background = Brushes.AliceBlue;
    }
}

```

```

Private Sub OnMouseExampleMouseLeave(ByVal sender As Object, ByVal e As MouseEventArgs)
    ' Cast the source of the event to a Button.
    Dim source As Button = TryCast(e.Source, Button)

    ' If source is a Button.
    If source IsNot Nothing Then
        source.Background = Brushes.AliceBlue
    End If
End Sub

```

Input di testo

L'evento [TextInput](#) consente di ascoltare l'input di testo in modo indipendente dal dispositivo. La tastiera è il mezzo principale per l'immissione di testo, ma anche i comandi vocali, il riconoscimento della grafia e altri dispositivi di input possono generare input di testo.

Per l'input da tastiera, WPF invia prima gli eventi [KeyDown/KeyUp](#) appropriati. Se tali eventi non vengono gestiti e la chiave è testuale, anziché una chiave di controllo come frecce direzionali o tasti funzione, viene generato un evento [TextInput](#). Non è sempre presente un semplice mapping uno-a-uno tra [KeyDown/KeyUp](#) e [TextInput](#) eventi perché più sequenze di tasti possono generare un singolo carattere di input di testo e le singole sequenze di tasti possono generare stringhe mult(carattere). Ciò vale soprattutto per le lingue quali il cinese, il giapponese e il coreano che usano IME (Input Method Editor) per generare le migliaia di caratteri possibili negli alfabeti corrispondenti.

Quando WPF Invia un evento [KeyUp/KeyDown](#), [Key](#) viene impostato su [Key.System](#) se le sequenze di tasti potrebbero far parte di un evento [TextInput](#) (ad esempio, se si preme ALT + S). Ciò consente al codice in un gestore dell'evento [KeyDown](#) di verificare la presenza di [Key.System](#) e, se presente, di lasciare l'elaborazione per il gestore dell'evento di [TextInput](#) generato successivamente. In questi casi, è possibile usare le varie proprietà dell'argomento [TextCompositionEventArgs](#) per determinare le sequenze di tasti originali.

Analogamente, se un IME è attivo, [Key](#) ha il valore di [Key.ImeProcessed](#) e [ImeProcessedKey](#) fornisce la sequenza di tasti originale o le sequenze di tasti.

Nell'esempio seguente viene definito un gestore per l'evento [Click](#) e un gestore per l'evento [KeyDown](#).

Il primo segmento di codice o markup crea l'interfaccia utente.

```

<StackPanel KeyDown="OnTextInputKeyDown">
    <Button Click="OnTextInputButtonClick"
            Content="Open" />
    <TextBox> . . . </TextBox>
</StackPanel>

```

```

// Create the UI elements.
StackPanel textInputStackPanel = new StackPanel();
Button textInputButton = new Button();
TextBox textInputTextBox = new TextBox();
textInputButton.Content = "Open";

// Attach elements to StackPanel.
textInputStackPanel.Children.Add(textInputButton);
textInputStackPanel.Children.Add(textInputTextBox);

// Attach event handlers.
textInputStackPanel.KeyDown += new KeyEventHandler(OnTextInputKeyDown);
textInputButton.Click += new RoutedEventHandler(OnTextInputButtonClick);

```

```

' Create the UI elements.
Dim textInputStackPanel As New StackPanel()
Dim textInputButton As New Button()
Dim textInputTextBox As New TextBox()
textInputButton.Content = "Open"

' Attach elements to StackPanel.
textInputStackPanel.Children.Add(textInputButton)
textInputStackPanel.Children.Add(textInputTextBox)

' Attach event handlers.
AddHandler textInputStackPanel.KeyDown, AddressOf OnTextInputKeyDown
AddHandler textInputButton.Click, AddressOf OnTextInputButtonClick

```

Il secondo segmento di codice contiene i gestori eventi.

```

private void OnTextInputKeyDown(object sender, KeyEventArgs e)
{
    if (e.Key == Key.O && Keyboard.Modifiers == ModifierKeys.Control)
    {
        handle();
        e.Handled = true;
    }
}

private void OnTextInputButtonClick(object sender, RoutedEventArgs e)
{
    handle();
    e.Handled = true;
}

public void handle()
{
    MessageBox.Show("Pretend this opens a file");
}

```

```

Private Sub OnTextInputKeyDown(ByVal sender As Object, ByVal e As KeyEventArgs)
    If e.Key = Key.O AndAlso Keyboard.Modifiers = ModifierKeys.Control Then
        handle()
        e.Handled = True
    End If
End Sub

Private Sub OnTextInputButtonClick(ByVal sender As Object, ByVal e As RoutedEventArgs)
    handle()
    e.Handled = True
End Sub

Public Sub handle()
    MessageBox.Show("Pretend this opens a file")
End Sub

```

Poiché gli eventi di input compongono la route dell'evento, il [StackPanel](#) riceve l'input indipendentemente dall'elemento con lo stato attivo della tastiera. Il controllo [TextBox](#) viene notificato per primo e viene chiamato il gestore [OnTextInputKeyDown](#) solo se il [TextBox](#) non ha gestito l'input. Se viene utilizzato l'evento [PreviewKeyDown](#) al posto dell'evento [KeyDown](#), viene chiamato per primo il gestore di [OnTextInputKeyDown](#).

In questo esempio la logica di gestione viene scritta due volte, una per CTRL+O e una per l'evento Click del pulsante. È possibile semplificare questa operazione usando i comandi, invece di gestire direttamente gli eventi di input. I comandi vengono illustrati in questo articolo e in [Cenni preliminari sull'esecuzione di comandi](#).

Tocco e manipolazione

Il nuovo hardware e le nuove API del sistema operativo Windows 7 consentono alle applicazioni di ricevere contemporaneamente input da più tocchi. WPF consente alle applicazioni di rilevare il tocco e rispondere in un modo simile alle risposte ad altri tipi di input, ad esempio di mouse o tastiera, generando eventi quando si verifica il tocco.

WPF espone due tipi di eventi quando si verifica il tocco: eventi di tocco ed eventi di manipolazione. Gli eventi di tocco forniscono dati non elaborati relativi a ogni dito appoggiato su un touchscreen e al suo movimento. Gli eventi di manipolazione interpretano l'input come azioni specifiche. Questa sezione illustra entrambi i tipi di eventi.

Prerequisites

Per sviluppare un'applicazione che risponde al tocco, sono necessari i componenti seguenti.

- Visual Studio 2010.
- Windows 7.
- Un dispositivo, ad esempio un touchscreen, che supporta Windows Touch.

Terminologia

Quando si parla di tocco, vengono usati i termini seguenti.

- **Tocco** è un tipo di input dell'utente riconosciuto da Windows 7. In genere, il tocco viene generato appoggiando le dita su uno schermo sensibile al tocco. Si noti che dispositivi come i touchpad, diffusi sui computer portatili, non supportano il tocco se il dispositivo si limita a convertire la posizione e il movimento del dito come input del mouse.
- **Multitocco** è un tocco che si verifica in più punti contemporaneamente. Windows 7 e WPF supportano l'input multitocco. Ogni volta che il tocco viene trattato nella documentazione relativa a WPF, i concetti illustrati si applicano al multitocco.
- Una **manipolazione** si verifica quando il tocco viene interpretato come azione fisica applicata a un oggetto. In WPF gli eventi di manipolazione interpretano l'input come una manipolazione di traslazione, espansione o rotazione.
- Un `touch device` rappresenta un dispositivo che produce un input tocco, ad esempio un singolo dito su un touchscreen.

Controlli che rispondono al tocco

È possibile scorrere i controlli seguenti trascinando un dito attraverso il controllo, se c'è contenuto non visualizzato.

- [ComboBox](#)
- [ContextMenu](#)
- [DataGrid](#)
- [ListBox](#)
- [ListView](#)
- [MenuItem](#)
- [TextBox](#)
- [ToolBar](#)

- [TreeView](#)

Il [ScrollViewer](#) definisce la proprietà associata [ScrollViewer.PanningMode](#) che consente di specificare se il panning tocco è abilitato orizzontalmente, verticalmente, entrambi o nessuno dei due. La proprietà [ScrollViewer.PanningDeceleration](#) specifica la velocità con cui lo scorrimento rallenta quando l'utente solleva il dito dal touchscreen. Il [ScrollViewer.PanningRatio](#) proprietà associata specifica il rapporto tra offset di scorrimento e offset di modifica della conversione.

Eventi di tocco

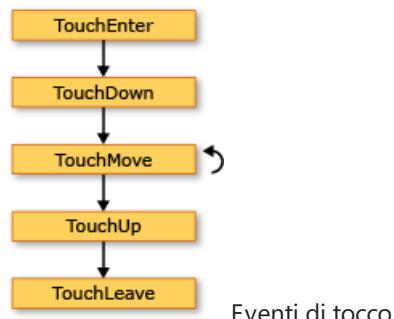
Le classi base, [UIElement](#), [UIElement3D](#) e [ContentElement](#) definiscono gli eventi che è possibile sottoscrivere, in modo che l'applicazione possa rispondere al tocco. Gli eventi di tocco sono utili quando l'applicazione interpreta il tocco come qualcosa di diverso rispetto alla manipolazione di un oggetto. Un'applicazione che consente a un utente di disegnare con un dito o con più dita deve ad esempio sottoscrivere gli eventi di tocco.

Tutte tre le classi definiscono gli eventi seguenti, che hanno un comportamento simile, indipendentemente dalla classe che li definisce.

- [TouchDown](#)
- [TouchMove](#)
- [TouchUp](#)
- [TouchEnter](#)
- [TouchLeave](#)
- [PreviewTouchDown](#)
- [PreviewTouchMove](#)
- [PreviewTouchUp](#)
- [GotTouchCapture](#)
- [LostTouchCapture](#)

Analogamente agli eventi di tastiera e mouse, gli eventi di tocco sono eventi indirizzati. Gli eventi che iniziano con `Preview` sono eventi di tunneling e gli eventi che iniziano con `Touch` sono eventi di bubbling. Per altre informazioni sugli eventi indirizzati, vedere [Cenni preliminari sugli eventi indirizzati](#). Quando si gestiscono questi eventi, è possibile ottenere la posizione dell'input, relativa a qualsiasi elemento, chiamando il metodo [GetTouchPoint](#) o [GetIntermediateTouchPoints](#).

Per comprendere l'interazione tra gli eventi di tocco, si consideri lo scenario in cui un utente appoggia un dito su un elemento, muove il dito all'interno dell'elemento e quindi solleva il dito dall'elemento. La figura seguente illustra l'esecuzione degli eventi di bubbling. Gli eventi di tunneling vengono omessi per maggiore semplicità.



L'elenco seguente descrive la sequenza degli eventi della figura precedente.

1. L'evento [TouchEnter](#) si verifica una volta quando l'utente inserisce un dito sull'elemento.

2. L'evento [TouchDown](#) si verifica una volta.
3. L'evento [TouchMove](#) si verifica più volte quando l'utente sposta il dito all'interno dell'elemento.
4. L'evento [TouchUp](#) si verifica una volta quando l'utente solleva il dito dall'elemento.
5. L'evento [TouchLeave](#) si verifica una volta.

Quando vengono usate più di due dita, gli eventi si verificano per ogni dito.

Eventi di manipolazione

Per i casi in cui un'applicazione consente a un utente di modificare un oggetto, la classe [UIElement](#) definisce gli eventi di manipolazione. A differenza degli eventi di tocco che segnalano semplicemente la posizione del tocco, gli eventi di manipolazione segnalano come può essere interpretato l'input. Ci sono tre tipi di manipolazioni: traslazione, espansione e rotazione. L'elenco seguente descrive come richiamare i tre tipi di manipolazioni.

- Appoggiare un dito su un oggetto e muovere il dito sul touchscreen per richiamare una manipolazione di traslazione. Ciò comporta in genere lo spostamento dell'oggetto.
- Appoggiare due dita su un oggetto e avvicinare o allontanare le dita tra loro per richiamare una manipolazione di espansione. Ciò comporta in genere il ridimensionamento dell'oggetto.
- Appoggiare due dita su un oggetto e ruotare le dita una attorno all'altra per richiamare una manipolazione di rotazione. Ciò comporta in genere la rotazione dell'oggetto.

Si possono verificare più tipi di manipolazioni contemporaneamente.

Quando gli oggetti rispondono alle manipolazioni, si può fare in modo che sembrino avere un'inerzia. In questo modo, gli oggetti possono simulare il mondo fisico. Ad esempio se si spinge un libro su un tavolo con sufficiente forza, il libro continuerà a muoversi anche dopo che lo si lascia. WPF consente di simulare questo comportamento generando eventi di manipolazione dopo che le dita dell'utente rilasciano l'oggetto.

Per informazioni su come creare un'applicazione che consente all'utente di spostare, ridimensionare e ruotare un oggetto, vedere [Procedura dettagliata: creazione della prima applicazione a tocco](#).

Il [UIElement](#) definisce gli eventi di manipolazione seguenti.

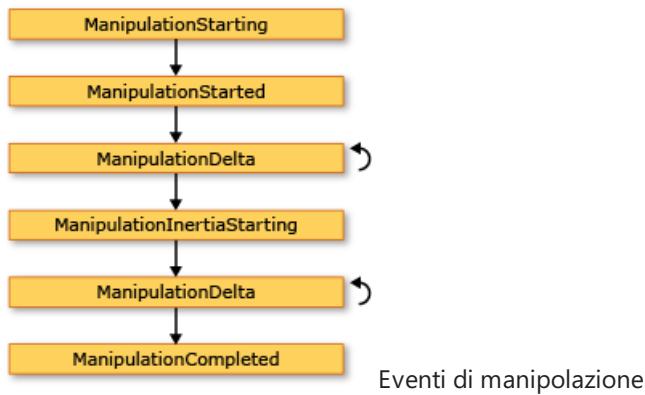
- [ManipulationStarting](#)
- [ManipulationStarted](#)
- [ManipulationDelta](#)
- [ManipulationInertiaStarting](#)
- [ManipulationCompleted](#)
- [ManipulationBoundaryFeedback](#)

Per impostazione predefinita, un [UIElement](#) non riceve questi eventi di manipolazione. Per ricevere eventi di manipolazione in un [UIElement](#), impostare [UIElement.IsManipulationEnabled](#) su `true`.

Percorso di esecuzione degli eventi di manipolazione

Si consideri uno scenario in cui un utente "lancia" un oggetto. L'utente appoggia un dito sull'oggetto, sposta il dito sul touchscreen per un breve tratto e quindi solleva il dito mentre l'oggetto si sta muovendo. Il risultato di questa azione è che l'oggetto si muoverà sotto il dito dell'utente e continuerà a muoversi dopo che l'utente ha sollevato il dito.

La figura seguente mostra il percorso di esecuzione degli eventi di manipolazione, con informazioni importanti relative a ogni evento.



L'elenco seguente descrive la sequenza degli eventi della figura precedente.

1. L'evento **ManipulationStarting** si verifica quando l'utente posiziona un dito sull'oggetto. Tra le altre cose, questo evento consente di impostare la proprietà **ManipulationContainer**. Negli eventi successivi la posizione della manipolazione sarà relativa all'**ManipulationContainer**. In eventi diversi da **ManipulationStarting** questa proprietà è di sola lettura, pertanto l'evento **ManipulationStarting** è l'unico momento in cui è possibile impostare questa proprietà.
2. L'evento **ManipulationStarted** si verifica successivamente. Questo evento segnala l'origine della manipolazione.
3. L'evento **ManipulationDelta** si verifica più volte quando il dito di un utente si sposta su un touchscreen. La proprietà **DeltaManipulation** della classe **ManipulationDeltaEventArgs** indica se la manipolazione viene interpretata come spostamento, espansione o traduzione. In questo contesto si esegue la maggior parte del lavoro di manipolazione di un oggetto.
4. L'evento **ManipulationInertiaStarting** si verifica quando le dita dell'utente perdono il contatto con l'oggetto. Questo evento consente di specificare la decelerazione delle manipolazioni durante l'inerzia. In questo modo, l'oggetto può emulare spazi fisici o attribuiti diversi, se si desidera. Si supponga, ad esempio, che l'applicazione includa due oggetti che rappresentano elementi nel mondo fisico e che uno di questi sia più pesante dell'altro. È possibile fare in modo che l'oggetto più pesante rallenti più velocemente rispetto a quello più leggero.
5. L'evento **ManipulationDelta** si verifica più volte quando si verifica un'inerzia. Si noti che questo evento si verifica quando le dita dell'utente si muovono sul touchscreen e quando WPF simula l'inerzia. In altre parole, **ManipulationDelta** si verifica prima e dopo l'evento **ManipulationInertiaStarting**. La proprietà **ManipulationDeltaEventArgs.IsInertial** segnala se l'evento **ManipulationDelta** si verifica durante l'inerzia, quindi è possibile controllare tale proprietà ed eseguire azioni diverse, a seconda del valore.
6. L'evento **ManipulationCompleted** si verifica al termine della manipolazione e di qualsiasi inerzia. Ciò significa che, dopo che si sono verificati tutti gli eventi **ManipulationDelta**, viene generato l'evento **ManipulationCompleted** per segnalare che la manipolazione è stata completata.

Il **UIElement** definisce anche l'evento **ManipulationBoundaryFeedback**. Questo evento si verifica quando viene chiamato il metodo **ReportBoundaryFeedback** nell'evento **ManipulationDelta**. L'evento **ManipulationBoundaryFeedback** consente alle applicazioni o ai componenti di fornire feedback visivo quando un oggetto raggiunge un limite. Ad esempio, la classe **Window** gestisce l'evento **ManipulationBoundaryFeedback** per far sì che la finestra si sposti leggermente quando viene rilevato il bordo.

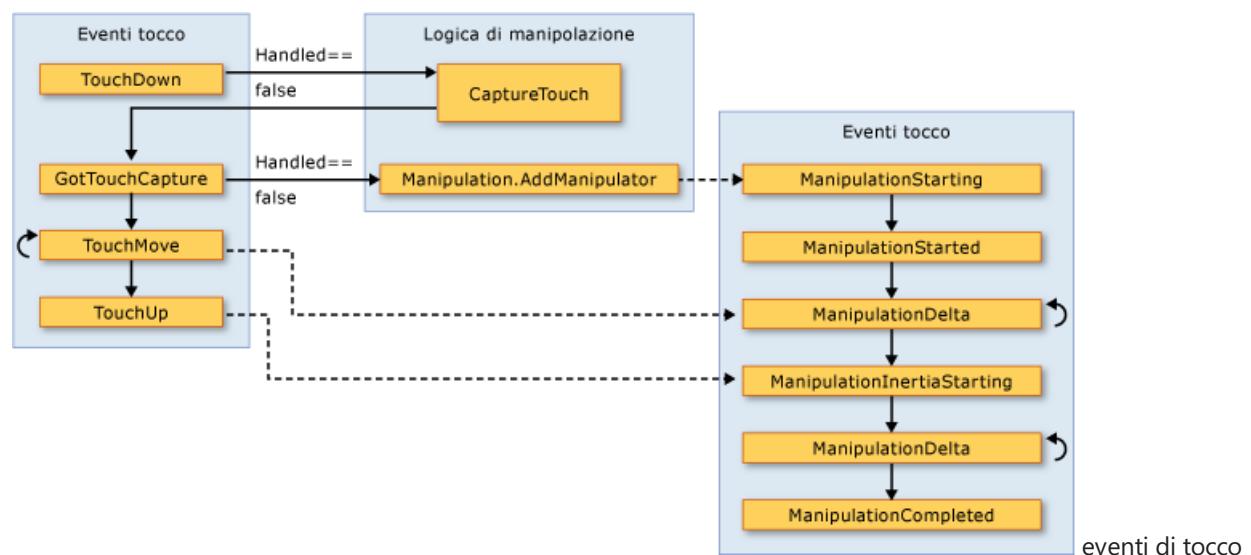
È possibile annullare la manipolazione chiamando il metodo **Cancel** sugli argomenti dell'evento in qualsiasi evento di manipolazione eccetto **ManipulationBoundaryFeedback** evento. Quando si chiama **Cancel**, gli eventi di manipolazione non vengono più generati e si verificano gli eventi del mouse per il tocco. La tabella seguente descrive la relazione tra il momento in cui viene annullata la manipolazione e gli eventi del mouse che si verificano.

EVENTO SU CUI VIENE CHIAMATO IL METODO CANCEL	EVENTI DEL MOUSE CHE SI VERIFICANO PER L'INPUT GIÀ AVVENUTO
ManipulationStarting e ManipulationStarted	Eventi di pressione del pulsante del mouse.
ManipulationDelta	Eventi di pressione del pulsante e di spostamento del mouse.
ManipulationInertiaStarting e ManipulationCompleted	Eventi di pressione e rilascio del pulsante e di spostamento del mouse.

Si noti che se si chiama `Cancel` quando la manipolazione è in inerzia, il metodo restituisce `false` e l'input non genera eventi del mouse.

Relazione tra eventi di tocco e di manipolazione

Un `UIElement` può sempre ricevere eventi di tocco. Quando la proprietà `IsManipulationEnabled` è impostata su `true`, un `UIElement` può ricevere eventi di tocco e di manipolazione. Se l'evento `TouchDown` non viene gestito (ovvero la proprietà `Handled` è `false`), la logica di manipolazione acquisisce il tocco sull'elemento e genera gli eventi di manipolazione. Se la proprietà `Handled` è impostata su `true` nell'evento `TouchDown`, la logica di manipolazione non genera eventi di manipolazione. La figura seguente mostra la relazione tra eventi di tocco ed eventi di manipolazione.



e manipolazione

L'elenco seguente descrive la relazione tra gli eventi di tocco e gli eventi di manipolazione illustrati nella figura precedente.

- Quando il primo dispositivo touch genera un evento `TouchDown` in un `UIElement`, la logica di manipolazione chiama il metodo `CaptureTouch`, che genera l'evento `GotTouchCapture`.
- Quando si verifica il `GotTouchCapture`, la logica di manipolazione chiama il metodo `Manipulation.AddManipulator`, che genera l'evento di `ManipulationStarting`.
- Quando si verificano gli eventi di `TouchMove`, la logica di manipolazione genera gli eventi di `ManipulationDelta` che si verificano prima dell'evento di `ManipulationInertiaStarting`.
- Quando l'ultimo dispositivo touch sull'elemento genera l'evento `TouchUp`, la logica di manipolazione genera l'evento `ManipulationInertiaStarting`.

Stato attivo

Ci sono due concetti principali relativi allo stato attivo in WPF: lo stato attivo della tastiera e lo stato attivo

logico.

Stato attivo della tastiera

Lo stato attivo della tastiera fa riferimento all'elemento che riceve l'input dalla tastiera. Su un desktop può esserci un solo elemento con lo stato attivo della tastiera. In WPF, l'elemento con lo stato attivo della tastiera sarà `IsKeyboardFocused` impostato su `true`. Il metodo statico di `Keyboard.FocusedElement` restituisce l'elemento che dispone attualmente dello stato attivo della tastiera.

Lo stato attivo della tastiera può essere ottenuto tramite tabulazione su un elemento o facendo clic sul mouse su determinati elementi, ad esempio un `TextBox`. Lo stato attivo della tastiera può essere ottenuto anche a livello di programmazione tramite il metodo `Focus` sulla classe `Keyboard`. `Focus` tenta di assegnare lo stato attivo della tastiera all'elemento specificato. L'elemento restituito da `Focus` è l'elemento che dispone attualmente dello stato attivo della tastiera.

Affinché un elemento ottenga lo stato attivo della tastiera, la proprietà `Focusable` e le proprietà `IsVisible` devono essere impostate su `true`. Alcune classi, ad esempio `Panel`, `Focusable` impostate su `false` per impostazione predefinita. Pertanto, potrebbe essere necessario impostare questa proprietà su `true` se si desidera che l'elemento possa ottenere lo stato attivo.

Nell'esempio seguente viene usato `Focus` per impostare lo stato attivo della tastiera su una `Button`. La posizione consigliata per impostare lo stato attivo iniziale in un'applicazione si trova nel gestore dell'evento `Loaded`.

```
private void OnLoaded(object sender, RoutedEventArgs e)
{
    // Sets keyboard focus on the first Button in the sample.
    Keyboard.Focus(firstButton);
}
```

```
Private Sub OnLoaded(ByVal sender As Object, ByVal e As RoutedEventArgs)
    ' Sets keyboard focus on the first Button in the sample.
    Keyboard.Focus(firstButton)
End Sub
```

Per altre informazioni sullo stato attivo della tastiera, vedere [Cenni preliminari sullo stato attivo](#).

Stato attivo logico

Lo stato attivo logico fa riferimento all'`FocusManager.FocusedElement` in un ambito di stato attivo. In un'applicazione possono esserci più elementi con lo stato attivo logico, ma in un determinato ambito di stato attivo può esserci un solo elemento con lo stato attivo logico.

Un ambito di stato attivo è un elemento contenitore che tiene traccia dell'`FocusedElement` all'interno dell'ambito. Quando lo stato attivo lascia un ambito di stato attivo, l'elemento con lo stato attivo perde lo stato attivo della tastiera, ma mantiene quello logico. Quando lo stato attivo torna nell'ambito di stato attivo, l'elemento con lo stato attivo ottiene nuovamente lo stato attivo della tastiera. In questo modo, lo stato attivo della tastiera può essere passato tra più ambiti, ma l'elemento con lo stato attivo in un determinato ambito rimane sicuramente tale al ritorno dello stato attivo.

Un elemento può essere trasformato in un ambito di stato attivo in Extensible Application Markup Language (XAML) impostando la proprietà associata `FocusManager.IsFocusScope` `true` o nel codice impostando la proprietà associata tramite il metodo `SetIsFocusScope`.

Nell'esempio seguente un `StackPanel` viene trasformato in un ambito di stato attivo impostando la proprietà `IsFocusScope` associata.

```
<StackPanel Name="focusScope1"
    FocusManager.IsFocusScope="True"
    Height="200" Width="200">
    <Button Name="button1" Height="50" Width="50"/>
    <Button Name="button2" Height="50" Width="50"/>
</StackPanel>
```

```
StackPanel focusScope2 = new StackPanel();
FocusManager.SetIsFocusScope(focusScope2, true);
```

```
Dim focusScope2 As New StackPanel()
FocusManager.SetIsFocusScope(focusScope2, True)
```

Per impostazione predefinita, le classi di WPF che sono ambiti di stato attivo sono [Window](#), [Menu](#), [ToolBar](#) e [ContextMenu](#).

Un elemento con lo stato attivo della tastiera avrà anche lo stato attivo logico per l'ambito di stato attivo a cui appartiene. Pertanto, l'impostazione dello stato attivo su un elemento con il metodo [Focus](#) sulla classe [Keyboard](#) o sulle classi degli elementi di base tenterà di assegnare lo stato attivo della tastiera e lo stato attivo logico dell'elemento.

Per determinare l'elemento con stato attivo in un ambito di stato attivo, utilizzare [GetFocusedElement](#). Per modificare l'elemento con stato attivo per un ambito di stato attivo, utilizzare [SetFocusedElement](#).

Per altre informazioni sullo stato attivo logico, vedere [Cenni preliminari sullo stato attivo](#).

Posizione del mouse

L'API di input WPF fornisce informazioni utili per quanto riguarda gli spazi delle coordinate. Ad esempio, la coordinata `(0,0)` rappresenta la coordinata superiore sinistra, ma di quale elemento dell'albero? Si tratta dell'elemento che costituisce la destinazione dell'input? O dell'elemento a cui è associato il gestore eventi? Oppure di qualche altro elemento? Per evitare confusione, l'API di input WPF richiede di specificare il frame di riferimento quando si utilizzano le coordinate ottenute tramite il mouse. Il metodo [GetPosition](#) restituisce la coordinata del puntatore del mouse relativa all'elemento specificato.

Mouse Capture

I dispositivi di tipo mouse hanno una caratteristica modale specifica nota come mouse capture. Tale caratteristica viene usata per mantenere uno stato di input di transizione all'avvio di un'operazione di trascinamento della selezione, in modo che le altre operazioni che riguardano la posizione su schermo nominale del puntatore del mouse non vengano necessariamente eseguite. Durante il trascinamento, l'utente non può fare clic senza interrompere il trascinamento della selezione, quindi la maggior parte dei segnali di passaggio del mouse risulterà inappropriata quando l'origine del trascinamento mantiene lo stato mouse capture. Il sistema di input espone le API che possono determinare lo stato di acquisizione del mouse, nonché le API che possono forzare l'acquisizione del mouse su un elemento specifico o cancellare lo stato di acquisizione del mouse. Per altre informazioni sulle operazioni di trascinamento della selezione, vedere [Cenni preliminari sul trascinamento della selezione](#).

Commands

I comandi consentono la gestione dell'input a un livello più semantico rispetto all'input dei dispositivi. I comandi sono semplici direttive, come `Cut`, `Copy`, `Paste` o `Open`. I comandi sono utili per centralizzare la logica di comando. È possibile accedere allo stesso comando da un [Menu](#), da un [ToolBar](#) tramite un tasto di

scelta rapida. I comandi forniscono anche un meccanismo per disabilitare i controlli quando il comando non è più disponibile.

[RoutedCommand](#) è l'implementazione WPF di [ICommand](#). Quando viene eseguita una [RoutedCommand](#), nella destinazione del comando vengono generati un [PreviewExecuted](#) e un evento [Executed](#), che eseguono il tunneling e il bubbling attraverso l'albero degli elementi come altro input. Se non è impostata una destinazione del comando, questa sarà rappresentata dall'elemento con lo stato attivo della tastiera. La logica che esegue il comando è associata a un [CommandBinding](#). Quando un evento [Executed](#) raggiunge un [CommandBinding](#) per quel comando specifico, viene chiamato il [ExecutedRoutedEventHandler](#) sul [CommandBinding](#). Il gestore esegue l'azione del comando.

Per altre informazioni sui comandi, vedere [Cenni preliminari sull'esecuzione di comandi](#).

WPF fornisce una libreria di comandi comuni costituiti da [ApplicationCommands](#), [MediaCommands](#), [ComponentCommands](#), [NavigationCommand](#) e [EditingCommands](#); oppure è possibile definirne di personalizzati.

Nell'esempio seguente viene illustrato come configurare un [MenuItem](#) in modo che, quando viene fatto clic, richiami il comando [Paste](#) sul [TextBox](#), supponendo che il [TextBox](#) abbia lo stato attivo della tastiera.

```
<StackPanel>
  <Menu>
    <MenuItem Command="ApplicationCommands.Paste" />
  </Menu>
  <TextBox />
</StackPanel>
```

```
// Creating the UI objects
StackPanel mainStackPanel = new StackPanel();
TextBox pasteTextBox = new TextBox();
Menu stackPanelMenu = new Menu();
MenuItem pasteMenuItem = new MenuItem();

// Adding objects to the panel and the menu
stackPanelMenu.Items.Add(pasteMenuItem);
mainStackPanel.Children.Add(stackPanelMenu);
mainStackPanel.Children.Add(pasteTextBox);

// Setting the command to the Paste command
pasteMenuItem.Command = ApplicationCommands.Paste;

// Setting the command target to the TextBox
pasteMenuItem.CommandTarget = pasteTextBox;
```

```
' Creating the UI objects
Dim mainStackPanel As New StackPanel()
Dim pasteTextBox As New TextBox()
Dim stackPanelMenu As New Menu()
Dim pasteMenuItem As New MenuItem()

' Adding objects to the panel and the menu
stackPanelMenu.Items.Add(pasteMenuItem)
mainStackPanel.Children.Add(stackPanelMenu)
mainStackPanel.Children.Add(pasteTextBox)

' Setting the command to the Paste command
pasteMenuItem.Command = ApplicationCommands.Paste
```

Per altre informazioni sui comandi in WPF, vedere [Cenni preliminari sull'esecuzione di comandi](#).

Sistema di input ed elementi di base

Gli eventi di input come gli eventi associati definiti dalle classi [Mouse](#), [Keyboarde Stylus](#) vengono generati dal sistema di input e inseriti in una determinata posizione nel modello a oggetti in base all'hit test della struttura ad albero visuale in fase di esecuzione.

Ogni evento che [Mouse](#), [Keyboarde Stylus](#) definire come un evento associato viene anche nuovamente esposto dalle classi degli elementi di base [UIElement](#) e [ContentElement](#) come nuovo evento indirizzato. Gli eventi indirizzati degli elementi di base vengono generati da classi che gestiscono l'evento associato originale e riutilizzano i dati dell'evento.

Quando l'evento di input viene associato a un particolare elemento di origine tramite l'implementazione dell'evento di input dei relativi elementi di base, può essere indirizzato lungo la parte restante della route di un evento basata su una combinazione di oggetti albero logico e struttura ad albero visuale ed essere gestito dal codice dell'applicazione. In genere, è più facile gestire questi eventi di input correlati al dispositivo usando gli eventi indirizzati in [UIElement](#) e [ContentElement](#), perché è possibile usare una sintassi del gestore eventi più intuitiva sia in XAML che nel codice. È possibile scegliere di gestire invece l'evento associato che ha avviato il processo, ma potrebbero sorgere diversi problemi: l'evento associato potrebbe essere contrassegnato come gestito dalla gestione della classe dell'elemento di base, per cui potrebbe essere necessario usare i metodi della funzione di accesso invece della sintassi dell'evento vera e propria per associare i gestori per gli eventi associati.

Operazioni successive

A questo punto si conoscono diverse tecniche per la gestione dell'input in WPF. Si dovrebbe anche avere una conoscenza più approfondita dei vari tipi di eventi di input e dei meccanismi degli eventi indirizzati usati in WPF.

Sono disponibili risorse aggiuntive che illustrano più dettagliatamente gli elementi del framework WPF e il routing degli eventi. Per altre informazioni generali, vedere [Cenni preliminari sull'esecuzione di comandi](#), [Cenni preliminari sullo stato attivo](#), [Cenni preliminari sugli elementi di base](#), [Strutture ad albero in WPF](#) e [Cenni preliminari sugli eventi indirizzati](#).

Vedere anche

- [Panoramica sullo stato attivo](#)
- [Panoramica sull'esecuzione di comandi](#)
- [Cenni preliminari sugli eventi indirizzati](#)
- [Cenni preliminari sugli elementi di base](#)
- [Proprietà](#)

Cenni preliminari sull'esecuzione di comandi

23/11/2019 • 24 minutes to read • [Edit Online](#)

I comandi sono un meccanismo di input in Windows Presentation Foundation (WPF) che consente la gestione dell'input a un livello più semantico rispetto all'input dei dispositivi. Sono esempi di comandi le operazioni **Copia**, **Taglia** e **Incolla** disponibili in molte applicazioni.

Questa panoramica presenta i comandi disponibili in WPF, le classi che fanno parte del modello di esecuzione dei comandi e le procedure per usare e creare comandi nelle applicazioni.

Di seguito sono elencate le diverse sezioni di questo argomento:

- [Cosa sono i comandi?](#)
- [Esempio di comando semplice in WPF](#)
- [Quattro concetti principali dell'esecuzione dei comandi WPF](#)
- [Libreria dei comandi](#)
- [Creazione di comandi personalizzati](#)

Cosa sono i comandi?

I comandi hanno diversi scopi. Il primo scopo è di separare la semantica e l'oggetto che richiama un comando dalla logica che esegue il comando. In questo modo, più codici sorgente diversi possono richiamare la stessa logica di comando che può quindi essere personalizzata per obiettivi differenti. Le operazioni di modifica **Copia**, **Taglia** e **Incolla**, ad esempio, disponibili in molte applicazioni, possono essere richiamate usando azioni dell'utente diverse se vengono implementate tramite i comandi. Un'applicazione potrebbe consentire a un utente di tagliare gli oggetti o il testo selezionati facendo clic su un pulsante, scegliendo una voce da un menu o usando una combinazione di tasti, ad esempio CTRL+X. Usando i comandi è possibile associare ogni tipo di azione utente alla stessa logica.

Un altro scopo dei comandi è di indicare se un'azione è disponibile. Per continuare con l'esempio in cui viene tagliato un oggetto o un testo, l'azione ha senso solo quando viene selezionato un elemento. Se un utente tenta di tagliare un oggetto o un testo senza avere selezionato alcun elemento, non viene eseguita alcuna operazione. Per indicarlo all'utente, in molte applicazioni vengono disabilitati i pulsanti e le voci di menu in modo che l'utente sappia se è possibile eseguire un'azione. Un comando può indicare se un'azione è possibile implementando il metodo [CanExecute](#). Un pulsante può sottoscrivere l'evento [CanExecuteChanged](#) ed essere disabilitato se [CanExecute](#) restituisce `false` oppure essere abilitato se [CanExecute](#) restituisce `true`.

La semantica di un comando può essere coerente tra le applicazioni e le classi, tuttavia la logica dell'azione è specifica dell'oggetto particolare su cui si agisce. La combinazione di tasti CTRL+X richiama il comando **Taglia** nelle classi di testo, nelle classi di immagine e nei Web browser, tuttavia la logica effettiva per l'esecuzione dell'operazione **Taglia** viene definita dall'applicazione in cui si esegue l'operazione di taglio. [RoutedCommand](#) consente ai client di implementare la logica. Un oggetto testo può tagliare il testo selezionato negli Appunti, mentre un oggetto immagine può tagliare l'immagine selezionata. Quando un'applicazione gestisce l'evento [Executed](#), può accedere alla destinazione del comando ed eseguire l'azione appropriata in base al tipo della destinazione.

Esempio di comando semplice in WPF

Il modo più semplice per usare un comando in WPF consiste nell'uso di un oggetto [RoutedCommand](#)

predefinito di una delle classi della libreria dei comandi, nell'uso di un controllo con supporto nativo per gestire il comando e nell'uso di un controllo con supporto nativo per richiamare un comando. Il comando **Paste** è uno dei comandi predefiniti della classe [ApplicationCommands](#). Il controllo **TextBox** è dotato di logica incorporata per la gestione del comando **Paste**. E la classe **MenuItem** è dotata del supporto nativo per la chiamata di comandi.

L'esempio seguente illustra come impostare un oggetto **MenuItem** in modo che quando si fa clic su di esso, richiami il comando **Paste** in un **TextBox**, supponendo che **TextBox** abbia lo stato attivo della tastiera.

```
<StackPanel>
    <Menu>
        <MenuItem Command="ApplicationCommands.Paste" />
    </Menu>
    <TextBox />
</StackPanel>
```

```
// Creating the UI objects
StackPanel mainStackPanel = new StackPanel();
TextBox pasteTextBox = new TextBox();
Menu stackPanelMenu = new Menu();
MenuItem pasteMenuItem = new MenuItem();

// Adding objects to the panel and the menu
stackPanelMenu.Items.Add(pasteMenuItem);
mainStackPanel.Children.Add(stackPanelMenu);
mainStackPanel.Children.Add(pasteTextBox);

// Setting the command to the Paste command
pasteMenuItem.Command = ApplicationCommands.Paste;

// Setting the command target to the TextBox
pasteMenuItem.CommandTarget = pasteTextBox;
```

```
' Creating the UI objects
Dim mainStackPanel As New StackPanel()
Dim pasteTextBox As New TextBox()
Dim stackPanelMenu As New Menu()
Dim pasteMenuItem As New MenuItem()

' Adding objects to the panel and the menu
stackPanelMenu.Items.Add(pasteMenuItem)
mainStackPanel.Children.Add(stackPanelMenu)
mainStackPanel.Children.Add(pasteTextBox)

' Setting the command to the Paste command
pasteMenuItem.Command = ApplicationCommands.Paste
```

Quattro concetti principali dell'esecuzione dei comandi WPF

Il modello di comando indirizzato di WPF può essere suddiviso in quattro concetti principali: il comando, l'origine del comando, la destinazione del comando e l'associazione del comando:

- Il *comando* è l'azione da eseguire.
- L'*origine del comando* è l'oggetto che richiama il comando.
- La *destinazione del comando* è l'oggetto su cui viene eseguito il comando.
- L'*associazione del comando* è l'oggetto che esegue il mapping della logica di comando al comando.

Nell'esempio precedente, il comando [Paste](#) corrisponde al comando, [MenuItem](#) è l'origine del comando, [TextBox](#) è la destinazione del comando e l'associazione del comando viene fornita dal controllo [TextBox](#). È importante notare che non sempre l'oggetto [CommandBinding](#) viene fornito dal controllo che rappresenta la classe di destinazione del comando. Molto spesso è lo sviluppatore dell'applicazione che deve creare [CommandBinding](#). In altri casi, può esistere un collegamento tra [CommandBinding](#) e un predecessore della destinazione del comando.

Commands

I comandi in WPF vengono creati tramite l'implementazione dell'interfaccia [ICommand](#). [ICommand](#) espone due metodi, [Execute](#) e [CanExecute](#), e un evento, [CanExecuteChanged](#). [Execute](#) esegue le azioni associate al comando. [CanExecute](#) determina se il comando può essere eseguito nella destinazione corrente del comando stesso. [CanExecuteChanged](#) viene generato se il gestore che centralizza le operazioni di comando rileva una modifica nell'origine del comando che potrebbe invalidare un comando generato ma non ancora eseguito dall'associazione del comando. L'implementazione WPF di [ICommand](#) è la classe [RoutedCommand](#) e costituisce l'obiettivo di questa panoramica.

Le origini di input principali di WPF sono il mouse, la tastiera, l'input penna e i comandi indirizzati. La maggior parte degli input orientati al dispositivo usano un oggetto [RoutedEvent](#) per notificare agli oggetti di una pagina dell'applicazione che si è verificato un evento di input. [RoutedCommand](#) non è diverso. I metodi [Execute](#) e [CanExecute](#) di [RoutedCommand](#) non contengono la logica dell'applicazione per il comando, ma generano eventi indirizzati che effettuano il tunneling e il bubbling attraverso la struttura ad albero degli elementi finché non rilevano un oggetto con [CommandBinding](#). [CommandBinding](#) contiene i gestori per questi eventi e sono i gestori a eseguire il comando. Per altre informazioni sul routing di eventi in WPF, vedere [Cenni preliminari sugli eventi indirizzati](#).

Il metodo [Execute](#) in un [RoutedCommand](#) genera gli eventi [PreviewExecuted](#) e [Executed](#) nella destinazione del comando. Il metodo [CanExecute](#) in un [RoutedCommand](#) genera gli eventi [CanExecute](#) e [PreviewCanExecute](#) nella destinazione del comando. Questi eventi eseguono il tunneling e il bubbling nell'albero degli elementi finché non incontrano un oggetto con un [CommandBinding](#) per quel particolare comando.

WPF fornisce un set di comandi indirizzati comuni distribuito in diverse classi: [MediaCommands](#), [ApplicationCommands](#), [NavigationCommands](#), [ComponentCommands](#) e [EditingCommands](#). Queste classi sono costituite solo da oggetti [RoutedCommand](#) e non dalla logica di implementazione del comando. La logica di implementazione è responsabilità dell'oggetto sul quale viene eseguito il comando.

Origini dei comandi

Un'origine del comando è l'oggetto che richiama il comando. [MenuItem](#), [Button](#) e [KeyGesture](#) sono esempi di origini del comando.

Le origini del comando in WPF in genere implementano l'interfaccia [ICommandSource](#).

[ICommandSource](#) espone tre proprietà: [Command](#), [CommandTarget](#) e [CommandParameter](#):

- [Command](#) è il comando da eseguire quando viene richiamata l'origine del comando.
- [CommandTarget](#) è l'oggetto su cui eseguire il comando. Si noti che in WPF la proprietà [CommandTarget](#) in [ICommandSource](#) è applicabile solo quando [ICommand](#) è [RoutedCommand](#). Se [CommandTarget](#) è impostato su un [ICommandSource](#) e il comando corrispondente non è un [RoutedCommand](#), la destinazione del comando viene ignorata. Se [CommandTarget](#) non è impostato, viene rappresentato dall'elemento con lo stato attivo della tastiera.
- [CommandParameter](#) è un tipo di dati definito dall'utente usato per passare informazioni ai gestori che implementano il comando.

Le classi WPF che implementano [ICommandSource](#) sono [ButtonBase](#), [MenuItem](#), [Hyperlink](#) e [InputBinding](#).

[ButtonBase](#), [MenuItem](#) e [Hyperlink](#) richiamano un comando quando ricevono un clic e [InputBinding](#) richiama un comando quando l'oggetto [InputGesture](#) associato viene eseguito.

L'esempio seguente illustra come usare un oggetto [MenuItem](#) in un oggetto [ContextMenu](#) come origine del comando [Properties](#).

```
<StackPanel>
  <StackPanel.ContextMenu>
    <ContextMenu>
      <MenuItem Command="ApplicationCommands.Properties" />
    </ContextMenu>
  </StackPanel.ContextMenu>
</StackPanel>
```

```
StackPanel cmdSourcePanel = new StackPanel();
ContextMenu cmdSourceContextMenu = new ContextMenu();
MenuItem cmdSourceMenuItem = new MenuItem();

// Add ContextMenu to the StackPanel.
cmdSourcePanel.ContextMenu = cmdSourceContextMenu;
cmdSourcePanel.ContextMenu.Items.Add(cmdSourceMenuItem);

// Associate Command with MenuItem.
cmdSourceMenuItem.Command = ApplicationCommands.Properties;
```

```
Dim cmdSourcePanel As New StackPanel()
Dim cmdSourceContextMenu As New ContextMenu()
Dim cmdSourceMenuItem As New MenuItem()

' Add ContextMenu to the StackPanel.
cmdSourcePanel.ContextMenu = cmdSourceContextMenu
cmdSourcePanel.ContextMenu.Items.Add(cmdSourceMenuItem)

' Associate Command with MenuItem.
cmdSourceMenuItem.Command = ApplicationCommands.Properties
```

Un'origine del comando è in genere in ascolto dell'evento [CanExecuteChanged](#). Questo evento informa l'origine del comando relativamente alla probabile modifica della capacità del comando di essere eseguito sulla destinazione del comando corrente. L'origine del comando può eseguire una query dello stato corrente dell'oggetto [RoutedCommand](#) tramite il metodo [CanExecute](#). L'origine comando può quindi disabilitarsi, se l'esecuzione del comando non riesce. Un esempio è un oggetto [MenuItem](#) che si disattiva quando non è possibile eseguire il comando corrispondente.

Un oggetto [InputGesture](#) può essere usato come origine del comando. Due tipi di movimento di input in WPF sono [KeyGesture](#) e [MouseGesture](#). È possibile considerare un oggetto [KeyGesture](#) come tasto di scelta rapida, ad esempio CTRL + C. Un [KeyGesture](#) è costituito da un [Key](#) e da un set di [ModifierKeys](#). Un [MouseGesture](#) è costituito da un [MouseAction](#) e da un set di [ModifierKeys](#) facoltativo.

Affinché un oggetto [InputGesture](#) funzioni come origine del comando, deve essere associato a un comando. Questa operazione può essere eseguita in diversi modi. Un modo consiste nell'uso di un oggetto [InputBinding](#).

L'esempio seguente illustra come creare un [KeyBinding](#) tra un [KeyGesture](#) e un [RoutedCommand](#).

```
<Window.InputBindings>
  <KeyBinding Key="B"
    Modifiers="Control"
    Command="ApplicationCommands.Open" />
</Window.InputBindings>
```

```
KeyGesture OpenKeyGesture = new KeyGesture(
  Key.B,
  ModifierKeys.Control);

KeyBinding OpenCmdKeybinding = new KeyBinding(
  ApplicationCommands.Open,
  OpenKeyGesture);

this.InputBindings.Add(OpenCmdKeybinding);
```

```
Dim OpenKeyGesture As New KeyGesture(Key.B, ModifierKeys.Control)

Dim OpenCmdKeybinding As New KeyBinding(ApplicationCommands.Open, OpenKeyGesture)

Me.InputBindings.Add(OpenCmdKeybinding)
```

Un altro modo per associare un [InputGesture](#) a un [RoutedCommand](#) consiste nell'aggiungere l'oggetto [InputGesture](#) a [InputGestureCollection](#) per [RoutedCommand](#).

L'esempio seguente illustra come aggiungere un [KeyGesture](#) alla raccolta [InputGestureCollection](#) di un [RoutedCommand](#).

```
KeyGesture OpenCmdKeyGesture = new KeyGesture(
  Key.B,
  ModifierKeys.Control);

ApplicationCommands.Open.InputGestures.Add(OpenCmdKeyGesture);
```

```
Dim OpenCmdKeyGesture As New KeyGesture(Key.B, ModifierKeys.Control)

ApplicationCommands.Open.InputGestures.Add(OpenCmdKeyGesture)
```

CommandBinding

Una classe [CommandBinding](#) associa un comando ai gestori degli eventi che implementano il comando.

La classe [CommandBinding](#) contiene la proprietà [Command](#) e gli eventi [PreviewExecuted](#), [Executed](#), [PreviewCanExecute](#) e [CanExecute](#).

[Command](#) è il comando a cui [CommandBinding](#) è associato. I gestori degli eventi associati agli eventi [PreviewExecuted](#) e [Executed](#) implementano la logica di comando. I gestori eventi associati agli eventi [PreviewCanExecute](#) e [CanExecute](#) determinano se il comando può essere eseguito nella destinazione corrente.

L'esempio seguente illustra come creare un [CommandBinding](#) in [Window](#) radice di un'applicazione. [CommandBinding](#) associa il comando [Open](#) ai gestori [Executed](#) e [CanExecute](#).

```
<Window.CommandBindings>
    <CommandBinding Command="ApplicationCommands.Open"
        Executed="OpenCmdExecuted"
        CanExecute="OpenCmdCanExecute"/>
</Window.CommandBindings>
```

```
// Creating CommandBinding and attaching an Executed and CanExecute handler
CommandBinding OpenCmdBinding = new CommandBinding(
    ApplicationCommands.Open,
    OpenCmdExecuted,
    OpenCmdCanExecute);

this.CommandBindings.Add(OpenCmdBinding);
```

```
' Creating CommandBinding and attaching an Executed and CanExecute handler
Dim OpenCmdBinding As New CommandBinding(ApplicationCommands.Open, AddressOf OpenCmdExecuted, AddressOf
OpenCmdCanExecute)

Me.CommandBindings.Add(OpenCmdBinding)
```

Successivamente vengono creati [ExecutedRoutedEventHandler](#) e [CanExecuteRoutedEventHandler](#). [ExecutedRoutedEventHandler](#) apre un oggetto [MessageBox](#) che visualizza una stringa che indica che il comando è stato eseguito. Il metodo [CanExecuteRoutedEventHandler](#) imposta la proprietà [CanExecute](#) su `true`.

```
void OpenCmdExecuted(object target, ExecutedRoutedEventArgs e)
{
    String command, targetobj;
    command = ((RoutedCommand)e.Command).Name;
    targetobj = ((FrameworkElement)target).Name;
    MessageBox.Show("The " + command + " command has been invoked on target object " + targetobj);
}
```

```
Private Sub OpenCmdExecuted(ByVal sender As Object, ByVal e As ExecutedRoutedEventArgs)
    Dim command, targetobj As String
    command = CType(e.Command, RoutedCommand).Name
    targetobj = CType(sender, FrameworkElement).Name
    MessageBox.Show("The " + command + " command has been invoked on target object " + targetobj)
End Sub
```

```
void OpenCmdCanExecute(object sender, CanExecuteRoutedEventArgs e)
{
    e.CanExecute = true;
}
```

```
Private Sub OpenCmdCanExecute(ByVal sender As Object, ByVal e As CanExecuteRoutedEventArgs)
    e.CanExecute = True
End Sub
```

Un oggetto [CommandBinding](#) viene collegato a un oggetto specifico, ad esempio all'oggetto [Window](#) radice dell'applicazione o di un controllo. L'oggetto a cui è collegato l'oggetto [CommandBinding](#) definisce l'ambito dell'associazione. È ad esempio possibile raggiungere un oggetto [CommandBinding](#) collegato a un predecessore della destinazione del comando tramite l'evento [Executed](#), ma un oggetto [CommandBinding](#) collegato a un discendente della destinazione del comando non è raggiungibile. Si tratta di una conseguenza

diretta del modo in cui un oggetto [RoutedEvent](#) effettua il tunneling e il bubbling dall'oggetto che genera l'evento.

In alcune situazioni l'oggetto [CommandBinding](#) è collegato alla destinazione del comando stessa, ad esempio con la classe [TextBox](#) e i comandi [Cut](#), [Copy](#) e [Paste](#). Tuttavia, spesso è consigliabile associare l'oggetto [CommandBinding](#) a un predecessore della destinazione del comando, ad esempio l'oggetto [Window](#) principale o l'oggetto Application, specialmente se lo stesso oggetto [CommandBinding](#) può essere usato per più destinazioni di comandi. Si tratta di decisioni di progettazione di cui tenere conto quando si crea l'infrastruttura di esecuzione dei comandi.

Destinazione del comando

La destinazione del comando è l'elemento su cui viene eseguito il comando. Per quanto riguarda [RoutedCommand](#), la destinazione del comando è l'elemento in corrispondenza del quale inizia il routing di [Executed](#) e [CanExecute](#). Come notato in precedenza, in WPF la proprietà [CommandTarget](#) in [ICommandSource](#) è applicabile solo quando [ICommand](#) è un oggetto [RoutedCommand](#). Se [CommandTarget](#) è impostato su un [ICommandSource](#) e il comando corrispondente non è un [RoutedCommand](#), la destinazione del comando viene ignorata.

L'origine del comando può impostare in modo esplicito la destinazione del comando. Se la destinazione del comando non viene definita, l'elemento con lo stato attivo verrà usato come destinazione del comando. Uno dei vantaggi dell'uso dell'elemento con lo stato attivo come destinazione del comando consiste nel fatto che consente allo sviluppatore di applicazioni di usare la stessa origine del comando per richiamare un comando su più destinazioni senza tenere traccia della destinazione del comando. Se, ad esempio, un oggetto [MenuItem](#) richiama il comando [Incolla](#) in un'applicazione che ha un controllo [TextBox](#) e un controllo [PasswordBox](#), la destinazione può essere l'oggetto [TextBox](#) o l'oggetto [PasswordBox](#) a seconda del controllo con stato attivo.

L'esempio seguente mostra come impostare in modo esplicito la destinazione del comando nel markup e nel code-behind.

```
<StackPanel>
    <Menu>
        <MenuItem Command="ApplicationCommands.Paste"
                  CommandTarget="{Binding ElementName=mainTextBox}" />
    </Menu>
    <TextBox Name="mainTextBox"/>
</StackPanel>
```

```
// Creating the UI objects
StackPanel mainStackPanel = new StackPanel();
TextBox pasteTextBox = new TextBox();
Menu stackPanelMenu = new Menu();
MenuItem pasteMenuItem = new MenuItem();

// Adding objects to the panel and the menu
stackPanelMenu.Items.Add(pasteMenuItem);
mainStackPanel.Children.Add(stackPanelMenu);
mainStackPanel.Children.Add(pasteTextBox);

// Setting the command to the Paste command
pasteMenuItem.Command = ApplicationCommands.Paste;

// Setting the command target to the TextBox
pasteMenuItem.CommandTarget = pasteTextBox;
```

```

' Creating the UI objects
Dim mainStackPanel As New StackPanel()
Dim pasteTextBox As New TextBox()
Dim stackPanelMenu As New Menu()
Dim pasteMenuItem As New MenuItem()

' Adding objects to the panel and the menu
stackPanelMenu.Items.Add(pasteMenuItem)
mainStackPanel.Children.Add(stackPanelMenu)
mainStackPanel.Children.Add(pasteTextBox)

' Setting the command to the Paste command
pasteMenuItem.Command = ApplicationCommands.Paste

```

CommandManager

[CommandManager](#) svolge un certo numero di funzioni correlate a comandi. Rende disponibile un set di metodi statici per l'aggiunta e la rimozione di gestori degli eventi [PreviewExecuted](#), [Executed](#), [PreviewCanExecute](#) e [CanExecute](#) da e verso un elemento specifico. Rappresenta un modo per registrare oggetti [CommandBinding](#) e [InputBinding](#) in una classe specifica. [CommandManager](#) consente anche, tramite l'evento [RequerySuggested](#), di notificare a un comando quando deve essere generato l'evento [CanExecuteChanged](#).

Il metodo [InvalidateRequerySuggested](#) impone a [CommandManager](#) di generare l'evento [RequerySuggested](#). Ciò è utile per le condizioni che richiedono l'abilitazione o la disabilitazione di un comando ma che l'oggetto [CommandManager](#) non è in grado di rilevare.

Libreria dei comandi

WPF offre un set di comandi predefiniti. La libreria dei comandi è costituita dalle classi seguenti: [ApplicationCommands](#), [NavigationCommands](#), [MediaCommands](#), [EditingCommands](#) e [ComponentCommands](#). Queste classi forniscono comandi quali [Cut](#), [BrowseBack](#) e [BrowseForward](#), [Play](#), [Stop](#) e [Pause](#).

Molti di questi comandi includono un set di associazioni di input predefinite. Se ad esempio si specifica che l'applicazione gestisce il comando copy, si ottiene automaticamente il tasto di scelta rapida "CTRL + C". si ottengono anche binding per altri dispositivi di input, ad esempio i movimenti di penna di Tablet PC e le informazioni vocali.

Quando si fa riferimento ai comandi delle diverse librerie dei comandi usando XAML, in genere è possibile omettere il nome della classe di libreria che espone la proprietà del comando statico. Di solito i nomi dei comandi sono stringhe non ambigue ed esistono tipi proprietari che forniscono un raggruppamento logico di comandi, ma che non sono necessari per la risoluzione dell'ambiguità. Ad esempio, è possibile specificare `command="Cut"` anziché il più dettagliato `command="ApplicationCommands.Cut"`. Si tratta di un meccanismo utile incorporato nel processore WPF XAML per i comandi. Più precisamente, si tratta di un comportamento del convertitore dei tipi di [ICommand](#) al quale il processore WPF XAML fa riferimento in fase di caricamento.

Creazione di comandi personalizzati

Se i comandi delle classi della libreria dei comandi non soddisfano le proprie esigenze, è possibile creare comandi personalizzati. Questa operazione può essere eseguita in due modi. Il primo prevede la progettazione da zero e l'implementazione dell'interfaccia [ICommand](#). L'altro, che rappresenta l'approccio più comune, consiste nel creare un oggetto [RoutedCommand](#) o [RoutedUICommand](#).

Per un esempio di creazione di un oggetto [RoutedCommand](#) personalizzato, vedere [Create a Custom RoutedCommand Sample](#) (Creare un esempio di oggetto RoutedCommand personalizzato).

Vedere anche

- [RoutedCommand](#)
- [CommandBinding](#)
- [InputBinding](#)
- [CommandManager](#)
- [Cenni preliminari sull'input](#)
- [Cenni preliminari sugli eventi indirizzati](#)
- [Implementare ICommandSource](#)
- [How to: Add a Command to a MenuItem \(Procedura: Aggiungere un comando a un MenuItem\)](#)
- [Create a Custom RoutedCommand Sample \(Creare un esempio di oggetto RoutedCommand personalizzato\)](#)

Cenni preliminari sullo stato attivo

23/10/2019 • 14 minutes to read • [Edit Online](#)

In WPF lo stato attivo è basato su due concetti principali, lo stato attivo della tastiera e lo stato attivo logico. Lo stato attivo della tastiera fa riferimento all'elemento che riceve l'input della tastiera, mentre lo stato attivo logico fa riferimento all'elemento di un ambito che ha ricevuto lo stato attivo. In questa panoramica verranno illustrati in dettaglio questi concetti. Comprendere le differenze esistenti tra questi concetti è fondamentale per la creazione di applicazioni complesse costituite da più aree in cui è possibile ottenere lo stato attivo.

Le principali classi che fanno parte di gestione dello stato attivo sono le [Keyboard](#) (classe), il [FocusManager](#) classe e l'elemento di base, classi, ad esempio [UIElement](#) e [ContentElement](#). Per altre informazioni sugli elementi di base, vedere [Cenni preliminari sugli elementi di base](#).

Il [Keyboard](#) classe riguarda principalmente lo stato attivo della tastiera e [FocusManager](#) riguarda principalmente con lo stato attivo logico, ma questo non è una differenza assoluta. Un elemento con lo stato attivo della tastiera presenta anche lo stato attivo logico, mentre un elemento che ha ottenuto lo stato attivo logico non ha necessariamente lo stato attivo della tastiera. Questo è evidente quando si usa il [Keyboard](#) classe per impostare l'elemento con lo stato attivo della tastiera, per tale imposta inoltre lo stato attivo logico sull'elemento.

Stato attivo della tastiera

Lo stato attivo della tastiera fa riferimento all'elemento che riceve l'input dalla tastiera. Su un desktop può esserci un solo elemento con lo stato attivo della tastiera. Nelle WPF, l'elemento con lo stato attivo della tastiera avrà [IsKeyboardFocused](#) impostato su `true`. La proprietà statica [FocusedElement](#) sul [Keyboard](#) classe ottiene l'elemento che attualmente ha lo stato attivo della tastiera.

Affinché un elemento ottenga lo stato attivo della tastiera, il [Focusable](#) e il [IsVisible](#) delle proprietà degli elementi di base devono essere impostate su `true`. Alcune classi, ad esempio la [Panel](#) classe di base, avere [Focusable](#) impostata su `false` per impostazione predefinita; pertanto, è necessario impostare [Focusable](#) a `true` se si desidera che l'elemento possa ottenere lo stato attivo della tastiera.

Lo stato attivo della tastiera può essere ottenuto tramite interazione dell'utente con l'Interfaccia utente, ad esempio usando il tasto TAB per spostarsi su un elemento o facendo clic con il mouse su alcuni elementi. Lo stato attivo della tastiera può anche essere ottenuto a livello di codice usando il [Focus](#) metodo su di [Keyboard](#) classe. Il [Focus](#) metodo tenta di assegnare lo stato attivo della tastiera di elemento specificato. L'elemento restituito è l'elemento con lo stato attivo della tastiera e potrebbe non essere quello richiesto se l'oggetto stato attivo nuovo o quello precedente blocca la richiesta.

L'esempio seguente usa il [Focus](#) per impostare lo stato attivo della tastiera su un [Button](#).

```
private void OnLoaded(object sender, RoutedEventArgs e)
{
    // Sets keyboard focus on the first Button in the sample.
    Keyboard.Focus(firstButton);
}
```

```
Private Sub OnLoaded(ByVal sender As Object, ByVal e As RoutedEventArgs)
    ' Sets keyboard focus on the first Button in the sample.
    Keyboard.Focus(firstButton)
End Sub
```

Il [IsKeyboardFocused](#) proprietà sulle classi di elementi di base Ottiene un valore che indica se l'elemento ha lo stato attivo della tastiera. Il [IsKeyboardFocusWithin](#) proprietà sulle classi di elementi di base Ottiene un valore che indica se l'elemento o uno qualsiasi dei relativi elementi figlio visivi ha lo stato attivo della tastiera.

Quando si imposta lo stato attivo iniziale all'avvio dell'applicazione, l'elemento che riceve lo stato attivo deve essere nella struttura visiva della finestra iniziale caricata dall'applicazione e l'elemento deve avere [Focusable](#) e [IsVisible](#) impostato su `true`. La posizione consigliata per impostare lo stato attivo iniziale è la [Loaded](#) gestore dell'evento. Oggetto [Dispatcher](#) callback nonché chiamando [Invoke](#) o [BeginInvoke](#).

Stato attivo logico

Lo stato attivo logico fa riferimento al [FocusManager.FocusedElement](#) nell'ambito di stato attivo. Ambito di stato attivo è un elemento che tiene traccia del [FocusedElement](#) nel relativo ambito. Quando lo stato attivo della tastiera lascia un ambito di stato attivo, l'elemento con lo stato attivo perde lo stato attivo della tastiera, ma mantiene quello logico. Quando lo stato attivo della tastiera torna nell'ambito di stato attivo, l'elemento con lo stato attivo ottiene lo stato attivo della tastiera. In questo modo, lo stato attivo della tastiera può essere passato tra più ambiti, ma l'elemento con lo stato attivo nell'ambito di stato attivo ottiene sicuramente di nuovo lo stato attivo della tastiera quando lo stato attivo ritorna all'ambito di stato attivo.

In un'applicazione possono esserci più elementi con lo stato attivo logico, ma in un determinato ambito di stato attivo può esserci un solo elemento con lo stato attivo logico.

Un elemento con stato attivo della tastiera ha lo stato attivo logico per l'ambito a cui appartiene.

Un elemento può essere convertito in un ambito di stato attivo Extensible Application Markup Language (XAML) impostando la [FocusManager](#) proprietà associata [IsFocusScope](#) a `true`. Nel codice, un elemento può essere convertito in un ambito dello stato attivo chiamando [SetIsFocusScope](#).

Nell'esempio seguente viene eseguita una [StackPanel](#) ambito di stato attivo impostando il [IsFocusScope](#) proprietà associata.

```
<StackPanel Name="focusScope1"
    FocusManager.IsFocusScope="True"
    Height="200" Width="200">
    <Button Name="button1" Height="50" Width="50"/>
    <Button Name="button2" Height="50" Width="50"/>
</StackPanel>
```

```
StackPanel focusScope2 = new StackPanel();
FocusManager.SetIsFocusScope(focusScope2, true);
```

```
Dim focusScope2 As New StackPanel()
FocusManager.SetIsFocusScope(focusScope2, True)
```

[GetFocusScope](#) Restituisce l'ambito dello stato attivo per l'elemento specificato.

Le classi nello WPF quali sono gli ambiti di stato attivo per impostazione predefinita viene [Window](#), [MenuItem](#), [ToolBar](#), e [ContextMenu](#).

[GetFocusedElement](#) Ottiene l'elemento con lo stato attivo per l'ambito specificato. [SetFocusedElement](#) Imposta l'elemento con lo stato attivo nell'ambito dello stato attivo specificato. [SetFocusedElement](#) in genere utilizzato per impostare l'elemento con lo stato attivo iniziale.

L'esempio seguente illustra come impostare e ottenere l'elemento con lo stato attivo in un ambito di stato attivo.

```
// Sets the focused element in focusScope1
// focusScope1 is a StackPanel.
FocusManager.SetFocusedElement(focusScope1, button2);

// Gets the focused element for focusScope 1
IInputElement focusedElement = FocusManager.GetFocusedElement(focusScope1);
```

```
' Sets the focused element in focusScope1
' focusScope1 is a StackPanel.
FocusManager.SetFocusedElement(focusScope1, button2)

' Gets the focused element for focusScope 1
Dim focusedElement As IInputElement = FocusManager.GetFocusedElement(focusScope1)
```

Navigazione tramite tastiera

Il [KeyboardNavigation](#) classe è responsabile dell'implementazione di navigazione dello stato attivo della tastiera predefinita quando viene premuto uno dei tasti di navigazione. I tasti di navigazione sono: Chiavi TAB, MAIUSC + TAB, CTRL + TAB, CTRL + MAIUSC + TAB, freccia su, freccia giù, freccia sinistra e freccia destra.

Il comportamento di navigazione di un contenitore di navigazione può essere modificato impostando l'oggetto associato [KeyboardNavigation](#) delle proprietà [TabNavigation](#), [ControlTabNavigation](#), e [DirectionalNavigation](#). Queste proprietà sono di tipo [KeyboardNavigationMode](#) e i valori possibili sono [Continue](#), [Local](#), [Contained](#), [Cycle](#), [Once](#), e [None](#). Il valore predefinito è [Continue](#), che indica l'elemento non è un contenitore di navigazione.

L'esempio seguente crea una [Menu](#) con un numero di [MenuItem](#) oggetti. Il [TabNavigation](#) proprietà associata è impostata su [Cycle](#) nel [Menu](#). Quando viene modificato lo stato attivo usando il tasto tab all'interno di [Menu](#), lo stato attivo passerà da ogni elemento e restituisce quando viene raggiunto l'ultimo elemento lo stato attivo al primo elemento.

```
<Menu KeyboardNavigation.TabNavigation="Cycle">
  <MenuItem Header="Menu Item 1" />
  <MenuItem Header="Menu Item 2" />
  <MenuItem Header="Menu Item 3" />
  <MenuItem Header="Menu Item 4" />
</Menu>
```

```
Menu navigationMenu = new Menu();
MenuItem item1 = new MenuItem();
MenuItem item2 = new MenuItem();
MenuItem item3 = new MenuItem();
MenuItem item4 = new MenuItem();

navigationMenu.Items.Add(item1);
navigationMenu.Items.Add(item2);
navigationMenu.Items.Add(item3);
navigationMenu.Items.Add(item4);

KeyboardNavigation.SetTabNavigation(navigationMenu,
  KeyboardNavigationMode.Cycle);
```

```

Dim navigationMenu As New Menu()
Dim item1 As New MenuItem()
Dim item2 As New MenuItem()
Dim item3 As New MenuItem()
Dim item4 As New MenuItem()

navigationMenu.Items.Add(item1)
navigationMenu.Items.Add(item2)
navigationMenu.Items.Add(item3)
navigationMenu.Items.Add(item4)

KeyboardNavigation.SetTabNavigation(navigationMenu, KeyboardNavigationMode.Cycle)

```

Spostamento dello stato attivo a livello di codice

API aggiuntive per lavorare con lo stato attivo sono [MoveFocus](#) e [PredictFocus](#).

[MoveFocus](#) le modifica dello stato attivo all'elemento successivo nell'applicazione. Oggetto [TraversalRequest](#) viene usato per specificare la direzione. Il [FocusNavigationDirection](#) passato a [MoveFocus](#) specifica lo stato attivo direzioni diverse può essere spostato, ad esempio [First](#), [Last](#), [Up](#) e [Down](#).

L'esempio seguente usa [MoveFocus](#) per modificare l'elemento con lo stato attivo.

```

// Creating a FocusNavigationDirection object and setting it to a
// local field that contains the direction selected.
FocusNavigationDirection focusDirection = _focusMoveValue;

// MoveFocus takes a TraversalRequest as its argument.
TraversalRequest request = new TraversalRequest(focusDirection);

// Gets the element with keyboard focus.
UIElement elementWithFocus = Keyboard.FocusedElement as UIElement;

// Change keyboard focus.
if (elementWithFocus != null)
{
    elementWithFocus.MoveFocus(request);
}

```

```

' Creating a FocusNavigationDirection object and setting it to a
' local field that contains the direction selected.
Dim focusDirection As FocusNavigationDirection = _focusMoveValue

' MoveFocus takes a TraversalRequest as its argument.
Dim request As New TraversalRequest(focusDirection)

' Gets the element with keyboard focus.
Dim elementWithFocus As UIElement = TryCast(Keyboard.FocusedElement, UIElement)

' Change keyboard focus.
If elementWithFocus IsNot Nothing Then
    elementWithFocus.MoveFocus(request)
End If

```

[PredictFocus](#) Restituisce l'oggetto che riceverebbe lo stato attivo se venisse modificato lo stato attivo.

Attualmente, solo [Up](#), [Down](#), [Left](#), e [Right](#) supportati da [PredictFocus](#).

Eventi di stato attivo

Gli eventi correlati allo stato attivo della tastiera sono [PreviewGotKeyboardFocus](#), [GotKeyboardFocus](#) e

[PreviewLostKeyboardFocus](#), [LostKeyboardFocus](#). Gli eventi sono definiti come eventi associati nel [Keyboard](#) classe, ma sono più facilmente accessibili come eventi indirizzati equivalenti sulle classi di elementi di base. Per altre informazioni sugli eventi, vedere [Cenni preliminari sugli eventi indirizzati](#).

[GotKeyboardFocus](#) viene generato quando l'elemento otterrà lo stato attivo della tastiera. [LostKeyboardFocus](#) viene generato quando l'elemento perde lo stato attivo della tastiera. Se il [PreviewGotKeyboardFocus](#) evento o la [PreviewLostKeyboardFocusEvent](#) viene gestito l'evento e [Handled](#) è impostata su `true`, quindi non modificherà lo stato attivo.

L'esempio seguente collega [GotKeyboardFocus](#) e [LostKeyboardFocus](#) gestori eventi per un [TextBox](#).

```
<Border BorderBrush="Black" BorderThickness="1"
        Width="200" Height="100" Margin="5">
    <StackPanel>
        <Label HorizontalAlignment="Center" Content="Type Text In This TextBox" />
        <TextBox Width="175"
                Height="50"
                Margin="5"
                TextWrapping="Wrap"
                HorizontalAlignment="Center"
                VerticalScrollBarVisibility="Auto"
                GotKeyboardFocus="TextBoxGotKeyboardFocus"
                LostKeyboardFocus="TextBoxLostKeyboardFocus"
                KeyDown="SourceTextKeyDown"/>
    </StackPanel>
</Border>
```

Quando la [TextBox](#) otterrà lo stato attivo della tastiera, il [Background](#) proprietà delle [TextBox](#) viene modificato in [LightBlue](#).

```
private void TextBoxGotKeyboardFocus(object sender, KeyboardFocusChangedEventArgs e)
{
    TextBox source = e.Source as TextBox;

    if (source != null)
    {
        // Change the TextBox color when it obtains focus.
        source.Background = Brushes.LightBlue;

        // Clear the TextBox.
        source.Clear();
    }
}
```

```
Private Sub TextBoxGotKeyboardFocus(ByVal sender As Object, ByVal e As KeyboardFocusChangedEventArgs)
    Dim source As TextBox = TryCast(e.Source, TextBox)

    If source IsNot Nothing Then
        ' Change the TextBox color when it obtains focus.
        source.Background = Brushes.LightBlue

        ' Clear the TextBox.
        source.Clear()
    End If
End Sub
```

Quando la [TextBox](#) perde lo stato attivo della tastiera, il [Background](#) proprietà del [TextBox](#) viene nuovamente impostato su bianco.

```
private void TextBoxLostKeyboardFocus(object sender, KeyboardFocusChangedEventArgs e)
{
    TextBox source = e.Source as TextBox;

    if (source != null)
    {
        // Change the TextBox color when it loses focus.
        source.Background = Brushes.White;

        // Set the hit counter back to zero and updates the display.
        this.ResetCounter();
    }
}
```

```
Private Sub TextBoxLostKeyboardFocus(ByVal sender As Object, ByVal e As KeyboardFocusChangedEventArgs)
    Dim source As TextBox = TryCast(e.Source, TextBox)

    If source IsNot Nothing Then
        ' Change the TextBox color when it loses focus.
        source.Background = Brushes.White

        ' Set the hit counter back to zero and updates the display.
        Me.ResetCounter()
    End If
End Sub
```

Gli eventi correlati allo stato attivo logico sono [GotFocus](#) e [LostFocus](#). Questi eventi sono definiti nel [FocusManager](#) come eventi associati, ma il [FocusManager](#) non espone wrapper di eventi CLR. [UIElement](#) e [ContentElement](#) espongono questi eventi in modo più appropriato.

Vedere anche

- [FocusManager](#)
- [UIElement](#)
- [ContentElement](#)
- [Cenni preliminari sull'input](#)
- [Cenni preliminari sugli elementi di base](#)

Applicazione di stili per lo stato attivo nei controlli e FocusVisualStyle

04/11/2019 • 11 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) offre due meccanismi paralleli per modificare l'aspetto visivo di un controllo che riceve lo stato attivo della tastiera. Il primo meccanismo consiste nell'usare i setter di proprietà per proprietà quali `IsKeyboardFocused` all'interno dello stile o del modello applicato al controllo. Il secondo meccanismo consiste nel fornire uno stile separato come valore della proprietà `FocusVisualStyle`; lo stile di visualizzazione dello stato attivo crea una struttura ad albero visuale separata per uno strumento decorativo che disegna sulla parte superiore del controllo, anziché modificare la struttura ad albero visuale del controllo o di un altro elemento dell'interfaccia utente sostituendolo. Questo argomento descrive gli scenari appropriati a ogni meccanismo.

Scopo dello stile di visualizzazione dello stato attivo

La funzionalità dello stile di visualizzazione dello stato attivo offre un "modello a oggetti" comune per introdurre un feedback utente visivo, basato sul passaggio tramite tastiera a un qualsiasi elemento dell'interfaccia utente. Questo è possibile senza bisogno di applicare un nuovo modello al controllo, né di conoscere la composizione specifica del modello.

Dal momento però che la funzionalità dello stile di visualizzazione dello stato attivo funziona a prescindere dai modelli di controllo, il feedback visivo che è possibile visualizzare per un controllo usando uno stile di visualizzazione dello stato attivo è necessariamente limitato. La funzionalità sovrappone una struttura ad albero visuale diversa (uno strumento decorativo visuale) sulla struttura ad albero visuale creata dal rendering di un controllo tramite il relativo modello. Si definisce questa struttura ad albero visuale separata usando uno stile che riempie la proprietà `FocusVisualStyle`.

Comportamento predefinito dello stile di visualizzazione dello stato attivo

Gli stili di visualizzazione dello stato attivo funzionano solo quando l'azione di impostazione dello stato attivo è stata iniziata dalla tastiera. Qualsiasi azione del mouse o modifica dello stato attivo a livello di codice disabilita la modalità degli stili di visualizzazione dello stato attivo. Per altre informazioni sulle differenze tra le modalità dello stato attivo, vedere [Cenni preliminari sullo stato attivo](#).

I temi dei controlli includono un comportamento predefinito dello stile di visualizzazione dello stato attivo che diventa lo stile di visualizzazione dello stato attivo per tutti i controlli del tema. Questo stile del tema è identificato dal valore della chiave statica `FocusVisualStyleKey`. Quando si dichiara uno stile di visualizzazione dello stato attivo personalizzato a livello di applicazione, questo sostituisce il comportamento dello stile predefinito dei temi. In alternativa, se si definisce l'intero tema, è necessario usare questa stessa chiave per definire lo stile del comportamento predefinito per tutto il tema.

Nei temi, lo stile di visualizzazione dello stato attivo predefinito in genere è molto semplice. Di seguito è riportata un'approssimazione generica:

```
<Style x:Key="{x:Static SystemParameters.FocusVisualStyleKey}">
  <Setter Property="Control.Template">
    <Setter.Value>
      <ControlTemplate>
        <Rectangle StrokeThickness="1"
          Stroke="Black"
          StrokeDashArray="1 2"
          SnapsToDevicePixels="true"/>
      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>
```

Quando usare gli stili di visualizzazione dello stato attivo

Dal punto di vista concettuale, l'aspetto degli stili di visualizzazione dello stato attivo applicati ai controlli deve essere coerente da un controllo all'altro. Un modo per garantire la coerenza consiste nel modificare lo stile di visualizzazione dello stato attivo solo se si sta componendo un tema completo, in cui ogni controllo definito nel tema ottiene lo stesso stile di visualizzazione dello stato attivo o una variazione di uno stile visivamente coerente tra i vari controlli. In alternativa, si può usare lo stesso stile (o stili simili) per assegnare uno stile a ogni elemento della pagina o di un'interfaccia utente in grado di ricevere lo stato attivo tramite tastiera.

L'impostazione di [FocusVisualStyle](#) per i singoli stili di controllo che non fanno parte di un tema non è l'utilizzo previsto degli stili di visualizzazione dello stato attivo. Un comportamento visivo incoerente tra controlli può infatti creare confusione nell'esperienza utente relativamente allo stato attivo. Se si intendono comportamenti specifici del controllo per lo stato attivo della tastiera che sono deliberatamente non coerenti in un tema, un approccio molto migliore consiste nell'usare i trigger negli stili per le singole proprietà dello stato di input, ad esempio [IsFocused](#) o [IsKeyboardFocused](#).

Gli stili di visualizzazione dello stato attivo funzionano esclusivamente per lo stato attivo da tastiera. Come tali, costituiscono un tipo di funzionalità di accessibilità. Se si vogliono apportare modifiche all'interfaccia utente per qualsiasi tipo di stato attivo, tramite mouse, tastiera o a livello di codice, è preferibile usare non gli stili di visualizzazione dello stato attivo, bensì setter e trigger all'interno degli stili oppure modelli che funzionino in base al valore delle proprietà generali dello stato attivo, ad esempio [IsFocused](#) o [IsKeyboardFocusWithin](#).

Come creare uno stile di visualizzazione dello stato attivo

Lo stile creato per uno stile di visualizzazione dello stato attivo deve avere sempre la [TargetType](#) di [Control](#). Lo stile deve essere costituito principalmente da un [ControlTemplate](#). Il tipo di destinazione non viene specificato come tipo in cui lo stile di visualizzazione dello stato attivo viene assegnato al [FocusVisualStyle](#).

Poiché il tipo di destinazione è sempre [Control](#), è necessario applicare uno stile usando proprietà comuni a tutti i controlli (usando le proprietà della classe [Control](#) e le relative classi di base). È necessario creare un modello che funzioni correttamente come sovrapposizione a un elemento dell'interfaccia utente e che non nasconde le aree funzionali del controllo. In generale, questo vuol dire che il feedback visivo deve essere visualizzato all'esterno dei margini del controllo o sotto forma di effetti temporanei o non intrusivi, che non blocchino l'hit testing sul controllo a cui viene applicato lo stile di visualizzazione dello stato attivo. Le proprietà che è possibile usare nell'associazione di modelli che sono utili per determinare il dimensionamento e il posizionamento del modello di sovrapposizione includono [ActualHeight](#), [ActualWidth](#), [Margin](#) e [Padding](#).

Alternative all'uso di uno stile di visualizzazione dello stato attivo

Per le situazioni in cui non è appropriato usare uno stile di visualizzazione dello stato attivo, perché si sta assegnando uno stile solo a singoli controlli o perché si vuole un controllo maggiore sul modello del controllo, sono disponibili molte altre proprietà e tecniche accessibili che creano un comportamento visivo in risposta alle

modifiche apportate allo stato attivo.

Trigger, setter e setter di eventi sono illustrati in dettaglio in [Applicazione di stili e modelli](#). La gestione degli eventi indirizzati è illustrata in [Cenni preliminari sugli eventi indirizzati](#).

IsKeyboardFocused

Se si è interessati specificamente allo stato attivo della tastiera, è possibile usare la proprietà di dipendenza [IsKeyboardFocused](#) per una proprietà [Trigger](#). Un trigger di proprietà in uno stile o in un modello rappresenta una tecnica più appropriata per la definizione di un comportamento dello stato attivo della tastiera specifico per un singolo controllo, che potrebbe non corrispondere visivamente al comportamento dello stato attivo della tastiera di altri controlli.

Un'altra proprietà di dipendenza simile è [IsKeyboardFocusWithin](#), che può essere utile per richiamare visivamente lo stato attivo della tastiera in un punto qualsiasi all'interno della composizione o all'interno dell'area funzionale del controllo. Ad esempio, è possibile inserire un trigger [IsKeyboardFocusWithin](#) in modo che un pannello che raggruppa diversi controlli venga visualizzato in modo diverso, anche se lo stato attivo della tastiera potrebbe essere più preciso su un singolo elemento all'interno di tale pannello.

È anche possibile usare gli eventi [GotKeyboardFocus](#) e [LostKeyboardFocus](#) (nonché gli equivalenti di anteprima). È possibile utilizzare questi eventi come base per un [EventSetter](#) oppure è possibile scrivere gestori per gli eventi nel code-behind.

Altre proprietà dello stato attivo

Se si desidera che tutte le possibili cause di modifica dello stato attivo per produrre un comportamento visivo, è necessario basare un setter o un trigger sulla proprietà di dipendenza [IsFocused](#) o in alternativa sul [GotFocus](#) o [LostFocus](#) eventi utilizzati per un [EventSetter](#).

Vedere anche

- [FocusVisualStyle](#)
- [Applicazione di stili e modelli](#)
- [Panoramica sullo stato attivo](#)
- [Cenni preliminari sull'input](#)

Procedura dettagliata: Creazione della prima applicazione a tocco

23/10/2019 • 8 minutes to read • [Edit Online](#)

WPF consente alle applicazioni rispondere al tocco. Ad esempio, è possibile interagire con un'applicazione utilizzando uno o più dita su un dispositivo sensibile al tocco, ad esempio un touchscreen che questa procedura dettagliata viene creata un'applicazione che consente all'utente di spostare, ridimensionare o ruotare un oggetto tramite tocco.

Prerequisiti

Per completare la procedura dettagliata, è necessario disporre dei componenti seguenti:

- Visual Studio.
- Un dispositivo che accetta un input tocco, ad esempio un touchscreen, che supporta Windows Touch.

Inoltre, si deve avere una conoscenza di base di come creare un'applicazione in WPF, in particolare la modalità sottoscrivere e gestire un evento. Per altre informazioni, vedere [Procedura dettagliata: Prima applicazione desktop WPF](#).

Creare l'applicazione

Per creare l'applicazione

1. Creare un nuovo progetto di applicazione WPF in Visual Basic o Visual C# denominato `BasicManipulation`.

Per altre informazioni, vedere [Procedura dettagliata: Prima applicazione desktop WPF](#).

2. Sostituire il contenuto del file `MainWindow.xaml` con il XAML seguente.

Questo codice crea un'applicazione semplice che contiene una linea rossa `Rectangle` su un `Canvas`. Il `IsManipulationEnabled` proprietà del `Rectangle` è impostato su `true` in modo che possa ricevere gli eventi di manipolazione. L'applicazione sottoscrive il `ManipulationStarting`, `ManipulationDelta`, e `ManipulationInertiaStarting` eventi. Questi eventi contengono la logica per spostare il `Rectangle` quando l'utente lo modifica.

```

<Window x:Class="BasicManipulation.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Move, Size, and Rotate the Square"
    WindowState="Maximized"
    ManipulationStarting="Window_ManipulationStarting"
    ManipulationDelta="Window_ManipulationDelta"
    ManipulationInertiaStarting="Window_InertiaStarting">
<Window.Resources>

    <!--The movement, rotation, and size of the Rectangle is
        specified by its RenderTransform.-->
    <MatrixTransform x:Key="InitialMatrixTransform">
        <MatrixTransform.Matrix>
            <Matrix OffsetX="200" OffsetY="200"/>
        </MatrixTransform.Matrix>
    </MatrixTransform>

</Window.Resources>

<Canvas>
    <Rectangle Fill="Red" Name="manRect"
        Width="200" Height="200"
        RenderTransform="{StaticResource InitialMatrixTransform}"
        IsManipulationEnabled="true" />
</Canvas>
</Window>

```

3. Se si usa Visual Basic, nella prima riga del file MainWindow.xaml, sostituire

`x:Class="BasicManipulation.MainWindow"` CON `x:Class="MainWindow"`.

4. Nel `MainWindow` classe, aggiungere il codice seguente `ManipulationStarting` gestore dell'evento.

Il `ManipulationStarting` evento si verifica quando WPF rileva un tocco input inizia a modificare un oggetto. Il codice specifica che la posizione della manipolazione deve essere relativa ai `Window` impostando la `ManipulationContainer` proprietà.

```

void Window_ManipulationStarting(object sender, ManipulationStartingEventArgs e)
{
    e.ManipulationContainer = this;
    e.Handled = true;
}

```

```

Private Sub Window_ManipulationStarting(ByVal sender As Object, ByVal e As
ManipulationStartingEventArgs)
    e.ManipulationContainer = Me
    e.Handled = True
End Sub

```

5. Nel `MainWindow` classe, aggiungere il codice seguente `ManipulationDelta` gestore dell'evento.

Il `ManipulationDelta` evento si verifica quando il tocco modifica la posizione di input e può essere presente più volte durante una manipolazione. L'evento può verificarsi anche dopo la generazione di un dito. Ad esempio, se l'utente sposta un dito su una schermata, il `ManipulationDelta` evento si verifica più volte durante lo spostamento del dito. Quando l'utente solleva un dito dallo schermo, la `ManipulationDelta` eventi continua a verificarsi per simulare l'inerzia.

Il codice si applica il `DeltaManipulation` per il `RenderTransform` del `Rectangle` per spostarla quando l'utente sposta il tocco di input. Viene inoltre verificato se il `Rectangle` fuori dei limiti del `Window` quando l'evento si

verifica durante l'inerzia. Pertanto, l'applicazione chiama il [ManipulationDeltaEventArgs.Complete](#) metodo per terminare la modifica.

```
void Window_ManipulationDelta(object sender, ManipulationDeltaEventArgs e)
{
    // Get the Rectangle and its RenderTransform matrix.
    Rectangle rectToMove = e.OriginalSource as Rectangle;
    Matrix rectMatrix = ((MatrixTransform)rectToMove.RenderTransform).Matrix;

    // Rotate the Rectangle.
    rectMatrix.RotateAt(e.DeltaManipulation.Rotation,
        e.ManipulationOrigin.X,
        e.ManipulationOrigin.Y);

    // Resize the Rectangle. Keep it square
    // so use only the X value of Scale.
    rectMatrix.ScaleAt(e.DeltaManipulation.Scale.X,
        e.DeltaManipulation.Scale.X,
        e.ManipulationOrigin.X,
        e.ManipulationOrigin.Y);

    // Move the Rectangle.
    rectMatrix.Translate(e.DeltaManipulation.Translation.X,
        e.DeltaManipulation.Translation.Y);

    // Apply the changes to the Rectangle.
    rectToMove.RenderTransform = new MatrixTransform(rectMatrix);

    Rect containingRect =
        new Rect(((FrameworkElement)e.ManipulationContainer).RenderSize);

    Rect shapeBounds =
        rectToMove.RenderTransform.TransformBounds(
            new Rect(rectToMove.RenderSize));

    // Check if the rectangle is completely in the window.
    // If it is not and inertia is occurring, stop the manipulation.
    if (e.IsInertial && !containingRect.Contains(shapeBounds))
    {
        e.Complete();
    }

    e.Handled = true;
}
```

```

Private Sub Window_ManipulationDelta(ByVal sender As Object, ByVal e As ManipulationDeltaEventArgs)

    ' Get the Rectangle and its RenderTransform matrix.
    Dim rectToMove As Rectangle = e.OriginalSource
    Dim rectTransform As MatrixTransform = rectToMove.RenderTransform
    Dim rectsMatrix As Matrix = rectTransform.Matrix

    ' Rotate the shape
    rectsMatrix.RotateAt(e.DeltaManipulation.Rotation,
        e.ManipulationOrigin.X,
        e.ManipulationOrigin.Y)

    ' Resize the Rectangle. Keep it square
    ' so use only the X value of Scale.
    rectsMatrix.ScaleAt(e.DeltaManipulation.Scale.X,
        e.DeltaManipulation.Scale.X,
        e.ManipulationOrigin.X,
        e.ManipulationOrigin.Y)

    'move the center
    rectsMatrix.Translate(e.DeltaManipulation.Translation.X,
        e.DeltaManipulation.Translation.Y)

    ' Apply the changes to the Rectangle.
    rectTransform = New MatrixTransform(rectsMatrix)
    rectToMove.RenderTransform = rectTransform

    Dim container As FrameworkElement = e.ManipulationContainer
    Dim containingRect As New Rect(container.RenderSize)

    Dim shapeBounds As Rect = rectTransform.TransformBounds(
        New Rect(rectToMove.RenderSize))

    ' Check if the rectangle is completely in the window.
    ' If it is not and inertia is occurring, stop the manipulation.
    If e.IsInertial AndAlso Not containingRect.Contains(shapeBounds) Then
        e.Complete()
    End If

    e.Handled = True
End Sub

```

6. Nel `MainWindow` classe, aggiungere il codice seguente `ManipulationInertiaStarting` gestore dell'evento.

Il `ManipulationInertiaStarting` evento si verifica quando l'utente genera tutte le dita dallo schermo. Il codice imposta la velocità iniziale e la decelerazione per il movimento, espansione e rotazione del rettangolo.

```

void Window_InertiaStarting(object sender, ManipulationInertiaStartingEventArgs e)
{
    // Decrease the velocity of the Rectangle's movement by
    // 10 inches per second every second.
    // (10 inches * 96 pixels per inch / 1000ms^2)
    e.TranslationBehavior.DesiredDeceleration = 10.0 * 96.0 / (1000.0 * 1000.0);

    // Decrease the velocity of the Rectangle's resizing by
    // 0.1 inches per second every second.
    // (0.1 inches * 96 pixels per inch / (1000ms^2))
    e.ExpansionBehavior.DesiredDeceleration = 0.1 * 96 / (1000.0 * 1000.0);

    // Decrease the velocity of the Rectangle's rotation rate by
    // 2 rotations per second every second.
    // (2 * 360 degrees / (1000ms^2))
    e.RotationBehavior.DesiredDeceleration = 720 / (1000.0 * 1000.0);

    e.Handled = true;
}

```

```

Private Sub Window_InertiaStarting(ByVal sender As Object,
                                   ByVal e As ManipulationInertiaStartingEventArgs)

    ' Decrease the velocity of the Rectangle's movement by
    ' 10 inches per second every second.
    ' (10 inches * 96 pixels per inch / 1000ms^2)
    e.TranslationBehavior.DesiredDeceleration = 10.0 * 96.0 / (1000.0 * 1000.0)

    ' Decrease the velocity of the Rectangle's resizing by
    ' 0.1 inches per second every second.
    ' (0.1 inches * 96 pixels per inch / (1000ms^2))
    e.ExpansionBehavior.DesiredDeceleration = 0.1 * 96 / (1000.0 * 1000.0)

    ' Decrease the velocity of the Rectangle's rotation rate by
    ' 2 rotations per second every second.
    ' (2 * 360 degrees / (1000ms^2))
    e.RotationBehavior.DesiredDeceleration = 720 / (1000.0 * 1000.0)

    e.Handled = True
End Sub

```

7. Compilare ed eseguire il progetto.

Nella finestra verrà visualizzato un quadrato rosso.

Verifica dell'applicazione

Per testare l'applicazione, provare le modifiche che seguono. Si noti che è possibile eseguire più di una delle seguenti operazioni nello stesso momento.

- Per spostare il **Rectangle**, posizionare un dito sul **Rectangle** e muovere il dito sullo schermo.
- Per ridimensionare il **Rectangle**, posizionare due dita sul **Rectangle** e sposta le dita o chiudere ogni.
- Per ruotare il **Rectangle**, posizionare due dita sul **Rectangle** e ruotare le dita una attorno a altra.

Affinché l'inerzia, generare rapidamente le dita dalla schermata quando si eseguono modifiche precedenti. Il **Rectangle** continueranno a spostare, ridimensionare o ruotare per alcuni secondi prima che venga interrotta.

Vedere anche

- [UIElement.ManipulationStarting](#)
- [UIElement.ManipulationDelta](#)
- [UIElement.ManipulationInertiaStarting](#)

Procedure relative all'input e ai comandi

23/10/2019 • 2 minutes to read • [Edit Online](#)

Gli argomenti in questa sezione descrivono come usare l'infrastruttura di input e di esecuzione dei comandi in Windows Presentation Foundation (WPF).

In questa sezione

- [Attivare un comando](#)
- [Modificare il tipo di cursore](#)
- [Modificare il colore di un elemento usando eventi di stato attivo](#)
- [Applicare un oggetto FocusVisualStyle a un controllo](#)
- [Rilevare il momento in cui è stato premuto il tasto INVIO](#)
- [Creare un effetto di attivazione usando gli eventi](#)
- [Fare in modo che un oggetto segua il puntatore del mouse](#)
- [Creare un oggetto RoutedCommand](#)
- [Implementare ICommandSource](#)
- [Associare un comando a un controllo senza supporto del comando](#)
- [Associare un comando a un controllo con supporto dei comandi](#)

Riferimenti

- [UIElement](#)
- [FrameworkElement](#)
- [ContentElement](#)
- [FrameworkContentElement](#)
- [Keyboard](#)
- [Mouse](#)
- [FocusManager](#)

Sezioni correlate

Procedura: Attivare un comando

23/10/2019 • 3 minutes to read • [Edit Online](#)

Nell'esempio seguente illustra l'utilizzo dei comandi in Windows Presentation Foundation (WPF). Nell'esempio viene illustrato come associare un [RoutedCommand](#) a un [Button](#), creare un [CommandBinding](#) e creare i gestori di eventi che implementano il [RoutedCommand](#). Per altre informazioni sull'esecuzione di comandi, vedere la [Cenni preliminari](#).

Esempio

La prima sezione di codice crea il interfaccia utente, che è costituito un [Button](#) e una [StackPanel](#) e crea un [CommandBinding](#) che associa i gestori di comando con il [RoutedCommand](#).

Il [Command](#) proprietà del [Button](#) è associato il [Close](#) comando.

Il [CommandBinding](#) viene aggiunto per il [CommandBindingCollection](#) della radice [Window](#). Il [Executed](#) e [CanExecute](#) gestori eventi sono associati a questa associazione e associati i [Close](#) comando.

Senza il [CommandBinding](#) non vi è alcuna logica di comando, ma solo un meccanismo per richiamare il comando. Quando il [Button](#) viene selezionata, il [PreviewExecuted RoutedEvent](#) viene generato il comando sulla destinazione seguita dal [Executed RoutedEvent](#). Questi eventi attraversano l'albero degli elementi cercando un [CommandBinding](#) per quel comando specifico. Vale la pena notare che, poiché [RoutedEvent](#) tunneling e bubbling attraverso l'albero degli elementi, prestare attenzione nella posizione in cui il [CommandBinding](#) viene inserito. Se il [CommandBinding](#) si trova in un elemento di pari livello di destinazione del comando o un altro nodo che non è presente nella route del [RoutedEvent](#), il [CommandBinding](#) non sarà accessibile.

```
<Window x:Class="WCSamples.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="CloseCommand"
    Name="RootWindow"
    >
<Window.CommandBindings>
    <CommandBinding Command="ApplicationCommands.Close"
        Executed="CloseCommandHandler"
        CanExecute="CanExecuteHandler"
        />
</Window.CommandBindings>
<StackPanel Name="MainStackPanel">
    <Button Command="ApplicationCommands.Close"
        Content="Close File" />
</StackPanel>
</Window>
```

```

// Create ui elements.
StackPanel CloseCmdStackPanel = new StackPanel();
Button CloseCmdButton = new Button();
CloseCmdStackPanel.Children.Add(CloseCmdButton);

// Set Button's properties.
CloseCmdButton.Content = "Close File";
CloseCmdButton.Command = ApplicationCommands.Close;

// Create the CommandBinding.
CommandBinding CloseCommandBinding = new CommandBinding(
    ApplicationCommands.Close, CloseCommandHandler, CanExecuteHandler);

// Add the CommandBinding to the root Window.
RootWindow.CommandBindings.Add(CloseCommandBinding);

```

```

' Create ui elements.
Dim CloseCmdStackPanel As New StackPanel()
Dim CloseCmdButton As New Button()
CloseCmdStackPanel.Children.Add(CloseCmdButton)

' Set Button's properties.
CloseCmdButton.Content = "Close File"
CloseCmdButton.Command = ApplicationCommands.Close

' Create the CommandBinding.
Dim CloseCommandBinding As New CommandBinding(ApplicationCommands.Close, AddressOf CloseCommandHandler,
AddressOf CanExecuteHandler)

' Add the CommandBinding to the root Window.
RootWindow.CommandBindings.Add(CloseCommandBinding)

```

La sezione successiva del codice implementa il [Executed](#) e [CanExecute](#) gestori eventi.

Il [Executed](#) gestore chiama un metodo per chiudere il file aperto. Il [CanExecute](#) gestore chiama un metodo per determinare se un file è aperto. Se un file è aperto [CanExecute](#) è impostata su `true`; in caso contrario, impostarlo su `false`.

```

// Executed event handler.
private void CloseCommandHandler(object sender, ExecutedRoutedEventArgs e)
{
    // Calls a method to close the file and release resources.
    CloseFile();
}

// CanExecute event handler.
private void CanExecuteHandler(object sender, CanExecuteRoutedEventArgs e)
{
    // Call a method to determine if there is a file open.
    // If there is a file open, then set CanExecute to true.
    if (IsFileOpened())
    {
        e.CanExecute = true;
    }
    // if there is not a file open, then set CanExecute to false.
    else
    {
        e.CanExecute = false;
    }
}

```

```
' Executed event handler.  
Private Sub CloseCommandHandler(ByVal sender As Object, ByVal e As ExecutedRoutedEventArgs)  
    ' Calls a method to close the file and release resources.  
    CloseFile()  
End Sub  
  
' CanExecute event handler.  
Private Sub CanExecuteHandler(ByVal sender As Object, ByVal e As CanExecuteRoutedEventArgs)  
    ' Call a method to determine if there is a file open.  
    ' If there is a file open, then set CanExecute to true.  
    If IsFileOpened() Then  
        e.CanExecute = True  
    ' if there is not a file open, then set CanExecute to false.  
    Else  
        e.CanExecute = False  
    End If  
End Sub
```

Vedere anche

- [Panoramica sull'esecuzione di comandi](#)

Procedura: Modificare il tipo di cursore

23/10/2019 • 4 minutes to read • [Edit Online](#)

In questo esempio viene illustrato come modificare il [Cursor](#) del puntatore del mouse per un elemento specifico e per l'applicazione.

In questo esempio è costituito un Extensible Application Markup Language (XAML) file e un file code-behind.

Esempio

Viene creato l'interfaccia utente, che è costituito un [ComboBox](#) per selezionare il valore desiderato [Cursor](#), una coppia di [RadioButton](#) oggetti per determinare se la modifica del cursore si applica a un solo elemento o si applica all'intera applicazione e un [Border](#) ovvero l'elemento che viene applicato il nuovo cursore.

```

<StackPanel>
    <Border Width="300">
        <StackPanel Orientation="Horizontal"
            HorizontalAlignment="Center">
            <StackPanel Margin="10">
                <Label HorizontalAlignment="Left">Cursor Type</Label>
                <ComboBox Width="100"
                    SelectionChanged="CursorTypeChanged"
                    HorizontalAlignment="Left"
                    Name="CursorSelector">
                    <ComboBoxItem Content="AppStarting" />
                    <ComboBoxItem Content="ArrowCD" />
                    <ComboBoxItem Content="Arrow" />
                    <ComboBoxItem Content="Cross" />
                    <ComboBoxItem Content="HandCursor" />
                    <ComboBoxItem Content="Help" />
                    <ComboBoxItem Content="IBeam" />
                    <ComboBoxItem Content="No" />
                    <ComboBoxItem Content="None" />
                    <ComboBoxItem Content="Pen" />
                    <ComboBoxItem Content="ScrollSE" />
                    <ComboBoxItem Content="ScrollWE" />
                    <ComboBoxItem Content="SizeAll" />
                    <ComboBoxItem Content="SizeNESW" />
                    <ComboBoxItem Content="SizeNS" />
                    <ComboBoxItem Content="SizeNWSE" />
                    <ComboBoxItem Content="SizeWE" />
                    <ComboBoxItem Content="UpArrow" />
                    <ComboBoxItem Content="WaitCursor" />
                    <ComboBoxItem Content="Custom" />
                </ComboBox>
            </StackPanel>
        <!-- The user can select different cursor types using this ComboBox -->
        <StackPanel Margin="10">
            <Label HorizontalAlignment="Left">Scope of Cursor</Label>
            <StackPanel>
                <RadioButton Name="rbScopeElement" IsChecked="True"
                    Checked="CursorScopeSelected">Display Area Only</RadioButton>
                <RadioButton Name="rbScopeApplication"
                    Checked="CursorScopeSelected">Entire Application</RadioButton>
            </StackPanel>
        </StackPanel>
    </StackPanel>
</Border>
<!-- When the mouse pointer is over this Border -->
<!-- the selected cursor type is shown -->
<Border Name="DisplayArea" Height="250" Width="400"
    Margin="20" Background="AliceBlue">
    <Label HorizontalAlignment="Center">
        Move Mouse Pointer Over This Area
    </Label>
</Border>
</StackPanel>

```

Il codice sottostante crea un [SelectionChanged](#) gestore eventi chiamato quando il tipo di cursore è stato modificato nel [ComboBox](#). Un'istruzione switch vengono applicati filtri al nome del cursore e imposta il [Cursor](#) proprietà di [Border](#) denominato *DisplayArea*.

Se la modifica del cursore è impostata su "Intera applicazione", il [OverrideCursor](#) è impostata sul [Cursor](#) proprietà del [Border](#) controllo. In tal modo il cursore da modificare per l'intera applicazione.

```

private void CursorTypeChanged(object sender, SelectionChangedEventArgs e)
{
    ComboBox source = e.Source as ComboBox;

```

```
if (source != null)
{
    ComboBoxItem selectedCursor = source.SelectedItem as ComboBoxItem;

    // Changing the cursor of the Border control
    // by setting the Cursor property
    switch (selectedCursor.Content.ToString())
    {
        case "AppStarting":
            DisplayArea.Cursor = Cursors.AppStarting;
            break;
        case "ArrowCD":
            DisplayArea.Cursor = Cursors.ArrowCD;
            break;
        case "Arrow":
            DisplayArea.Cursor = Cursors.Arrow;
            break;
        case "Cross":
            DisplayArea.Cursor = Cursors.Cross;
            break;
        case "HandCursor":
            DisplayArea.Cursor = Cursors.Hand;
            break;
        case "Help":
            DisplayArea.Cursor = Cursors.Help;
            break;
        case "IBeam":
            DisplayArea.Cursor = Cursors.IBeam;
            break;
        case "No":
            DisplayArea.Cursor = Cursors.No;
            break;
        case "None":
            DisplayArea.Cursor = Cursors.None;
            break;
        case "Pen":
            DisplayArea.Cursor = Cursors.Pen;
            break;
        case "ScrollSE":
            DisplayArea.Cursor = Cursors.ScrollSE;
            break;
        case "ScrollWE":
            DisplayArea.Cursor = Cursors.ScrollWE;
            break;
        case "SizeAll":
            DisplayArea.Cursor = Cursors.SizeAll;
            break;
        case "SizeNESW":
            DisplayArea.Cursor = Cursors.SizeNESW;
            break;
        case "SizeNS":
            DisplayArea.Cursor = Cursors.SizeNS;
            break;
        case "SizeNWSE":
            DisplayArea.Cursor = Cursors.SizeNWSE;
            break;
        case "SizeWE":
            DisplayArea.Cursor = Cursors.SizeWE;
            break;
        case "UpArrow":
            DisplayArea.Cursor = Cursors.UpArrow;
            break;
        case "WaitCursor":
            DisplayArea.Cursor = Cursors.Wait;
            break;
        case "Custom":
            DisplayArea.Cursor = CustomCursor;
            break;
        default:
```

```
        break;  
    }  
  
    // If the cursor scope is set to the entire application  
    // Use OverrideCursor to force the cursor for all elements  
    if (cursorScopeElementOnly == false)  
    {  
        Mouse.OverrideCursor = DisplayArea.Cursor;  
    }  
}
```

```

' When the Radiobox changes, a new cursor type is set
Private Sub CursorTypeChanged(ByVal sender As Object, ByVal e As SelectionChangedEventArgs)

    Dim item As String = CType(e.Source, ComboBox).SelectedItem.Content.ToString()

    Select Case item
        Case "AppStarting"
            DisplayArea.Cursor = Cursors.AppStarting
        Case "ArrowCD"
            DisplayArea.Cursor = Cursors.ArrowCD
        Case "Arrow"
            DisplayArea.Cursor = Cursors.Arrow
        Case "Cross"
            DisplayArea.Cursor = Cursors.Cross
        Case "HandCursor"
            DisplayArea.Cursor = Cursors.Hand
        Case "Help"
            DisplayArea.Cursor = Cursors.Help
        Case "IBeam"
            DisplayArea.Cursor = Cursors.IBeam
        Case "No"
            DisplayArea.Cursor = Cursors.No
        Case "None"
            DisplayArea.Cursor = Cursors.None
        Case "Pen"
            DisplayArea.Cursor = Cursors.Pen
        Case "ScrollSE"
            DisplayArea.Cursor = Cursors.ScrollSE
        Case "ScrollWE"
            DisplayArea.Cursor = Cursors.ScrollWE
        Case "SizeAll"
            DisplayArea.Cursor = Cursors.SizeAll
        Case "SizeNESW"
            DisplayArea.Cursor = Cursors.SizeNESW
        Case "SizeNS"
            DisplayArea.Cursor = Cursors.SizeNS
        Case "SizeNWSE"
            DisplayArea.Cursor = Cursors.SizeNWSE
        Case "SizeWE"
            DisplayArea.Cursor = Cursors.SizeWE
        Case "UpArrow"
            DisplayArea.Cursor = Cursors.UpArrow
        Case "WaitCursor"
            DisplayArea.Cursor = Cursors.Wait
        Case "Custom"
            DisplayArea.Cursor = CustomCursor
    End Select

    ' if the cursor scope is set to the entire application
    ' use OverrideCursor to force the cursor for all elements
    If (cursorScopeElementOnly = False) Then
        Mouse.OverrideCursor = DisplayArea.Cursor
    End If

End Sub

```

Vedere anche

- [Cenni preliminari sull'input](#)

Procedura: Modificare il colore di un elemento utilizzando eventi di stato attivo

23/10/2019 • 2 minutes to read • [Edit Online](#)

In questo esempio viene illustrato come modificare il colore di un elemento quando si ottiene e perde lo stato attivo usando gli eventi [GotFocus](#) e [LostFocus](#).

Questo esempio è costituito da un file Extensible Application Markup Language (XAML) e da un file code-behind.

Esempio

Il codice XAML seguente crea l'interfaccia utente, che è costituita da due oggetti [Button](#) e connette i gestori eventi per gli eventi [GotFocus](#) e [LostFocus](#) agli oggetti [Button](#).

```
<StackPanel>
    <StackPanel.Resources>
        <Style TargetType="{x:Type Button}">
            <Setter Property="Height" Value="20"/>
            <Setter Property="Width" Value="250"/>
            <Setter Property="HorizontalAlignment" Value="Left"/>
        </Style>
    </StackPanel.Resources>
    <Button
        GotFocus="OnGotFocusHandler"
        LostFocus="OnLostFocusHandler">Click Or Tab To Give Keyboard Focus</Button>
    <Button
        GotFocus="OnGotFocusHandler"
        LostFocus="OnLostFocusHandler">Click Or Tab To Give Keyborad Focus</Button>
</StackPanel>
```

Il codice sottostante seguente consente di creare i gestori eventi [GotFocus](#) e [LostFocus](#). Quando il [Button](#) Ottiene lo stato attivo della tastiera, il [Background](#) del [Button](#) viene modificato in rosso. Quando il [Button](#) perde lo stato attivo della tastiera, il [Background](#) del [Button](#) viene modificato di nuovo in bianco.

```

public partial class Window1 : Window
{
    public Window1()
    {
        InitializeComponent();
    }

    // Raised when Button gains focus.
    // Changes the color of the Button to Red.
    private void OnGotFocusHandler(object sender, RoutedEventArgs e)
    {
        Button tb = e.Source as Button;
        tb.Background = Brushes.Red;
    }

    // Raised when Button losses focus.
    // Changes the color of the Button back to white.
    private void OnLostFocusHandler(object sender, RoutedEventArgs e)
    {
        Button tb = e.Source as Button;
        tb.Background = Brushes.White;
    }
}

```

```

Partial Public Class Window1
Inherits Window

Public Sub New()
    InitializeComponent()
End Sub

'raised when Button gains focus. Changes the color of the Button to red.
Private Sub OnGotFocusHandler(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Dim tb As Button = CType(e.Source, Button)
    tb.Background = Brushes.Red
End Sub

'raised when Button loses focus. Changes the color back to white.
Private Sub OnLostFocusHandler(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Dim tb As Button = CType(e.Source, Button)
    tb.Background = Brushes.White
End Sub
End Class

```

Vedere anche

- [Cenni preliminari sull'input](#)

Procedura: applicare un oggetto FocusVisualStyle a un controllo

04/11/2019 • 2 minutes to read • [Edit Online](#)

Questo esempio illustra come creare uno stile di visualizzazione dello stato attivo nelle risorse e applicare lo stile a un controllo usando la proprietà [FocusVisualStyle](#).

Esempio

Nell'esempio seguente viene definito uno stile che consente di creare la composizione di controlli aggiuntivi che si applica solo quando il controllo è focalizzato sulla tastiera nel interfaccia utente. Questa operazione viene eseguita definendo uno stile con una [ControlTemplate](#), quindi facendo riferimento a tale stile come risorsa quando si imposta la proprietà [FocusVisualStyle](#).

Un rettangolo esterno simile a un bordo viene inserito all'esterno dell'area rettangolare. Se non diversamente modificato, il dimensionamento dello stile usa il [ActualHeight](#) e [ActualWidth](#) del controllo rettangolare in cui viene applicato lo stile di visualizzazione dello stato attivo. In questo esempio vengono impostati i valori negativi per il [Margin](#) per far apparire il bordo leggermente al di fuori del controllo attivo.

```
<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    >
    <Page.Resources>
        <Style x:Key="MyFocusVisual">
            <Setter Property="Control.Template">
                <Setter.Value>
                    <ControlTemplate>
                        <Rectangle Margin="-2" StrokeThickness="1" Stroke="Red" StrokeDashArray="1 2"/>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>
        </Style>
    </Page.Resources>
    <StackPanel Background="Ivory" Orientation="Horizontal">
        <Canvas Width="10"/>
        <Button Width="100" Height="30" FocusVisualStyle="{DynamicResource MyFocusVisual}">
            Focus Here</Button>
        <Canvas Width="100"/>
        <Button Width="100" Height="30" FocusVisualStyle="{DynamicResource MyFocusVisual}">
            Focus Here</Button>
    </StackPanel>
</Page>
```

Un [FocusVisualStyle](#) è additivo per qualsiasi stile di modello di controllo che deriva da uno stile esplicito o da uno stile del tema; lo stile principale di un controllo può comunque essere creato utilizzando un [ControlTemplate](#) e impostando tale stile sulla proprietà [Style](#).

Gli stili di visualizzazione dello stato attivo devono essere usati in modo coerente in un tema o in un'interfaccia utente, anziché usare uno diverso per ogni elemento attivabile. Per informazioni dettagliate, vedere applicazione di [stili per lo stato attivo nei controlli e FocusVisualStyle](#).

Vedere anche

- [FocusVisualStyle](#)
- [Applicazione di stili e modelli](#)
- [Applicazione di stili per lo stato attivo nei controlli e FocusVisualStyle](#)

Procedura: Rileva quando viene premuto il tasto invio

23/10/2019 • 2 minutes to read • [Edit Online](#)

Questo esempio Mostra come rilevare quando il tasto **Enter** è premuto sulla tastiera.

Questo esempio è costituito da un file Extensible Application Markup Language (XAML) e da un file code-behind.

Esempio

Quando l'utente preme il tasto **Enter** nella **TextBox**, l'input nella casella di testo viene visualizzato in un'altra area del interfaccia utente.

Il codice XAML seguente crea l'interfaccia utente, che è costituita da un **StackPanel**, da un **TextBlock** e da un **TextBox**.

```
<StackPanel>
    <TextBlock Width="300" Height="20">
        Type some text into the TextBox and press the Enter key.
    </TextBlock>
    <TextBox Width="300" Height="30" Name="textBox1"
        KeyDown="OnKeyDownHandler"/>
    <TextBlock Width="300" Height="100" Name="textBlock1"/>
</StackPanel>
```

Il codice sottostante seguente crea il gestore dell'evento **KeyDown**. Se il tasto premuto è il tasto **Enter**, viene visualizzato un messaggio nel **TextBlock**.

```
private void OnKeyDownHandler(object sender, KeyEventArgs e)
{
    if (e.Key == Key.Return)
    {
        textBlock1.Text = "You Entered: " + textBox1.Text;
    }
}
```

```
Private Sub OnKeyDownHandler(ByVal sender As Object, ByVal e As KeyEventArgs)
    If (e.Key = Key.Return) Then
        textBlock1.Text = "You Entered: " + textBox1.Text
    End If
End Sub
```

Vedere anche

- [Cenni preliminari sull'input](#)
- [Cenni preliminari sugli eventi indirizzati](#)

Procedura: creare un effetto di attivazione utilizzando gli eventi

04/11/2019 • 2 minutes to read • [Edit Online](#)

Questo esempio Mostra come modificare il colore di un elemento quando il puntatore del mouse entra e lascia l'area occupata dall'elemento.

Questo esempio è costituito da un file di Extensible Application Markup Language (XAML) e da un file code-behind.

NOTE

In questo esempio viene illustrato come utilizzare gli eventi, ma il metodo consigliato per ottenere questo risultato consiste nell'utilizzare un [Trigger](#) in uno stile. Per altre informazioni, vedere [Applicazione di stili e modelli](#).

Esempio

Il codice XAML seguente crea l'interfaccia utente, che è costituita da [Border](#) intorno a una [TextBlock](#) connette i gestori eventi di [MouseEnter](#) e [MouseLeave](#) al [Border](#).

```
<StackPanel>
    <Border MouseEnter="OnMouseEnterHandler"
            MouseLeave="OnMouseLeaveHandler"
            Name="border1" Margin="10"
            BorderThickness="1"
            BorderBrush="Black"
            VerticalAlignment="Center"
            Width="300" Height="100">
        <Label Margin="10" FontSize="14"
              HorizontalAlignment="Center">Move Cursor Over Me</Label>
    </Border>
</StackPanel>
```

Il codice sottostante seguente crea i gestori eventi [MouseEnter](#) e [MouseLeave](#). Quando il puntatore del mouse entra nel [Border](#), lo sfondo del [Border](#) viene modificato in rosso. Quando il puntatore del mouse esce dalla [Border](#), lo sfondo del [Border](#) viene modificato di nuovo in bianco.

```
public partial class Window1 : Window
{
    public Window1()
    {
        InitializeComponent();
    }

    // raised when mouse cursor enters the area occupied by the element
    void OnMouseEnterHandler(object sender, MouseEventArgs e)
    {
        border1.Background = Brushes.Red;
    }

    // raised when mouse cursor leaves the area occupied by the element
    void OnMouseLeaveHandler(object sender, MouseEventArgs e)
    {
        border1.Background = Brushes.White;
    }
}
```

```
Partial Public Class Window1
    Inherits Window

    Public Sub New()
        InitializeComponent()
    End Sub
    ' raised when mouse cursor enters the are occupied by the element
    Private Sub OnMouseEnterHandler(ByVal sender As Object, ByVal e As MouseEventArgs)
        border1.Background = Brushes.Red
    End Sub
    ' raised when mouse cursor leaves the are occupied by the element
    Private Sub OnMouseLeaveHandler(ByVal sender As Object, ByVal e As MouseEventArgs)
        border1.Background = Brushes.White
    End Sub
End Class
```

Procedura: Fare in modo che un oggetto segua il puntatore del mouse

23/10/2019 • 2 minutes to read • [Edit Online](#)

Questo esempio Mostra come modificare le dimensioni di un oggetto quando il puntatore del mouse viene spostato sullo schermo.

Nell'esempio è incluso un file Extensible Application Markup Language (XAML) che crea il interfaccia utente e un file code-behind che crea il gestore eventi.

Esempio

Il codice XAML seguente crea il Interfaccia utente, che è costituito da un [Ellipse](#) all'interno di un [StackPanel](#) e connette il gestore eventi per l'evento [MouseMove](#).

```
<Window x:Class="WCSamples.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="mouseMoveWithPointer"
    Height="400"
    Width="500"
    >
<Canvas MouseMove="MouseMoveHandler"
    Background="LemonChiffon">
    <Ellipse Name="ellipse" Fill="LightBlue"
        Width="100" Height="100"/>
</Canvas>
</Window>
```

Il codice sottostante seguente crea il gestore dell'evento [MouseMove](#). Quando il puntatore del mouse viene spostato, l'altezza e la larghezza del [Ellipse](#) vengono aumentate e diminuite.

```
// raised when the mouse pointer moves.
// Expands the dimensions of an Ellipse when the mouse moves.
private void MouseMoveHandler(object sender, MouseEventArgs e)
{
    // Get the x and y coordinates of the mouse pointer.
    System.Windows.Point position = e.GetPosition(this);
    double pX = position.X;
    double pY = position.Y;

    // Sets the Height/Width of the circle to the mouse coordinates.
    ellipse.Width = pX;
    ellipse.Height = pY;
}
```

```
' raised when the mouse pointer moves.  
' Expands the dimensions of an Ellipse when the mouse moves.  
Private Sub OnMouseMoveHandler(ByVal sender As Object, ByVal e As MouseEventArgs)  
  
    'Get the x and y coordinates of the mouse pointer.  
    Dim position As System.Windows.Point  
    position = e.GetPosition(Me)  
    Dim pX As Double  
    pX = position.X  
    Dim pY As Double  
    pY = position.Y  
  
    'Set the Height and Width of the Ellipse to the mouse coordinates.  
    ellipse1.Height = pY  
    ellipse1.Width = pX  
End Sub
```

Vedere anche

- [Cenni preliminari sull'input](#)

Procedura: Creare un oggetto RoutedCommand

23/10/2019 • 2 minutes to read • [Edit Online](#)

Questo esempio viene illustrato come creare una classe personalizzata [RoutedCommand](#) e come implementare il comando personalizzato tramite la creazione di un [ExecutedRoutedEventHandler](#) e una [CanExecuteRoutedEventHandler](#) e il ricollegamento di un [CommandBinding](#). Per altre informazioni sull'esecuzione di comandi, vedere la [Cenni preliminari](#).

Esempio

Il primo passaggio nella creazione di un [RoutedCommand](#) è la definizione di comando e crearne un'istanza.

```
public static RoutedCommand CustomRoutedCommand = new RoutedCommand();
```

```
Public Shared CustomRoutedCommand As New RoutedCommand()
```

Per usare il comando in un'applicazione, è necessario creare i gestori di eventi che definiscono il comando non

```
private void ExecutedCustomCommand(object sender,
    ExecutedRoutedEventArgs e)
{
    MessageBox.Show("Custom Command Executed");
}
```

```
Private Sub ExecutedCustomCommand(ByVal sender As Object, ByVal e As ExecutedRoutedEventArgs)
    MessageBox.Show("Custom Command Executed")
End Sub
```

```
// CanExecuteRoutedEventHandler that only returns true if
// the source is a control.
private void CanExecuteCustomCommand(object sender,
    CanExecuteRoutedEventArgs e)
{
    Control target = e.Source as Control;

    if(target != null)
    {
        e.CanExecute = true;
    }
    else
    {
        e.CanExecute = false;
    }
}
```

```

' CanExecuteRoutedEventHandler that only returns true if
' the source is a control.
Private Sub CanExecuteCustomCommand(ByVal sender As Object, ByVal e As CanExecuteRoutedEventArgs)
    Dim target As Control = TryCast(e.Source, Control)

    If target IsNot Nothing Then
        e.CanExecute = True
    Else
        e.CanExecute = False
    End If
End Sub

```

Successivamente, un [CommandBinding](#) viene creato il comando che associa i gestori di eventi. Il [CommandBinding](#) viene creato in un oggetto specifico. Questo oggetto definisce l'ambito del [CommandBinding](#) nell'albero degli elementi

```

<Window x:Class="SDKSamples.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:custom="clr-namespace:SDKSamples"
    Height="600" Width="800"
    >
<Window.CommandBindings>
    <CommandBinding Command="{x:Static custom:Window1.CustomRoutedCommand}"
                    Executed="ExecutedCustomCommand"
                    CanExecute="CanExecuteCustomCommand" />
</Window.CommandBindings>

```

```

CommandBinding customCommandBinding = new CommandBinding(
    CustomRoutedCommand, ExecutedCustomCommand, CanExecuteCustomCommand);

// attach CommandBinding to root window
this.CommandBindings.Add(customCommandBinding);

```

```

Dim customCommandBinding As New CommandBinding(CustomRoutedCommand, AddressOf ExecutedCustomCommand, AddressOf
CanExecuteCustomCommand)

' attach CommandBinding to root window
Me.CommandBindings.Add(customCommandBinding)

```

Il passaggio finale consiste nel richiamare il comando. Un modo per richiamare un comando consiste nell'associare a un [ICommandSource](#), ad esempio un [Button](#).

```

<StackPanel>
    <Button Command="{x:Static custom:Window1.CustomRoutedCommand}"
            Content="CustomRoutedCommand"/>
</StackPanel>

```

```

// create the ui
StackPanel CustomCommandStackPanel = new StackPanel();
Button CustomCommandButton = new Button();
CustomCommandStackPanel.Children.Add(CustomCommandButton);

CustomCommandButton.Command = CustomRoutedCommand;

```

```
' create the ui
Dim CustomCommandStackPanel As New StackPanel()
Dim CustomCommandButton As New Button()
CustomCommandStackPanel.Children.Add(CustomCommandButton)

CustomCommandButton.Command = CustomRoutedCommand
```

Quando si fa clic sul pulsante, il [Execute](#) metodo sull'oggetto personalizzato [RoutedCommand](#) viene chiamato. Il [RoutedCommand](#) genera il [PreviewExecuted](#) e [Executed](#) eventi indirizzati. Questi eventi attraversano l'albero degli elementi cercando un [CommandBinding](#) per questo particolare comando. Se un [CommandBinding](#) viene trovato, il [ExecutedRoutedEventHandler](#) associati [CommandBinding](#) viene chiamato.

Vedere anche

- [RoutedCommand](#)
- [Panoramica sull'esecuzione di comandi](#)

Procedura: implementare ICommandSource

08/01/2020 • 6 minutes to read • [Edit Online](#)

Questo esempio illustra come creare un'origine comando implementando [ICommandSource](#). Un'origine comando è un oggetto che sa come richiamare un comando. L'interfaccia [ICommandSource](#) espone tre membri:

- [Command](#): comando che verrà richiamato.
- [CommandParameter](#): tipo di dati definito dall'utente che viene passato dall'origine del comando al metodo che gestisce il comando.
- [CommandTarget](#): oggetto in cui viene eseguito il comando.

In questo esempio viene creata una classe che eredita dal controllo [Slider](#) e implementa l'interfaccia [ICommandSource](#).

Esempio

WPF offre una serie di classi che implementano [ICommandSource](#), ad esempio [Button](#), [MenuItem](#) e [Hyperlink](#). Un'origine comando definisce il modo in cui richiama un comando. Queste classi richiamano un comando quando vengono selezionate e diventano solo un'origine di comando quando viene impostata la relativa proprietà [Command](#).

In questo esempio verrà richiamato il comando quando il dispositivo di scorrimento viene spostato o più accuratamente quando viene modificata la proprietà [Value](#).

Di seguito è riportata la definizione della classe:

```
public class CommandSlider : Slider, ICommandSource
{
    public CommandSlider() : base()
    {
    }
}
```

```
Public Class CommandSlider
    Inherits Slider
    Implements ICommandSource
    Public Sub New()
        MyBase.New()
    End Sub
```

Il passaggio successivo consiste nell'implementare i membri [ICommandSource](#). In questo esempio, le proprietà vengono implementate come oggetti [DependencyProperty](#). Questo consente alle proprietà di utilizzare data binding. Per ulteriori informazioni sulla classe [DependencyProperty](#), vedere [Cenni preliminari sulle proprietà di dipendenza](#). Per ulteriori informazioni su data binding, vedere [Cenni preliminari sull'associazione dati](#).

Qui viene mostrata solo la proprietà [Command](#).

```

// Make Command a dependency property so it can use databinding.
public static readonly DependencyProperty CommandProperty =
    DependencyProperty.Register(
        "Command",
        typeof(ICommand),
        typeof(CommandSlider),
        new PropertyMetadata((ICommand)null,
            new PropertyChangedCallback(CommandChanged)));
}

public ICommand Command
{
    get
    {
        return (ICommand)GetValue(CommandProperty);
    }
    set
    {
        SetValue(CommandProperty, value);
    }
}

```

```

' Make Command a dependency property so it can use databinding.
Public Shared ReadOnly CommandProperty As DependencyProperty =
    DependencyProperty.Register("Command", GetType(ICommand),
        GetType(CommandSlider),
        New PropertyMetadata(CType(Nothing, ICommand),
            New PropertyChangedCallback(AddressOf CommandChanged)))

Public ReadOnly Property Command1() As ICommand Implements ICommandSource.Command
    Get
        Return CType(GetValue(CommandProperty), ICommand)
    End Get
End Property

Public Property Command() As ICommand
    Get
        Return CType(GetValue(CommandProperty), ICommand)
    End Get
    Set(ByVal value As ICommand)
        SetValue(CommandProperty, value)
    End Set
End Property

```

Di seguito è riportato il callback di modifica [DependencyProperty](#):

```

// Command dependency property change callback.
private static void CommandChanged(DependencyObject d,
    DependencyPropertyChangedEventArgs e)
{
    CommandSlider cs = (CommandSlider)d;
    cs.HookUpCommand((ICommand)e.OldValue, (ICommand)e.NewValue);
}

```

```

' Command dependency property change callback.
Private Shared Sub CommandChanged(ByVal d As DependencyObject, ByVal e As DependencyPropertyChangedEventArgs)
    Dim cs As CommandSlider = CType(d, CommandSlider)
    cs.HookUpCommand(CType(e.OldValue, ICommand), CType(e.NewValue, ICommand))
End Sub

```

Il passaggio successivo consiste nell'aggiungere e rimuovere il comando associato all'origine del comando. Non è possibile sovrascrivere la proprietà [Command](#) quando viene aggiunto un nuovo comando, perché i gestori eventi

associati al comando precedente, se presente, devono essere rimossi per primi.

```
// Add a new command to the Command Property.  
private void HookUpCommand(ICommand oldCommand, ICommand newCommand)  
{  
    // If oldCommand is not null, then we need to remove the handlers.  
    if (oldCommand != null)  
    {  
        RemoveCommand(oldCommand, newCommand);  
    }  
    AddCommand(oldCommand, newCommand);  
}  
  
// Remove an old command from the Command Property.  
private void RemoveCommand(ICommand oldCommand, ICommand newCommand)  
{  
    EventHandler handler = CanExecuteChanged;  
    oldCommand.CanExecuteChanged -= handler;  
}  
  
// Add the command.  
private void AddCommand(ICommand oldCommand, ICommand newCommand)  
{  
    EventHandler handler = new EventHandler(CanExecuteChanged);  
    canExecuteChangedHandler = handler;  
    if (newCommand != null)  
    {  
        newCommand.CanExecuteChanged += canExecuteChangedHandler;  
    }  
}
```

```
' Add a new command to the Command Property.  
Private Sub HookUpCommand(ByVal oldCommand As ICommand, ByVal newCommand As ICommand)  
    ' If oldCommand is not null, then we need to remove the handlers.  
    If oldCommand IsNot Nothing Then  
        RemoveCommand(oldCommand, newCommand)  
    End If  
    AddCommand(oldCommand, newCommand)  
End Sub  
  
' Remove an old command from the Command Property.  
Private Sub RemoveCommand(ByVal oldCommand As ICommand, ByVal newCommand As ICommand)  
    Dim handler As EventHandler = AddressOf CanExecuteChanged  
    RemoveHandler oldCommand.CanExecuteChanged, handler  
End Sub  
  
' Add the command.  
Private Sub AddCommand(ByVal oldCommand As ICommand, ByVal newCommand As ICommand)  
    Dim handler As New EventHandler(AddressOf CanExecuteChanged)  
    canExecuteChangedHandler = handler  
    If newCommand IsNot Nothing Then  
        AddHandler newCommand.CanExecuteChanged, canExecuteChangedHandler  
    End If  
End Sub
```

Il passaggio successivo consiste nel creare la logica per il gestore [CanExecuteChanged](#).

L'evento [CanExecuteChanged](#) notifica all'origine del comando che è possibile che la capacità del comando da eseguire sulla destinazione corrente del comando sia stata modificata. Quando l'origine di un comando riceve questo evento, in genere chiama il metodo [CanExecute](#) sul comando. Se non è possibile eseguire il comando sulla destinazione del comando corrente, l'origine comando si disabilita in genere. Se il comando può essere eseguito sulla destinazione del comando corrente, l'origine del comando in genere lo Abilita.

```

private void CanExecuteChanged(object sender, EventArgs e)
{
    if (this.Command != null)
    {
        RoutedCommand command = this.Command as RoutedCommand;

        // If a RoutedCommand.
        if (command != null)
        {
            if (command.CanExecute(CommandParameter, CommandTarget))
            {
                this.IsEnabled = true;
            }
            else
            {
                this.IsEnabled = false;
            }
        }
        // If a not RoutedCommand.
        else
        {
            if (Command.CanExecute(CommandParameter))
            {
                this.IsEnabled = true;
            }
            else
            {
                this.IsEnabled = false;
            }
        }
    }
}

```

```

Private Sub CanExecuteChanged(ByVal sender As Object, ByVal e As EventArgs)

If Me.Command IsNot Nothing Then
    Dim command As RoutedCommand = TryCast(Me.Command, RoutedCommand)

    ' If a RoutedCommand.
    If command IsNot Nothing Then
        If command.CanExecute(CommandParameter, CommandTarget) Then
            Me.IsEnabled = True
        Else
            Me.IsEnabled = False
        End If
        ' If a not RoutedCommand.
    Else
        If Me.Command.CanExecute(CommandParameter) Then
            Me.IsEnabled = True
        Else
            Me.IsEnabled = False
        End If
    End If
End If
End Sub

```

L'ultimo passaggio è il metodo [Execute](#). Se il comando è un [RoutedCommand](#), viene chiamato il metodo [Execute RoutedCommand](#); in caso contrario, viene chiamato il metodo [ICommand Execute](#).

```

// If Command is defined, moving the slider will invoke the command;
// Otherwise, the slider will behave normally.
protected override void OnValueChanged(double oldValue, double newValue)
{
    base.OnValueChanged(oldValue, newValue);

    if (this.Command != null)
    {
        RoutedCommand command = Command as RoutedCommand;

        if (command != null)
        {
            command.Execute(CommandParameter, CommandTarget);
        }
        else
        {
            ((ICommand)Command).Execute(CommandParameter);
        }
    }
}

```

```

' If Command is defined, moving the slider will invoke the command;
' Otherwise, the slider will behave normally.
Protected Overrides Sub OnValueChanged(ByVal oldValue As Double, ByVal newValue As Double)
    MyBase.OnValueChanged(oldValue, newValue)

    If Me.Command IsNot Nothing Then
        Dim command As RoutedCommand = TryCast(Me.Command, RoutedCommand)

        If command IsNot Nothing Then
            command.Execute(CommandParameter, CommandTarget)
        Else
            CType(Me.Command, ICommand).Execute(CommandParameter)
        End If
    End If
End Sub

```

Vedere anche

- [ICommandSource](#)
- [ICommand](#)
- [RoutedCommand](#)
- [Panoramica sull'esecuzione di comandi](#)

Procedura: Associare un comando a un controllo senza supporto dei comandi

23/10/2019 • 3 minutes to read • [Edit Online](#)

Nell'esempio seguente viene illustrato come associare una classe [RoutedCommand](#) a una classe [Control](#) che non dispone del supporto incorporato per il comando. Per un esempio completo in cui i comandi sono associati a più origini, vedere l'esempio [Create a Custom RoutedCommand Sample](#) (Creare un esempio di oggetto RoutedCommand personalizzato).

Esempio

Windows Presentation Foundation (WPF) fornisce una libreria di comandi comuni usati regolarmente dai programmatori di applicazioni. Le classi che costituiscono la libreria del comando sono: [ApplicationCommands](#), [ComponentCommands](#), [NavigationCommands](#), [MediaCommands](#) e [EditingCommands](#).

Gli oggetti [RoutedCommand](#) statici che costituiscono queste classi non forniscono la logica per il comando. La logica per il comando è associata al comando con un [CommandBinding](#). Molti controlli in WPF dispongono del supporto incorporato per alcuni comandi della libreria di comandi. [TextBox](#), ad esempio, supporta molti dei comandi di modifica dell'applicazione, ad esempio [Paste](#), [Copy](#), [Cut](#), [Redo](#) e [Undo](#). Lo sviluppatore dell'applicazione non deve eseguire operazioni particolari per garantire il funzionamento di questi comandi con tali controlli. Se la destinazione del comando eseguito è [TextBox](#), il comando verrà gestito utilizzando il [CommandBinding](#) incorporato nel controllo.

Nell'esempio seguente viene illustrato come utilizzare un [Button](#) come origine del comando [Open](#). Viene creata una classe [CommandBinding](#) che associa il metodo [CanExecuteRoutedEventArgs](#) specificato e il metodo [CanExecuteRoutedEventArgs](#) con la classe [RoutedCommand](#).

Innanzitutto, viene creata l'origine del comando. Una classe [Button](#) viene utilizzata come origine del comando.

```
<Button Command="ApplicationCommands.Open" Name="MyButton"
        Height="50" Width="200">
    Open (KeyBindings: Ctrl+R, Ctrl+0)
</Button>
```

```
// Button used to invoke the command
Button CommandButton = new Button();
CommandButton.Command = ApplicationCommands.Open;
CommandButton.Content = "Open (KeyBindings: Ctrl-R, Ctrl-0)";
MainStackPanel.Children.Add(CommandButton);
```

```
' Button used to invoke the command
Dim CommandButton As New Button()
CommandButton.Command = ApplicationCommands.Open
CommandButton.Content = "Open (KeyBindings: Ctrl-R, Ctrl-0)"
MainStackPanel.Children.Add(CommandButton)
```

Successivamente, vengono creati i metodi [ExecutedRoutedEventHandler](#) e [CanExecuteRoutedEventArgs](#). Il metodo [ExecutedRoutedEventArgs](#) apre semplicemente una classe [MessageBox](#) per indicare che il comando è stato eseguito. Il metodo [CanExecuteRoutedEventArgs](#) imposta la proprietà [CanExecute](#) su `true`. In genere, il gestore [CanExecute](#) eseguirebbe controlli più approfonditi per vedere se è possibile eseguire il comando sulla

destinazione del comando corrente.

```
void OpenCmdExecuted(object target, ExecutedRoutedEventArgs e)
{
    String command, targetobj;
    command = ((RoutedCommand)e.Command).Name;
    targetobj = ((FrameworkElement)target).Name;
    MessageBox.Show("The " + command + " command has been invoked on target object " + targetobj);
}
void OpenCmdCanExecute(object sender, CanExecuteRoutedEventArgs e)
{
    e.CanExecute = true;
}
```

```
Private Sub OpenCmdExecuted(ByVal sender As Object, ByVal e As ExecutedRoutedEventArgs)
    Dim command, targetobj As String
    command = CType(e.Command, RoutedCommand).Name
    targetobj = CType(sender, FrameworkElement).Name
    MessageBox.Show("The " + command + " command has been invoked on target object " + targetobj)
End Sub
Private Sub OpenCmdCanExecute(ByVal sender As Object, ByVal e As CanExecuteRoutedEventArgs)
    e.CanExecute = True
End Sub
```

Infine, viene creata una classe [CommandBinding](#) nella classe [Window](#) principale dell'applicazione che associa i gestori di eventi indirizzati al comando [Open](#).

```
<Window.CommandBindings>
    <CommandBinding Command="ApplicationCommands.Open"
                    Executed="OpenCmdExecuted"
                    CanExecute="OpenCmdCanExecute"/>
</Window.CommandBindings>
```

```
// Creating CommandBinding and attaching an Executed and CanExecute handler
CommandBinding OpenCmdBinding = new CommandBinding(
    ApplicationCommands.Open,
    OpenCmdExecuted,
    OpenCmdCanExecute);

this.CommandBindings.Add(OpenCmdBinding);
```

```
' Creating CommandBinding and attaching an Executed and CanExecute handler
Dim OpenCmdBinding As New CommandBinding(ApplicationCommands.Open, AddressOf OpenCmdExecuted, AddressOf
OpenCmdCanExecute)

Me.CommandBindings.Add(OpenCmdBinding)
```

Vedere anche

- [Panoramica sull'esecuzione di comandi](#)
- [Associare un comando a un controllo con supporto dei comandi](#)

Procedura: Associare un comando a un controllo con supporto del comando

23/10/2019 • 4 minutes to read • [Edit Online](#)

Nell'esempio seguente viene illustrato come associare una classe [RoutedCommand](#) a una classe [Control](#) che dispone del supporto incorporato per il comando. Per un esempio completo in cui i comandi sono associati a più origini, vedere l'esempio [Create a Custom RoutedCommand Sample](#) (Creare un esempio di oggetto RoutedCommand personalizzato).

Esempio

Windows Presentation Foundation (WPF) fornisce una libreria di comandi comuni usati regolarmente dai programmatori di applicazioni. Le classi che costituiscono la libreria del comando sono: [ApplicationCommands](#), [ComponentCommands](#), [NavigationCommands](#), [MediaCommands](#) e [EditingCommands](#).

Gli oggetti [RoutedCommand](#) statici che costituiscono queste classi non forniscono la logica per il comando. La logica per il comando è associata al comando con un [CommandBinding](#). Alcuni controlli dispongono di [CommandBinding](#) incorporati per determinati comandi. Questo meccanismo consente di mantenere invariata la semantica di un comando, mentre l'implementazione effettiva può cambiare. Un oggetto [TextBox](#), ad esempio, gestisce il comando [Paste](#) in modo diverso rispetto a un controllo progettato per supportare immagini, ma l'idea alla base dell'operazione di incollamento rimane invariata. La logica di comando non può essere fornita dal comando, bensì dal controllo o dall'applicazione.

Molti controlli in WPF dispongono del supporto incorporato per alcuni comandi della libreria di comandi. [TextBox](#), ad esempio, supporta molti dei comandi di modifica dell'applicazione, ad esempio [Paste](#), [Copy](#), [Cut](#), [Redo](#) e [Undo](#). Lo sviluppatore dell'applicazione non deve eseguire operazioni particolari per garantire il funzionamento di questi comandi con tali controlli. Se la destinazione del comando eseguito è [TextBox](#), il comando verrà gestito utilizzando il [CommandBinding](#) incorporato nel controllo.

Nell'esempio seguente viene illustrato come utilizzare un [MenuItem](#) come origine del comando [Paste](#) in cui [TextBox](#) è la destinazione del comando. Tutta la logica che definisce il modo in cui [TextBox](#) esegue Incolla è incorporata nel controllo [TextBox](#).

Viene creata una classe [MenuItem](#) e la relativa proprietà [Command](#) è impostata sul comando [Paste](#). La proprietà [CommandTarget](#) non è impostata esplicitamente sull'oggetto [TextBox](#). Quando [CommandTarget](#) non è impostata, la destinazione per il comando è l'elemento con lo stato attivo della tastiera. Se l'elemento con lo stato attivo della tastiera non supporta il comando [Paste](#) oppure non è attualmente in grado di eseguire il comando Incolla, ad esempio gli Appunti sono vuoti, [MenuItem](#) viene visualizzato in grigio.

```

<Window x:Class="SDKSamples.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MenuItemCommandTask"
    >
    <DockPanel>
        <Menu DockPanel.Dock="Top">
            <MenuItem Command="ApplicationCommands.Paste" Width="75" />
        </Menu>
        <TextBox BorderBrush="Black" BorderThickness="2" Margin="25"
            TextWrapping="Wrap">
            The MenuItem will not be enabled until
            this TextBox gets keyboard focus
        </TextBox>
    </DockPanel>
</Window>

```

```

// Window1 constructor
public Window1()
{
    InitializeComponent();

    // Instantiating UIElements.
    DockPanel mainPanel = new DockPanel();
    Menu mainMenu = new Menu();
    MenuItem pasteMenuItem = new MenuItem();
    TextBox mainTextBox = new TextBox();

    // Associating the MenuItem with the Paste command.
    pasteMenuItem.Command = ApplicationCommands.Paste;

    // Setting properties on the TextBox.
    mainTextBox.Text =
        "The MenuItem will not be enabled until this TextBox receives keyboard focus.";
    mainTextBox.Margin = new Thickness(25);
    mainTextBox.BorderBrush = Brushes.Black;
    mainTextBox.BorderThickness = new Thickness(2);
    mainTextBox.TextWrapping = TextWrapping.Wrap;

    // Attaching UIElements to the Window.
    this.AddChild(mainPanel);
    mainMenu.Items.Add(pasteMenuItem);
    mainPanel.Children.Add(mainMenu);
    mainPanel.Children.Add(mainTextBox);

    // Defining DockPanel layout.
    DockPanel.SetDock(mainMenu, Dock.Top);
    DockPanel.SetDock(mainTextBox, Dock.Bottom);
}

```

```

' Window1 constructor
Public Sub New()
    InitializeComponent()

    ' Instantiating UIElements.
    Dim mainPanel As New DockPanel()
    Dim mainMenu As New Menu()
    Dim pasteMenuItem As New MenuItem()
    Dim mainTextBox As New TextBox()

    ' Associating the MenuItem with the Paste command.
    pasteMenuItem.Command = ApplicationCommands.Paste

    ' Setting properties on the TextBox.
    mainTextBox.Text = "The MenuItem will not be enabled until this TextBox receives keyboard focus."
    mainTextBox.Margin = New Thickness(25)
    mainTextBox.BorderBrush = Brushes.Black
    mainTextBox.BorderThickness = New Thickness(2)
    mainTextBox.TextWrapping = TextWrapping.Wrap

    ' Attaching UIElements to the Window.
    Me.AddChild(mainPanel)
    mainMenu.Items.Add(pasteMenuItem)
    mainPanel.Children.Add(mainMenu)
    mainPanel.Children.Add(mainTextBox)

    ' Defining DockPanel layout.
    DockPanel.SetDock(mainMenu, Dock.Top)
    DockPanel.SetDock(mainTextBox, Dock.Bottom)
End Sub

```

Vedere anche

- [Panoramica sull'esecuzione di comandi](#)
- [Associare un comando a un controllo senza supporto del comando](#)

Input penna

23/10/2019 • 2 minutes to read • [Edit Online](#)

In questa sezione descrive l'uso dell'input penna digitale nella WPF. In passato disponibile solo nel SDK di Tablet PC, input penna digitale è ora disponibile in Windows Presentation Foundation di base. Ciò significa che è ora possibile sviluppare applicazioni per Tablet PC complete usando le potenzialità di Windows Presentation Foundation.

In questa sezione

[Panoramiche](#)

[Procedure relative alle proprietà](#)

Sezioni correlate

[Windows Presentation Foundation](#)

Cenni preliminari sull'input penna

23/10/2019 • 2 minutes to read • [Edit Online](#)

In questa sezione

[Introduzione all'input penna](#)

[Raccolta di input penna](#)

[Riconoscimento della grafia](#)

[Archiviazione dell'input penna](#)

[Modello a oggetti Ink: Windows Form e COM e WPF](#)

[Gestione avanzata dell'input penna](#)

Introduzione a input penna in WPF

03/02/2020 • 5 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) dispone di una funzionalità di input penna che semplifica l'incorporamento dell'input penna digitale nell'app.

Prerequisiti

Per usare gli esempi seguenti, installare innanzitutto [Visual Studio](#). Consente inoltre di comprendere come scrivere app WPF di base. Per informazioni introduttive su WPF, vedere [procedura dettagliata: applicazione desktop WPF](#).

Introduzione

Questa sezione consente di scrivere una semplice applicazione WPF che raccoglie input penna.

Hai un input penna?

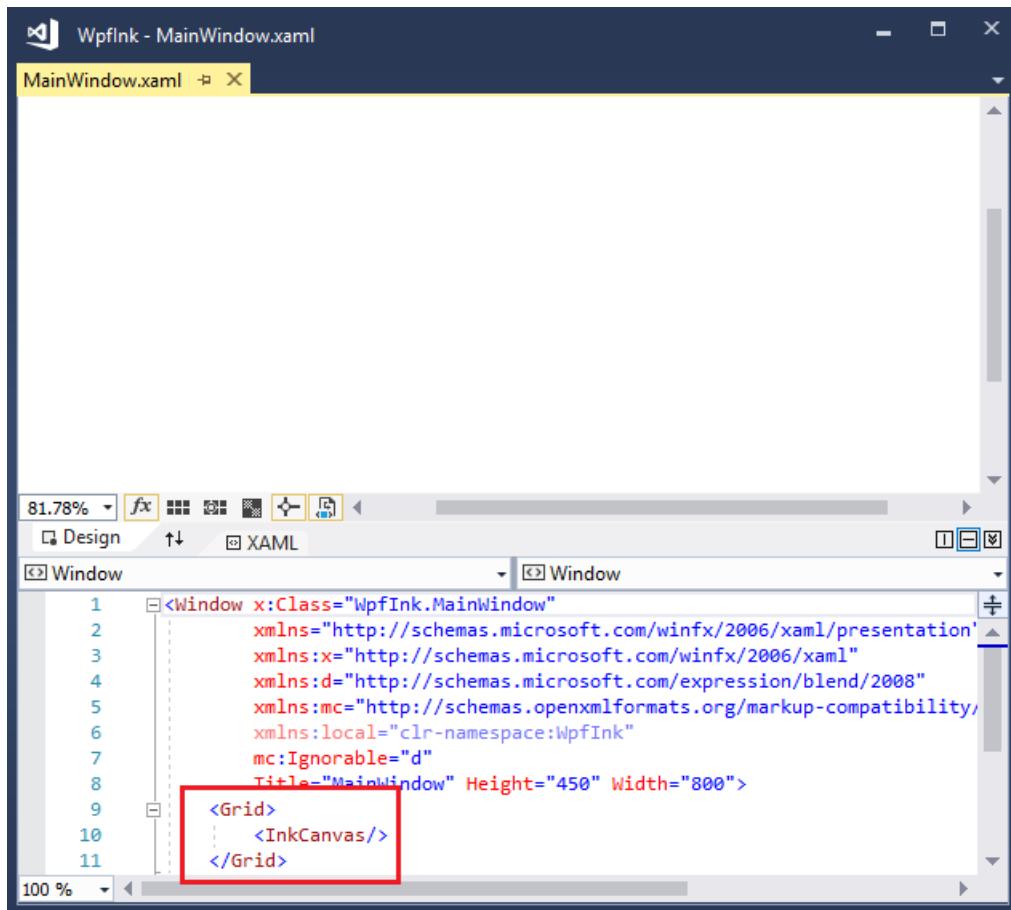
Per creare un'applicazione WPF che supporti l'input penna:

1. Apri Visual Studio.
2. Creare una nuova **app WPF**.

Nella finestra di dialogo **nuovo progetto** espandere la categoria **installato > Visual C# o Visual Basic > desktop di Windows**. Quindi, selezionare il modello app **WPF (.NET Framework)**. Immettere un nome e quindi fare clic su **OK**.

Visual Studio crea il progetto e *MainWindow.xaml* si apre nella finestra di progettazione.

3. Digitare `<InkCanvas/>` tra i tag di `<Grid>`.



4. Premere **F5** per avviare l'applicazione nel debugger.
5. Usando uno stilo o un mouse, scrivere **Hello World** nella finestra.

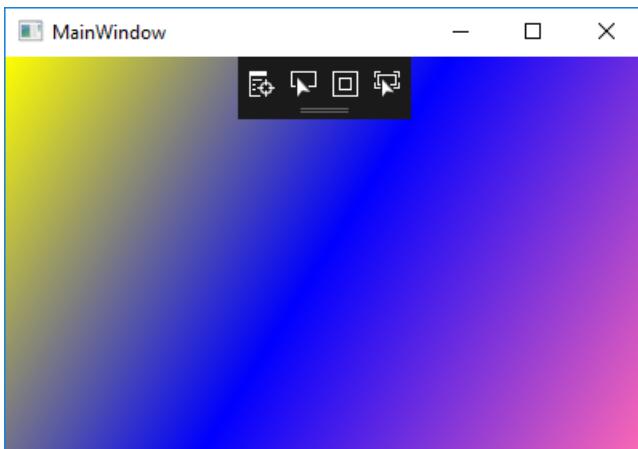
Hai scritto l'equivalente dell'input penna di un'applicazione "Hello World" con solo 12 sequenze di tasti.

Ravviva la tua app

È ora opportuno usufruire di alcune funzionalità di WPF. Sostituire tutti gli elementi tra la finestra di <di apertura e di chiusura > Tag con il markup seguente:

```
<Page>
<InkCanvas Name="myInkCanvas" MouseRightButtonUp="RightMouseUpHandler">
    <InkCanvas.Background>
        <LinearGradientBrush>
            <GradientStop Color="Yellow" Offset="0.0" />
            <GradientStop Color="Blue" Offset="0.5" />
            <GradientStop Color="HotPink" Offset="1.0" />
        </LinearGradientBrush>
    </InkCanvas.Background>
</InkCanvas>
</Page>
```

Questo codice XAML crea uno sfondo del pennello sfumatura sull'area di Inking.



Aggiungere codice sottostante a XAML

Mentre XAML rende molto semplice progettare l'interfaccia utente, qualsiasi applicazione reale deve aggiungere codice per gestire gli eventi. Ecco un semplice esempio che esegue lo zoom avanti sull'input penna in risposta a un clic con il pulsante destro del mouse.

1. Impostare il gestore `MouseRightButtonUp` in XAML:

```
<InkCanvas Name="myInkCanvas" MouseRightButtonUp="RightMouseUpHandler">
```

2. In **Esplora soluzioni** espandere `MainWindow.xaml` e aprire il file code-behind (`MainWindow.xaml.cs` o `MainWindow.xaml.vb`). Aggiungere il seguente codice del gestore eventi:

```
private void RightMouseUpHandler(object sender,
                                 System.Windows.Input.MouseEventArgs e)
{
    Matrix m = new Matrix();
    m.Scale(1.1d, 1.1d);
    ((InkCanvas)sender).Strokes.Transform(m, true);
}
```

```
Private Sub RightMouseUpHandler(ByVal sender As Object, _
                                ByVal e As System.Windows.Input.MouseEventArgs)

    Dim m As New Matrix()
    m.Scale(1.1, 1.1)
    CType(sender, InkCanvas).Strokes.Transform(m, True)

End Sub
```

3. Eseguire l'applicazione. Aggiungere un input penna, quindi fare clic con il pulsante destro del mouse con il mouse o eseguire un valore equivalente a premere e tenere premuto uno stilo.

La visualizzazione viene ingrandita ogni volta che si fa clic con il pulsante destro del mouse.

Usa codice procedurale anziché XAML

È possibile accedere a tutte le funzionalità WPF dal codice procedurale. Seguire questa procedura per creare un'applicazione "Hello Ink World" per WPF che non usa alcun codice XAML.

1. Creare un nuovo progetto di applicazione console in Visual Studio.

Nella finestra di dialogo **nuovo progetto** espandere la categoria **installato > Visual C# o Visual Basic > desktop di Windows**. Quindi, selezionare il modello app **Console (.NET Framework)**. Immettere un nome e quindi fare clic su **OK**.

2. Incollare il codice seguente nel file Program.cs o Program.vb:

```
using System;
using System.Windows;
using System.Windows.Controls;
class Program : Application
{
    Window win;
    InkCanvas ic;

    protected override void OnStartup(StartupEventArgs args)
    {
        base.OnStartup(args);
        win = new Window();
        ic = new InkCanvas();
        win.Content = ic;
        win.Show();
    }

    [STAThread]
    static void Main(string[] args)
    {
        new Program().Run();
    }
}
```

```
Imports System.Windows
Imports System.Windows.Controls

Class Program
    Inherits Application
    Private win As Window
    Private ic As InkCanvas

    Protected Overrides Sub OnStartup(ByVal args As StartupEventArgs)
        MyBase.OnStartup(args)
        win = New Window()
        ic = New InkCanvas()
        win.Content = ic
        win.Show()
    End Sub

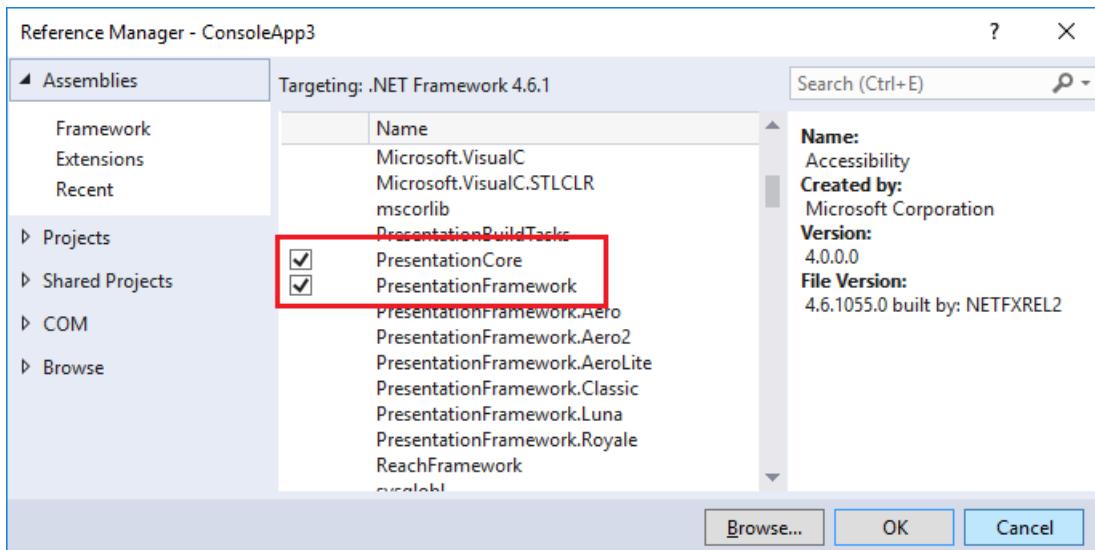
End Class

Module Module1

    Sub Main()
        Dim prog As New Program()
        prog.Run()
    End Sub

End Module
```

3. Aggiungere riferimenti agli assembly PresentationCore, PresentationFramework e WindowsBase facendo clic con il pulsante destro del mouse su **riferimenti** in **Esplora soluzioni** e scegliendo **Aggiungi riferimento**.



4. Compilare l'applicazione premendo **F5**.

Vedere anche

- [Input penna](#)
- [Raccolta di input penna](#)
- [Riconoscimento della grafia](#)
- [Archiviazione dell'input penna](#)

Raccogli input penna

03/02/2020 • 7 minutes to read • [Edit Online](#)

La raccolta di input penna è una delle funzionalità principali della piattaforma [Windows Presentation Foundation](#).

In questo argomento vengono illustrati i metodi per la raccolta di input penna in Windows Presentation Foundation (WPF).

Prerequisiti

Per usare gli esempi seguenti, è necessario innanzitutto installare Visual Studio e il Windows SDK. È inoltre necessario comprendere come scrivere applicazioni per WPF. Per ulteriori informazioni su come iniziare a usare WPF, vedere [procedura dettagliata: applicazione desktop WPF](#).

Usare l'elemento InkCanvas

L'elemento [System.Windows.Controls.InkCanvas](#) rappresenta il modo più semplice per raccogliere input penna in WPF. Usare un elemento [InkCanvas](#) per ricevere e visualizzare input penna. In genere, l'input penna viene effettuato tramite uno stilo che interagisce con un digitalizzatore per produrre i tratti. È anche possibile usare un mouse al posto dello stilo. I tratti creati sono rappresentati come oggetti [Stroke](#) e possono essere modificati sia a livello di codice sia in base all'input dell'utente. Il [InkCanvas](#) consente agli utenti di selezionare, modificare o eliminare un [Stroke](#) esistente.

Usando XAML, è possibile configurare la raccolta di input penna con la stessa facilità con cui si aggiunge un elemento [InkCanvas](#) all'albero. Nell'esempio seguente viene aggiunto un [InkCanvas](#) a un progetto WPF predefinito creato in Visual Studio:

```
<Grid>
    <InkCanvas/>
</Grid>
```

L'elemento [InkCanvas](#) può inoltre contenere elementi figlio, rendendo possibile l'aggiunta di funzionalità di annotazione Ink a quasi tutti i tipi di elementi XAML. Ad esempio, per aggiungere le funzionalità di Inking a un elemento di testo, è sufficiente renderlo figlio di un [InkCanvas](#):

```
<InkCanvas>
    <TextBlock>Show text here.</TextBlock>
</InkCanvas>
```

L'aggiunta del supporto per contrassegnare un'immagine con input penna è semplice:

```
<InkCanvas>
    <Image Source="myPicture.jpg"/>
</InkCanvas>
```

Modalità InkCollection

Il [InkCanvas](#) fornisce supporto per varie modalità di input tramite la relativa proprietà [EditingMode](#).

Modificare l'input penna

Il [InkCanvas](#) fornisce il supporto per molte operazioni di modifica dell'input penna. Ad esempio, [InkCanvas](#)

supporta la cancellazione della penna indietro e non è necessario alcun codice aggiuntivo per aggiungere la funzionalità all'elemento.

Selezione

Per impostare la modalità di selezione è sufficiente impostare la proprietà [InkCanvasEditingStyle](#) su **Selezione**.

Il codice seguente imposta la modalità di modifica in base al valore di un [CheckBox](#):

```
// Set the selection mode based on a checkbox
if ((bool)cbSelectionMode.IsChecked)
{
    theInkCanvas.EditingMode = InkCanvasEditingStyle.Select;
}
else
{
    theInkCanvas.EditingMode = InkCanvasEditingStyle.Ink;
}
```

```
' Set the selection mode based on a checkbox
If CBool(cbSelectionMode.IsChecked) Then
    theInkCanvas.EditingMode = InkCanvasEditingStyle.Select
Else
    theInkCanvas.EditingMode = InkCanvasEditingStyle.Ink
End If
```

DrawingAttributes

Utilizzare la proprietà [DrawingAttributes](#) per modificare l'aspetto dei tratti di input penna. Ad esempio, il membro [Color](#) di [DrawingAttributes](#) imposta il colore del [Stroke](#)sottoposto a rendering.

Nell'esempio seguente il colore dei tratti selezionati viene modificato in rosso:

```
// Get the selected strokes from the InkCanvas
StrokeCollection selection = theInkCanvas.GetSelectedStrokes();

// Check to see if any strokes are actually selected
if (selection.Count > 0)
{
    // Change the color of each stroke in the collection to red
    foreach (System.Windows.Ink.Stroke stroke in selection)
    {
        stroke.DrawingAttributes.Color = System.Windows.Media.Colors.Red;
    }
}
```

```
' Get the selected strokes from the InkCanvas
Dim selection As StrokeCollection = theInkCanvas.GetSelectedStrokes()

' Check to see if any strokes are actually selected
If selection.Count > 0 Then
    ' Change the color of each stroke in the collection to red
    Dim stroke As System.Windows.Ink.Stroke
    For Each stroke In selection
        stroke.DrawingAttributes.Color = System.Windows.Media.Colors.Red
    Next stroke
End If
```

DefaultDrawingAttributes

La proprietà [DefaultDrawingAttributes](#) fornisce l'accesso a proprietà quali l'altezza, la larghezza e il colore dei tratti da creare in una [InkCanvas](#). Una volta modificato il [DefaultDrawingAttributes](#), viene eseguito il rendering di tutti i

tratti successivi immessi nel [InkCanvas](#) con i nuovi valori delle proprietà.

Oltre a modificare il [DefaultDrawingAttributes](#) nel file code-behind, è possibile utilizzare la sintassi XAML per specificare [DefaultDrawingAttributes](#) proprietà.

Nell'esempio seguente viene illustrato come impostare la proprietà [Color](#). Per usare questo codice, creare un nuovo progetto WPF denominato "HelloInkCanvas" in Visual Studio. Sostituire il codice nel file *MainWindow.xaml* con il codice seguente:

```
<Window x:Class="HelloInkCanvas.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:Ink="clr-namespace:System.Windows.Ink;assembly=PresentationCore"
    Title="Hello, InkCanvas!" Height="300" Width="300"
    >
<Grid>
    <InkCanvas Name="inkCanvas1" Background="Ivory">
        <InkCanvas.DefaultDrawingAttributes>
            <Ink:DrawingAttributes xmlns:ink="system-windows-ink" Color="Red" Width="5" />
        </InkCanvas.DefaultDrawingAttributes>
    </InkCanvas>

    <!-- This stack panel of buttons is a sibling to InkCanvas (not a child) but overlapping it,
        higher in z-order, so that ink is collected and rendered behind -->
    <StackPanel Name="buttonBar" VerticalAlignment="Top" Height="26" Orientation="Horizontal" Margin="5">
        <Button Click="Ink">Ink</Button>
        <Button Click="Highlight">Highlight</Button>
        <Button Click="EraseStroke">EraseStroke</Button>
        <Button Click="Select">Select</Button>
    </StackPanel>
</Grid>
</Window>
```

Successivamente, aggiungere i gestori eventi del pulsante seguenti al file code-behind, all'interno della classe *MainWindow*:

```

// Set the EditingMode to ink input.
private void Ink(object sender, RoutedEventArgs e)
{
    inkCanvas1.EditingMode = InkCanvasEditingMode.Ink;

    // Set the DefaultDrawingAttributes for a red pen.
    inkCanvas1.DefaultDrawingAttributes.Color = Colors.Red;
    inkCanvas1.DefaultDrawingAttributes.IsHighlighter = false;
    inkCanvas1.DefaultDrawingAttributes.Height = 2;
}

// Set the EditingMode to highlighter input.
private void Highlight(object sender, RoutedEventArgs e)
{
    inkCanvas1.EditingMode = InkCanvasEditingMode.Ink;

    // Set the DefaultDrawingAttributes for a highlighter pen.
    inkCanvas1.DefaultDrawingAttributes.Color = Colors.Yellow;
    inkCanvas1.DefaultDrawingAttributes.IsHighlighter = true;
    inkCanvas1.DefaultDrawingAttributes.Height = 25;
}

// Set the EditingMode to erase by stroke.
private void EraseStroke(object sender, RoutedEventArgs e)
{
    inkCanvas1.EditingMode = InkCanvasEditingMode.EraseByStroke;
}

// Set the EditingMode to selection.
private void Select(object sender, RoutedEventArgs e)
{
    inkCanvas1.EditingMode = InkCanvasEditingMode.Select;
}

```

Dopo aver copiato il codice, premere **F5** in Visual Studio per eseguire il programma nel debugger.

Si noti come il [StackPanel](#) posiziona i pulsanti sulla parte superiore del [InkCanvas](#). Se si tenta di eseguire l'input penna sulla parte superiore dei pulsanti, il [InkCanvas](#) raccoglie ed esegue il rendering dell'input penna dietro i pulsanti. Il motivo è che i pulsanti sono elementi di pari livello del [InkCanvas](#) anziché degli elementi figlio. Il rendering dell'input penna viene eseguito dietro i pulsanti anche perché si trovano più in alto nel z order.

Vedere anche

- [DrawingAttributes](#)
- [DefaultDrawingAttributes](#)
- [System.Windows.Ink](#)

Riconoscimento della grafia

23/10/2019 • 2 minutes to read • [Edit Online](#)

Questa sezione illustra alcune nozioni di base del riconoscimento relativamente all'input penna digitale nella piattaforma WPF.

Soluzioni di riconoscimento

Nell'esempio seguente viene mostrato come riconoscere l'input penna usando la classe `Microsoft.Ink.InkCollector`.

NOTE

Per questo esempio è necessario che il riconoscimento grafia sia installato nel sistema.

Creare un nuovo progetto di applicazione WPF in Visual Studio denominato **InkRecognition**. Sostituire il contenuto del file Window1.xaml con il codice XAML seguente. Questo codice esegue il rendering dell'interfaccia utente dell'applicazione.

```
<Window x:Class="InkRecognition.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="InkRecognition"
    >
<Canvas Name="theRootCanvas">
    <Border
        Background="White"
        BorderBrush="Black"
        BorderThickness="2"
        Height="300"
        Width="300"
        Canvas.Top="10"
        Canvas.Left="10">
        <InkCanvas Name="theInkCanvas"></InkCanvas>
    </Border>
    <TextBox Name="textBox1"
        Height="25"
        Width="225"
        Canvas.Top="325"
        Canvas.Left="10"></TextBox>
    <Button
        Height="25"
        Width="75"
        Canvas.Top="325"
        Canvas.Left="235"
        Click="buttonClick">Recognize</Button>
    <Button x:Name="btnClear" Content="Clear Canvas" Canvas.Left="10" Canvas.Top="367" Width="75"
        Click="btnClear_Click"/>
</Canvas>
</Window>
```

Aggiungere un riferimento all'assembly Microsoft Ink Microsoft.Ink.dll, che può trovarsi in \Programmi\File comuni\Microsoft Shared\Ink. Sostituire il contenuto del file code-behind con il codice seguente.

```
using System.Windows;
using Microsoft.Ink;
using System.IO;

namespace InkRecognition
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>

    public partial class Window1 : Window
    {

        public Window1()
        {
            InitializeComponent();
        }

        // Recognizes handwriting by using RecognizerContext
        private void buttonClick(object sender, RoutedEventArgs e)
        {
            using (MemoryStream ms = new MemoryStream())
            {
                theInkCanvas.Strokes.Save(ms);
                var myInkCollector = new InkCollector();
                var ink = new Ink();
                ink.Load(ms.ToArray());

                using (RecognizerContext context = new RecognizerContext())
                {
                    if (ink.Strokes.Count > 0)
                    {
                        context.Strokes = ink.Strokes;
                        RecognitionStatus status;

                        var result = context.Recognize(out status);

                        if (status == RecognitionStatus.NoError)
                            textBox1.Text = result.TopString;
                        else
                            MessageBox.Show("Recognition failed");
                    }
                    else
                    {
                        MessageBox.Show("No stroke detected");
                    }
                }
            }
        }

        private void btnClear_Click(object sender, RoutedEventArgs e) {
            theInkCanvas.Strokes.Clear();
        }
    }
}
```

```

Imports System.Windows
Imports Microsoft.Ink
Imports System.IO

'<summary>
' Interaction logic for Window1.xaml
'</summary>

Namespace InkRecognition

    Class Window1
        Inherits Window

        Public Sub New()
            InitializeComponent()
        End Sub

        ' Recognizes handwriting by using RecognizerContext
        Private Sub buttonClick(ByVal sender As Object, ByVal e As RoutedEventArgs)

            Using ms As New MemoryStream()
                theInkCanvas.Strokes.Save(ms)
                Dim myInkCollector As InkCollector = New InkCollector()
                Dim ink As Ink = New Ink()
                ink.Load(ms.ToArray())

                Using context As New RecognizerContext()
                    If ink-strokes.Count > 0 Then
                        context.Strokes = ink-strokes
                        Dim status As RecognitionStatus

                        Dim result As RecognitionResult = context.Recognize(status)

                        If status = RecognitionStatus.NoError Then
                            textBox1.Text = result.TopString
                        Else
                            MessageBox.Show("Recognition failed")
                        End If
                    Else
                        MessageBox.Show("No stroke detected")
                    End If
                End Using
            End Using
        End Sub

        Private Sub btnClear_Click(sender As Object, e As RoutedEventArgs)
            theInkCanvas.Strokes.Clear()
        End Sub
    End Class
End Namespace

```

Vedere anche

- [Microsoft.Ink.InkCollector](#)

Memorizzazione dell'input penna

23/10/2019 • 2 minutes to read • [Edit Online](#)

Il [Save](#) metodi forniscono supporto per l'archiviazione dell'input penna in formato ISF (Ink Serialized Format). I costruttori per la [StrokeCollection](#) classi forniscono supporto per la lettura dei dati di input penna.

Archiviazione dell'input penna e il recupero

Questa sezione illustra come archiviare e recuperare l'input penna nel WPF piattaforma.

L'esempio seguente implementa un gestore eventi clic del pulsante che presenta all'utente una finestra di dialogo Salva File e Salva l'input penna da un [InkCanvas](#) out in un file.

```
private void buttonSaveAsClick(object sender, RoutedEventArgs e)
{
    SaveFileDialog saveFileDialog1 = new SaveFileDialog();
    saveFileDialog1.Filter = "isf files (*.isf)|*.isf";

    if (saveFileDialog1.ShowDialog() == true)
    {
        FileStream fs = new FileStream(saveFileDialog1.FileName,
                                         FileMode.Create);
        theInkCanvas.Strokes.Save(fs);
        fs.Close();
    }
}
```

```
Private Sub buttonSaveAsClick(ByVal sender As Object, ByVal e As RoutedEventArgs)

    Dim saveFileDialog1 As New SaveFileDialog()
    saveFileDialog1.Filter = "isf files (*.isf)|*.isf"

    If saveFileDialog1.ShowDialog() Then
        Dim fs As New FileStream(saveFileDialog1.FileName, FileMode.Create)
        theInkCanvas.Strokes.Save(fs)
        fs.Close()
    End If

End Sub
```

L'esempio seguente implementa un gestore eventi clic del pulsante che presenta all'utente una finestra di dialogo Apri File e legge l'input penna dal file in un [InkCanvas](#) elemento.

```
private void buttonLoadClick(object sender, RoutedEventArgs e)
{
    OpenFileDialog openFileDialog1 = new OpenFileDialog();
    openFileDialog1.Filter = "isf files (*.isf)|*.isf";

    if (openFileDialog1.ShowDialog() == true)
    {
        FileStream fs = new FileStream(openFileDialog1.FileName,
                                         FileMode.Open);
        theInkCanvas.Strokes = new StrokeCollection(fs);
        fs.Close();
    }
}
```

```
Private Sub buttonLoadClick(ByVal sender As Object, ByVal e As RoutedEventArgs)

    Dim openFileDialog1 As New OpenFileDialog()
    openFileDialog1.Filter = "isf files (*.isf)|*.isf"

    If openFileDialog1.ShowDialog() Then
        Dim fs As New FileStream(openFileDialog1.FileName, FileMode.Open)
        theInkCanvas.Strokes = New StrokeCollection(fs)
        fs.Close()
    End If

End Sub
```

Vedere anche

- [InkCanvas](#)
- [Windows Presentation Foundation](#)

Modello a oggetti Ink: confronto di Windows Form e COM con WPF

03/02/2020 • 12 minutes to read • [Edit Online](#)

Sono essenzialmente disponibili tre piattaforme che supportano l'input penna digitale: la piattaforma Windows Forms Tablet PC, la piattaforma COM per Tablet PC e la piattaforma Windows Presentation Foundation (WPF). Le piattaforme Windows Forms e COM condividono un modello a oggetti simile, ma il modello a oggetti per la piattaforma WPF è sostanzialmente diverso. In questo argomento vengono illustrate le differenze a livello generale, in modo che gli sviluppatori che hanno lavorato con un modello a oggetti possano comprendere meglio l'altro.

Abilitazione di input penna in un'applicazione

Tutte e tre le piattaforme inviano oggetti e controlli che consentono a un'applicazione di ricevere input da una penna del tablet. Le piattaforme Windows Forms e COM vengono fornite con le classi [Microsoft.Ink.InkPicture](#), [Microsoft.Ink.InkEdit](#), [Microsoft.Ink.InkOverlay](#) e [Microsoft.Ink.InkCollector](#). [Microsoft.Ink.InkPicture](#) e [Microsoft.Ink.InkEdit](#) sono controlli che è possibile aggiungere a un'applicazione per raccogliere input penna. È possibile collegare [Microsoft.Ink.InkOverlay](#) e [Microsoft.Ink.InkCollector](#) a una finestra esistente per abilitare l'input penna per finestre e controlli personalizzati.

La piattaforma WPF include il controllo [InkCanvas](#). È possibile aggiungere un [InkCanvas](#) all'applicazione e iniziare a raccogliere immediatamente l'input penna. Con il [InkCanvas](#), l'utente può copiare, selezionare e ridimensionare l'input penna. È possibile aggiungere altri controlli all'[InkCanvas](#) l'utente può scrivere anche su tali controlli. È possibile creare un controllo personalizzato abilitato per l'input penna aggiungendo un [InkPresenter](#) e raccogliendo i punti dello stilo.

Nella tabella seguente sono elencate le informazioni su come abilitare l'input penna in un'applicazione:

PER ESEGUIRE QUESTA OPERAZIONE...	SULLA PIATTAFORMA WPF...	SULLE PIATTAFORME WINDOWS FORMS/COM...
Aggiungere un controllo abilitato per l'input penna a un'applicazione	Vedere Introduzione con input penna .	Vedere l'esempio di modulo Claims automatico
Abilitare l'input penna in un controllo personalizzato	Vedere creazione di un controllo input penna .	Vedere l'esempio di appunti input penna .

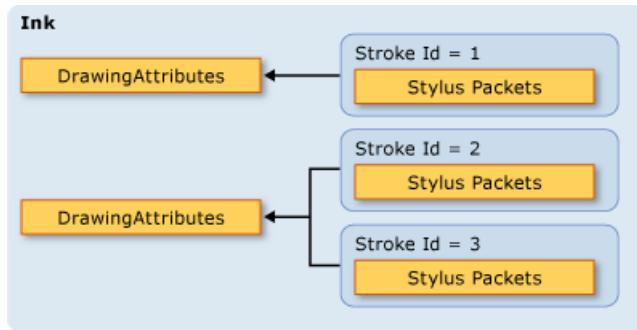
Dati Ink

Nelle piattaforme Windows Forms e COM [Microsoft.Ink.InkCollector](#), [Microsoft.Ink.InkOverlay](#), [Microsoft.Ink.InkEdit](#) [Microsoft.Ink.InkPicture](#) espongono ogni oggetto [Microsoft.Ink.Ink](#). L'oggetto [Microsoft.Ink.Ink](#) contiene i dati per uno o più oggetti [Microsoft.Ink.Stroke](#) ed espone metodi e proprietà comuni per gestire e modificare tali tratti. L'oggetto [Microsoft.Ink.Ink](#) gestisce la durata dei tratti che contiene; l'oggetto [Microsoft.Ink.Ink](#) crea ed Elimina i tratti di sua proprietà. Ogni [Microsoft.Ink.Stroke](#) dispone di un identificatore univoco all'interno dell'oggetto padre [Microsoft.Ink.Ink](#).

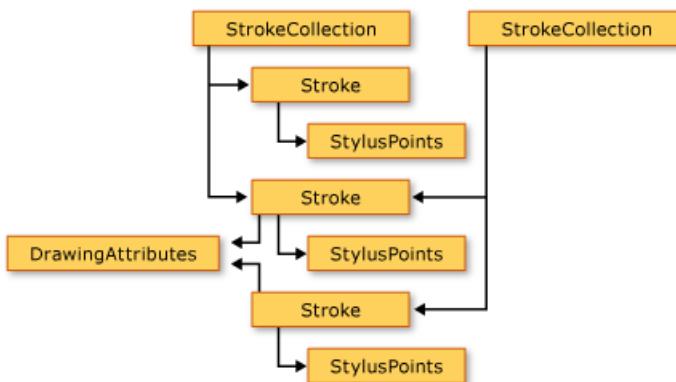
Nella piattaforma WPF la classe [System.Windows.Ink.Stroke](#) possiede e gestisce la propria durata. Un gruppo di oggetti [Stroke](#) può essere raccolto insieme in una [StrokeCollection](#), che fornisce metodi per operazioni di gestione dei dati di input penna comuni, ad esempio hit testing, cancellazione, trasformazione e serializzazione dell'input penna. Un [Stroke](#) può appartenere a zero, uno o più oggetti [StrokeCollection](#) in qualsiasi momento. Anziché disporre di un oggetto [Microsoft.Ink.Ink](#), il [InkCanvas](#) e [InkPresenter](#) contengono un

System.Windows.Ink.StrokeCollection.

Nella coppia di illustrazioni seguente vengono confrontati i modelli a oggetti dati dell'input penna. Sulle piattaforme Windows Forms e COM, l'oggetto [Microsoft. Ink. Ink](#) vincola la durata degli oggetti [Microsoft. Ink. Stroke](#) e i pacchetti dello stilo appartengono ai singoli tratti. Due o più tratti possono fare riferimento allo stesso oggetto [Microsoft. Ink. DrawingAttributes](#), come illustrato nella figura seguente.



Nel WPF, ogni [System.Windows.Ink.Stroke](#) è un oggetto Common Language Runtime esistente a condizione che vi sia un riferimento. Ogni [Stroke](#) fa riferimento a un oggetto [StylusPointCollection](#) e [System.Windows.Ink.DrawingAttributes](#), che sono anche Common Language Runtime oggetti.



Nella tabella seguente viene illustrato come eseguire alcune attività comuni sulla piattaforma WPF e sulle piattaforme Windows Forms e COM.

ATTIVITÀ	WINDOWS PRESENTATION FOUNDATION	WINDOWS FORMS E COM
Salva input penna	Save	Microsoft. Ink. Ink. Save
Carica input penna	Creare una StrokeCollection con il costruttore di StrokeCollection .	Microsoft. Ink. Ink. Load
Hit test	HitTest	Microsoft. Ink. Ink. HitTest
Copia input penna	CopySelection	Microsoft. Ink. Ink. ClipboardCopy
Incolla input penna	Paste	Microsoft. Ink. Ink. ClipboardPaste
Accedere a proprietà personalizzate in una raccolta di tratti	AddPropertyData (le proprietà vengono archiviate internamente ed è possibile accedervi tramite AddPropertyData , RemovePropertyData e ContainsPropertyData)	Utilizzare Microsoft. Ink. Ink. ExtendedProperties

Condivisione di input penna tra piattaforme

Anche se le piattaforme hanno modelli a oggetti diversi per i dati di input penna, la condivisione dei dati tra le

piattaforme è molto semplice. Gli esempi seguenti salvano l'input penna da un Windows Forms Application e caricano l'input penna in un'applicazione Windows Presentation Foundation.

```
using Microsoft.Ink;
using System.Drawing;
```

```
Imports Microsoft.Ink
Imports System.Drawing
```

```
/// <summary>
/// Saves the digital ink from a Windows Forms application.
/// </summary>
/// <param name="inkToSave">An Ink object that contains the
/// digital ink.</param>
/// <returns>A MemoryStream containing the digital ink.</returns>
MemoryStream SaveInkInWinforms(Ink inkToSave)
{
    byte[] savedInk = inkToSave.Save();

    return (new MemoryStream(savedInk));
}
```

```
' / <summary>
' / Saves the digital ink from a Windows Forms application.
' / </summary>
' / <param name="inkToSave">An Ink object that contains the
' / digital ink.</param>
' / <returns>A MemoryStream containing the digital ink.</returns>
Function SaveInkInWinforms(ByVal inkToSave As Ink) As MemoryStream
    Dim savedInk As Byte() = inkToSave.Save()

    Return New MemoryStream(savedInk)

End Function 'SaveInkInWinforms
```

```
using System.Windows.Ink;
```

```
Imports System.Windows.Ink
```

```
/// <summary>
/// Loads digital ink into a StrokeCollection, which can be
/// used by a WPF application.
/// </summary>
/// <param name="savedInk">A MemoryStream containing the digital ink.</param>
public void LoadInkInWPF(MemoryStream inkStream)
{
    strokes = new StrokeCollection(inkStream);
}
```

```

' / <summary>
' / Loads digital ink into a StrokeCollection, which can be
' / used by a WPF application.
' / </summary>
' / <param name="savedInk">A MemoryStream containing the digital ink.</param>
Public Sub LoadInkInWPF(ByVal inkStream As MemoryStream)
    strokes = New StrokeCollection(inkStream)

End Sub

```

Gli esempi seguenti salvano l'input penna da un'applicazione Windows Presentation Foundation e caricano l'input penna in una Windows Forms Application.

```
using System.Windows.Ink;
```

```
Imports System.Windows.Ink
```

```

/// <summary>
/// Saves the digital ink from a WPF application.
/// </summary>
/// <param name="inkToSave">A StrokeCollection that contains the
/// digital ink.</param>
/// <returns>A MemoryStream containing the digital ink.</returns>
MemoryStream SaveInkInWPF(StrokeCollection strokesToSave)
{
    MemoryStream savedInk = new MemoryStream();

    strokesToSave.Save(savedInk);

    return savedInk;
}

```

```

' / <summary>
' / Saves the digital ink from a WPF application.
' / </summary>
' / <param name="inkToSave">A StrokeCollection that contains the
' / digital ink.</param>
' / <returns>A MemoryStream containing the digital ink.</returns>
Function SaveInkInWPF(ByVal strokesToSave As StrokeCollection) As MemoryStream
    Dim savedInk As New MemoryStream()

    strokesToSave.Save(savedInk)

    Return savedInk
End Function 'SaveInkInWPF

```

```
using Microsoft.Ink;
using System.Drawing;
```

```
Imports Microsoft.Ink
Imports System.Drawing
```

```

/// <summary>
/// Loads digital ink into a Windows Forms application.
/// </summary>
/// <param name="savedInk">A MemoryStream containing the digital ink.</param>
public void LoadInkInWinforms(MemoryStream savedInk)
{
    theInk = new Ink();
    theInk.Load(savedInk.ToArray());
}

```

```

'<summary>
' Loads digital ink into a Windows Forms application.
'</summary>
'<param name="savedInk">A MemoryStream containing the digital ink.</param>
Public Sub LoadInkInWinforms(ByVal savedInk As MemoryStream)
    theInk = New Ink()
    theInk.Load(savedInk.ToArray())
End Sub

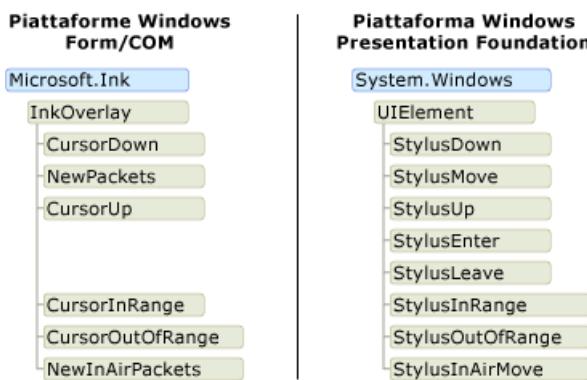
```

Eventi della penna del Tablet

[Microsoft. Ink. InkOverlay](#), [Microsoft. Ink. InkCollectore](#) [Microsoft. ink. INKPICTURE](#) sulle piattaforme Windows Forms e com ricevono eventi quando l'utente immette dati penna. [Microsoft. Ink. InkOverlay](#) o [Microsoft. Ink. InkCollector](#) è associato a una finestra o a un controllo e può sottoscrivere gli eventi generati dai dati di input del tablet. Il thread in cui si verificano questi eventi varia a seconda che gli eventi vengano generati con una penna, un mouse o a livello di codice. Per ulteriori informazioni sul threading in relazione a questi eventi, vedere [considerazioni generali relative al threading e thread sui quali un evento può essere attivato](#).

Nella piattaforma Windows Presentation Foundation la classe [UIElement](#) dispone di eventi per l'input penna. Ciò significa che ogni controllo espone il set completo di eventi dello stilo. Gli eventi dello stilo hanno coppie di eventi di tunneling/bubbling e si verificano sempre nel thread dell'applicazione. Per ulteriori informazioni, vedere [Cenni preliminari sugli eventi indirizzati](#).

Il diagramma seguente mostra un confronto tra i modelli a oggetti per le classi che generano eventi dello stilo. Il modello a oggetti Windows Presentation Foundation Mostra solo gli eventi di bubbling, non le controparti degli eventi di tunneling.



Dati penna

Tutte e tre le piattaforme forniscono modi per intercettare e modificare i dati provenienti da una penna del tablet. Sulle piattaforme Windows Forms e COM, questa operazione viene eseguita creando un oggetto [Microsoft. StylusInput. RealTimeStylus](#), collegando una finestra o un controllo e creando una classe che implementa l'interfaccia [Microsoft. StylusInput. IStylusSyncPlugin](#) o [Microsoft. StylusInput. IStylusAsyncPlugin](#). Il plug-in personalizzato viene quindi aggiunto alla raccolta plug-in di [Microsoft. StylusInput. RealTimeStylus](#). Per altre

informazioni su questo modello a oggetti, vedere [architettura delle API StylusInput](#).

Nella piattaforma WPF la classe [UIElement](#) espone una raccolta di plug-in, simile in progettazione a [Microsoft StylusInput](#). [RealTimeStylus](#). Per intercettare i dati di penna, creare una classe che erediti da [StylusPlugIn](#) e aggiungere l'oggetto alla raccolta [StylusPlugins](#) dell'[UIElement](#). Per altre informazioni su questa interazione, vedere [intervento dell'input dallo stilo](#).

In tutte le piattaforme, un pool di thread riceve i dati di input penna tramite gli eventi dello stilo e li invia al thread dell'applicazione. Per altre informazioni sul threading sulle piattaforme COM e Windows, vedere [considerazioni relative al threading per le API StylusInput](#). Per ulteriori informazioni sul threading sul software Windows Presentation, vedere [il modello di threading dell'input penna](#).

Nella figura seguente vengono confrontati i modelli a oggetti per le classi che ricevono dati penna nel pool di thread della penna.



Gestione avanzata dell'input penna

23/10/2019 • 2 minutes to read • [Edit Online](#)

Il WPF viene fornito con il [InkCanvas](#), ed è un elemento che è possibile inserire nell'applicazione per iniziare immediatamente a raccogliere e visualizzare input penna. Tuttavia, se il [InkCanvas](#) controllo non fornisce un livello sufficiente di controllo, è possibile mantenere il controllo a un livello superiore personalizzando proprie classi di raccolta e input penna per il rendering usando [System.Windows.Input.StylusPlugIns](#).

Il [System.Windows.Input.StylusPlugIns](#) classi forniscono un meccanismo per l'implementazione del controllo di basso livello tramite [Stylus](#) input e il rendering dinamico dell'input penna. Il [StylusPlugIn](#) classe fornisce un meccanismo per implementare il comportamento personalizzato e applicarlo al flusso di dati provenienti dal dispositivo stilo per prestazioni ottimali. Il [DynamicRenderer](#), specializzata [StylusPlugIn](#), consente di personalizzare in modo dinamico i dati di input penna per il rendering in tempo reale in altri termini, il [DynamicRenderer](#) disegna input penna digitale immediatamente come [StylusPoint](#) dati vengono generati in modo che sembri "fluire" dallo stilo dispositivo.

In questa sezione

[Personalizzare il rendering dell'input penna](#)

[Intercettazione dell'input dello stilo](#)

[Creazione di un controllo di input penna](#)

[Modello di threading dell'input penna](#)

Personalizzare il rendering dell'input penna

23/10/2019 • 10 minutes to read • [Edit Online](#)

Il [DrawingAttributes](#) proprietà di un tratto consente di specificare l'aspetto di un tratto, ad esempio dimensioni, colore e forma, ma potrebbe capitare che si desidera personalizzare l'aspetto oltre ciò che [DrawingAttributes](#) consentire. È possibile personalizzare l'aspetto dell'input penna eseguendo il rendering con l'aspetto di un aerografo, di una pittura a olio e di molti altri effetti. Windows Presentation Foundation (WPF) consente di personalizzare il rendering dell'input penna implementando una classe personalizzata [DynamicRenderer](#) e [Stroke](#) oggetto.

In questo argomento sono contenute le seguenti sottosezioni:

- [Architettura](#)
- [Implementazione di un renderer dinamico](#)
- [Implementazione di tratti personalizzati](#)
- [Implementazione di un oggetto InkCanvas personalizzato](#)
- [Conclusione](#)

Architettura

Il rendering dell'input penna viene eseguito due volte: quando l'utente scrive tramite una penna su un'apposita superficie e dopo l'aggiunta del tratto alla superficie abilitata per l'input penna. Il [DynamicRenderer](#) esegue il rendering dell'input penna quando l'utente sposta la penna del tablet sul digitalizzatore e il [Stroke](#) esegue il rendering di se stesso dopo essere stato aggiunto a un elemento.

Per il rendering dinamico dell'input penna è necessario implementare tre classi.

1. **DynamicRenderer:** Implementare una classe che derivi da [DynamicRenderer](#). Questa classe è un'oggetto specifico [StylusPlugIn](#) che esegue il rendering del tratto mentre viene tracciato. Il [DynamicRenderer](#) esegue il rendering su un thread separato, in modo che la superficie di input penna sembra raccogliere l'input penna anche quando viene bloccato il thread dell'interfaccia utente dell'applicazione. Per altre informazioni sul modello di threading, vedere [Modello di threading dell'input penna](#). Per personalizzare il rendering dinamico di un tratto, eseguire l'override di [OnDraw](#) (metodo).
2. **Stroke:** Implementare una classe che derivi da [Stroke](#). Questa classe è responsabile del rendering statico del [StylusPoint](#) dati dopo che è stato convertito in un [Stroke](#) oggetto. Eseguire l'override di [DrawCore](#) metodo per assicurarsi che il rendering statico del tratto sia coerente con il rendering dinamico.
3. **InkCanvas:** Implementare una classe che derivi da [InkCanvas](#). Assegnare l'oggetto personalizzato [DynamicRenderer](#) per il [DynamicRenderer](#) proprietà. Eseguire l'override di [OnStrokeCollected](#) metodo e aggiungere un tratto personalizzato per il [Strokes](#) proprietà. In questo modo, è possibile assicurarsi che l'aspetto dell'input penna sia coerente.

Implementazione di un renderer dinamico

Anche se il [DynamicRenderer](#) classe è una parte standard del WPF, per eseguire più specializzato per il rendering, è necessario creare un renderer dinamico personalizzato che deriva dalle [DynamicRenderer](#) ed eseguire l'override di [OnDraw](#) (metodo).

L'esempio seguente illustra un oggetto personalizzato [DynamicRenderer](#) che disegna l'input penna con un effetto

pennello a sfumatura lineare.

```
using System;
using System.Windows.Media;
using System.Windows;
using System.Windows.Input.StylusPlugIns;
using System.Windows.Input;
using System.Windows.Ink;
```

```
Imports System.Windows.Media
Imports System.Windows
Imports System.Windows.Input.StylusPlugIns
Imports System.Windows.Input
Imports System.Windows.Ink
```

```

// A StylusPlugin that renders ink with a linear gradient brush effect.
class CustomDynamicRenderer : DynamicRenderer
{
    [ThreadStatic]
    static private Brush brush = null;

    [ThreadStatic]
    static private Pen pen = null;

    private Point prevPoint;

    protected override void OnStylusDown(RawStylusInput rawStylusInput)
    {
        // Allocate memory to store the previous point to draw from.
        prevPoint = new Point(double.NegativeInfinity, double.NegativeInfinity);
        base.OnStylusDown(rawStylusInput);
    }

    protected override void OnDraw(DrawingContext drawingContext,
                                   StylusPointCollection stylusPoints,
                                   Geometry geometry, Brush fillBrush)
    {
        // Create a new Brush, if necessary.
        brush ??= new LinearGradientBrush(Colors.Red, Colors.Blue, 20d);

        // Create a new Pen, if necessary.
        pen ??= new Pen(brush, 2d);

        // Draw linear gradient ellipses between
        // all the StylusPoints that have come in.
        for (int i = 0; i < stylusPoints.Count; i++)
        {
            Point pt = (Point)stylusPoints[i];
            Vector v = Point.Subtract(prevPoint, pt);

            // Only draw if we are at least 4 units away
            // from the end of the last ellipse. Otherwise,
            // we're just redrawing and wasting cycles.
            if (v.Length > 4)
            {
                // Set the thickness of the stroke based
                // on how hard the user pressed.
                double radius = stylusPoints[i].PressureFactor * 10d;
                drawingContext.DrawEllipse(brush, pen, pt, radius, radius);
                prevPoint = pt;
            }
        }
    }
}

```

```

' A StylusPlugin that renders ink with a linear gradient brush effect.
Class CustomDynamicRenderer
    Inherits DynamicRenderer
    <ThreadStatic()> _
    Private Shared brush As Brush = Nothing

    <ThreadStatic()> _
    Private Shared pen As Pen = Nothing

    Private prevPoint As Point

Protected Overrides Sub OnStylusDown(ByVal rawStylusInput As RawStylusInput)
    ' Allocate memory to store the previous point to draw from.
    prevPoint = New Point(Double.NegativeInfinity, Double.NegativeInfinity)
    MyBase.OnStylusDown(rawStylusInput)

End Sub

Protected Overrides Sub OnDraw(ByVal drawingContext As DrawingContext, _
                             ByVal stylusPoints As StylusPointCollection, _
                             ByVal geometry As Geometry, _
                             ByVal fillBrush As Brush)

    ' Create a new Brush, if necessary.
    If brush Is Nothing Then
        brush = New LinearGradientBrush(Colors.Red, Colors.Blue, 20.0)
    End If

    ' Create a new Pen, if necessary.
    If pen Is Nothing Then
        pen = New Pen(brush, 2.0)
    End If

    ' Draw linear gradient ellipses between
    ' all the StylusPoints that have come in.
    Dim i As Integer
    For i = 0 To stylusPoints.Count - 1

        Dim pt As Point = CType(stylusPoints(i), Point)
        Dim v As Vector = Point.Subtract(prevPoint, pt)

        ' Only draw if we are at least 4 units away
        ' from the end of the last ellipse. Otherwise,
        ' we're just redrawing and wasting cycles.
        If v.Length > 4 Then
            ' Set the thickness of the stroke based
            ' on how hard the user pressed.
            Dim radius As Double = stylusPoints(i).PressureFactor * 10.0
            drawingContext.DrawEllipse(brush, pen, pt, radius, radius)
            prevPoint = pt
        End If
    Next i

End Sub
End Class

```

Implementazione di tratti personalizzati

Implementare una classe che derivi da [Stroke](#). Questa classe è responsabile del rendering [StylusPoint](#) dati dopo che è stato convertito in un [Stroke](#) oggetto. Eseguire l'override di [DrawCore](#) classe per eseguire il disegno effettivo.

Nella classe [Stroke](#) può anche archiviare dati personalizzati usando il [AddPropertyData](#) (metodo). Questi dati vengono archiviati con i dati del tratto quando sono resi persistenti.

Il [Stroke](#) classe può anche eseguire l'hit testing. È anche possibile implementare il proprio algoritmo di hit testing eseguendo l'override di [HitTest](#) metodo nella classe corrente.

Il seguente C# codice illustra una classe personalizzata [Stroke](#) classe che esegue il rendering [StylusPoint](#) dati come tratto 3D.

```
using System;
using System.Windows.Media;
using System.Windows;
using System.Windows.Input.StylusPlugIns;
using System.Windows.Input;
using System.Windows.Ink;
```

```
Imports System.Windows.Media
Imports System.Windows
Imports System.Windows.Input.StylusPlugIns
Imports System.Windows.Input
Imports System.Windows.Ink
```

```
// A class for rendering custom strokes
class CustomStroke : Stroke
{
    Brush brush;
    Pen pen;

    public CustomStroke(StylusPointCollection stylusPoints)
        : base(stylusPoints)
    {
        // Create the Brush and Pen used for drawing.
        brush = new LinearGradientBrush(Colors.Red, Colors.Blue, 20d);
        pen = new Pen(brush, 2d);
    }

    protected override void DrawCore(DrawingContext drawingContext,
                                    DrawingAttributes drawingAttributes)
    {
        // Allocate memory to store the previous point to draw from.
        Point prevPoint = new Point(double.NegativeInfinity,
                                    double.NegativeInfinity);

        // Draw linear gradient ellipses between
        // all the StylusPoints in the Stroke.
        for (int i = 0; i < this.StylusPoints.Count; i++)
        {
            Point pt = (Point)this.StylusPoints[i];
            Vector v = Point.Subtract(prevPoint, pt);

            // Only draw if we are at least 4 units away
            // from the end of the last ellipse. Otherwise,
            // we're just redrawing and wasting cycles.
            if (v.Length > 4)
            {
                // Set the thickness of the stroke
                // based on how hard the user pressed.
                double radius = this.StylusPoints[i].PressureFactor * 10d;
                drawingContext.DrawEllipse(brush, pen, pt, radius, radius);
                prevPoint = pt;
            }
        }
    }
}
```

```

' A class for rendering custom strokes
Class CustomStroke
    Inherits Stroke
    Private brush As Brush
    Private pen As Pen

    Public Sub New(ByVal stylusPoints As StylusPointCollection)
        MyBase.New(stylusPoints)
        ' Create the Brush and Pen used for drawing.
        brush = New LinearGradientBrush(Colors.Red, Colors.Blue, 20.0)
        pen = New Pen(brush, 2.0)

    End Sub

    Protected Overrides Sub DrawCore(ByVal drawingContext As DrawingContext, _
                                    ByVal drawingAttributes As DrawingAttributes)

        ' Allocate memory to store the previous point to draw from.
        Dim prevPoint As New Point(Double.NegativeInfinity, Double.NegativeInfinity)

        ' Draw linear gradient ellipses between
        ' all the StylusPoints in the Stroke.
        Dim i As Integer
        For i = 0 To Me.StylusPoints.Count - 1
            Dim pt As Point = CType(Me.StylusPoints(i), Point)
            Dim v As Vector = Point.Subtract(prevPoint, pt)

            ' Only draw if we are at least 4 units away
            ' from the end of the last ellipse. Otherwise,
            ' we're just redrawing and wasting cycles.
            If v.Length > 4 Then
                ' Set the thickness of the stroke
                ' based on how hard the user pressed.
                Dim radius As Double = Me.StylusPoints(i).PressureFactor * 10.0
                drawingContext.DrawEllipse(brush, pen, pt, radius, radius)
                prevPoint = pt
            End If
        Next i

    End Sub
End Class

```

Implementazione di un oggetto InkCanvas personalizzato

Il modo più semplice per usare l'oggetto personalizzato [DynamicRenderer](#) tratto e implementare una classe che deriva da [InkCanvas](#) e Usa queste classi. Il [InkCanvas](#) ha un [DynamicRenderer](#) proprietà che specifica la modalità di rendering del tratto mentre l'utente lo disegna.

Su custom rendering tratti su un [InkCanvas](#) eseguire le operazioni seguenti:

- Creare una classe che deriva dal [InkCanvas](#).
- Assegnare l'oggetto personalizzato [DynamicRenderer](#) per il [InkCanvas.DynamicRenderer](#) proprietà.
- Eseguire l'override del metodo [OnStrokeCollected](#). In questo metodo, rimuovere il tratto originale aggiunto all'oggetto [InkCanvas](#). Creare quindi un tratto personalizzato, aggiungerlo al [Strokes](#) proprietà e chiamare la classe di base con un nuovo [InkCanvasStrokeCollectedEventArgs](#) che contiene il tratto personalizzato.

Il seguente C# codice illustra una classe personalizzata [InkCanvas](#) classe che utilizza un oggetto personalizzato [DynamicRenderer](#) e raccoglie tratti personalizzati.

```

public class CustomRenderingInkCanvas : InkCanvas
{
    CustomDynamicRenderer customRenderer = new CustomDynamicRenderer();

    public CustomRenderingInkCanvas() : base()
    {
        // Use the custom dynamic renderer on the
        // custom InkCanvas.
        this.DynamicRenderer = customRenderer;
    }

    protected override void OnStrokeCollected(InkCanvasStrokeCollectedEventArgs e)
    {
        // Remove the original stroke and add a custom stroke.
        this.Strokes.Remove(e.Stroke);
        CustomStroke customStroke = new CustomStroke(e.Stroke.StylusPoints);
        this.Strokes.Add(customStroke);

        // Pass the custom stroke to base class' OnStrokeCollected method.
        InkCanvasStrokeCollectedEventArgs args =
            new InkCanvasStrokeCollectedEventArgs(customStroke);
        base.OnStrokeCollected(args);
    }
}

```

Un' [InkCanvas](#) può avere più di uno [DynamicRenderer](#). È possibile aggiungere più [DynamicRenderer](#) gli oggetti per il [InkCanvas](#) aggiungendole al [StylusPlugIns](#) proprietà.

Conclusione

È possibile personalizzare l'aspetto dell'input penna derivando [DynamicRenderer](#), [Stroke](#), e [InkCanvas](#) classi. Insieme, queste classi assicurano che il tratto abbia un aspetto coerente quando l'utente lo disegna e dopo che viene raccolto.

Vedere anche

- [Gestione avanzata dell'input penna](#)

Intercettazione dell'input dello stilo

10/02/2020 • 7 minutes to read • [Edit Online](#)

L'architettura [System.Windows.Input.StylusPlugIns](#) fornisce un meccanismo per l'implementazione del controllo di basso livello sull'input [Stylus](#) e la creazione di oggetti [Stroke](#) di input digitali. La classe [StylusPlugIn](#) fornisce un meccanismo per implementare il comportamento personalizzato e applicarlo al flusso di dati provenienti dal dispositivo stilo per ottenere prestazioni ottimali.

In questo argomento sono contenute le seguenti sottosezioni:

- [Architettura](#)
- [Implementazione di plug-in dello stilo](#)
- [Aggiunta del plug-in a un InkCanvas](#)
- [Conclusioni](#)

Architecture

Il [StylusPlugIn](#) è l'evoluzione delle API di [StylusInput](#), descritte in [accesso e manipolazione dell'input penna](#).

Ogni [UIElement](#) dispone di una proprietà [StylusPlugIns](#) che è un [StylusPlugInCollection](#). È possibile aggiungere una [StylusPlugIn](#) alla proprietà [StylusPlugIns](#) di un elemento per modificare i dati [StylusPoint](#) durante la generazione. i dati [StylusPoint](#) sono costituiti da tutte le proprietà supportate dal digitalizzatore di sistema, inclusi i dati di [X](#) e [Y](#) punto, nonché i dati [PressureFactor](#).

Gli oggetti [StylusPlugIn](#) vengono inseriti direttamente nel flusso di dati provenienti dal dispositivo [Stylus](#) quando si aggiunge il [StylusPlugIn](#) alla proprietà [StylusPlugIns](#). L'ordine in cui i plug-in vengono aggiunti alla raccolta [StylusPlugIns](#) determina l'ordine in cui riceveranno i dati [StylusPoint](#). Se ad esempio si aggiunge un plug-in di filtro che limita l'input a una determinata area, quindi si aggiunge un plug-in che riconosce i movimenti quando vengono scritti, il plug-in che riconosce i movimenti riceverà dati filtrati [StylusPoint](#).

Implementazione di plug-in dello stilo

Per implementare un plug-in, derivare una classe da [StylusPlugIn](#). Questa classe viene applicata o il flusso di dati in entrata dal [Stylus](#). In questa classe è possibile modificare i valori dei dati [StylusPoint](#).

Caution

Se un [StylusPlugIn](#) genera o causa un'eccezione, l'applicazione verrà chiusa. È necessario verificare accuratamente i controlli che utilizzano un [StylusPlugIn](#) e utilizzare un controllo solo se si è certi che l'[StylusPlugIn](#) non genererà un'eccezione.

Nell'esempio riportato di seguito viene illustrato un plug-in che limita l'input dello stilo modificando il [X](#) e [Y](#) i valori nei dati del [StylusPoint](#) così come provengono dal dispositivo [Stylus](#).

```
using System;
using System.Windows.Media;
using System.Windows;
using System.Windows.Input.StylusPlugIns;
using System.Windows.Input;
using System.Windows.Ink;
```

```
Imports System.Windows.Media
Imports System.Windows
Imports System.Windows.Input.StylusPlugIns
Imports System.Windows.Input
Imports System.Windows.Ink
```

```
// A StylusPlugin that restricts the input area.
class FilterPlugin : StylusPlugIn
{
    protected override void OnStylusDown(RawStylusInput rawStylusInput)
    {
        // Call the base class before modifying the data.
        base.OnStylusDown(rawStylusInput);

        // Restrict the stylus input.
        Filter(rawStylusInput);
    }

    protected override void OnStylusMove(RawStylusInput rawStylusInput)
    {
        // Call the base class before modifying the data.
        base.OnStylusMove(rawStylusInput);

        // Restrict the stylus input.
        Filter(rawStylusInput);
    }

    protected override void OnStylusUp(RawStylusInput rawStylusInput)
    {
        // Call the base class before modifying the data.
        base.OnStylusUp(rawStylusInput);

        // Restrict the stylus input
        Filter(rawStylusInput);
    }

    private void Filter(RawStylusInput rawStylusInput)
    {
        // Get the StylusPoints that have come in.
        StylusPointCollection stylusPoints = rawStylusInput.GetStylusPoints();

        // Modify the (X,Y) data to move the points
        // inside the acceptable input area, if necessary.
        for (int i = 0; i < stylusPoints.Count; i++)
        {
            StylusPoint sp = stylusPoints[i];
            if (sp.X < 50) sp.X = 50;
            if (sp.X > 250) sp.X = 250;
            if (sp.Y < 50) sp.Y = 50;
            if (sp.Y > 250) sp.Y = 250;
            stylusPoints[i] = sp;
        }

        // Copy the modified StylusPoints back to the RawStylusInput.
        rawStylusInput.SetStylusPoints(stylusPoints);
    }
}
```

```

' A StylusPlugin that restricts the input area.
Class FilterPlugin
    Inherits StylusPlugIn

    Protected Overrides Sub OnStylusDown(ByVal rawStylusInput As RawStylusInput)
        ' Call the base class before modifying the data.
        MyBase.OnStylusDown(rawStylusInput)

        ' Restrict the stylus input.
        Filter(rawStylusInput)

    End Sub

    Protected Overrides Sub OnStylusMove(ByVal rawStylusInput As RawStylusInput)
        ' Call the base class before modifying the data.
        MyBase.OnStylusMove(rawStylusInput)

        ' Restrict the stylus input.
        Filter(rawStylusInput)

    End Sub

    Protected Overrides Sub OnStylusUp(ByVal rawStylusInput As RawStylusInput)
        ' Call the base class before modifying the data.
        MyBase.OnStylusUp(rawStylusInput)

        ' Restrict the stylus input
        Filter(rawStylusInput)

    End Sub

    Private Sub Filter(ByVal rawStylusInput As RawStylusInput)
        ' Get the StylusPoints that have come in.
        Dim stylusPoints As StylusPointCollection = rawStylusInput.GetStylusPoints()

        ' Modify the (X,Y) data to move the points
        ' inside the acceptable input area, if necessary.
        Dim i As Integer
        For i = 0 To stylusPoints.Count - 1
            Dim sp As StylusPoint = stylusPoints(i)
            If sp.X < 50 Then
                sp.X = 50
            End If
            If sp.X > 250 Then
                sp.X = 250
            End If
            If sp.Y < 50 Then
                sp.Y = 50
            End If
            If sp.Y > 250 Then
                sp.Y = 250
            End If
            stylusPoints(i) = sp
        Next i

        ' Copy the modified StylusPoints back to the RawStylusInput.
        rawStylusInput.SetStylusPoints(stylusPoints)

    End Sub
End Class

```

Aggiunta del plug-in a un InkCanvas

Il modo più semplice per usare il plug-in personalizzato consiste nell'implementare una classe che deriva da `InkCanvas` e aggiungerla alla proprietà `StylusPlugIns`.

Nell'esempio seguente viene illustrata una `InkCanvas` personalizzata che filtra l'input penna.

```
public class FilterInkCanvas : InkCanvas
{
    FilterPlugin filter = new FilterPlugin();

    public FilterInkCanvas()
        : base()
    {
        this.StylusPlugIns.Add(filter);
    }
}
```

Se si aggiunge un `FilterInkCanvas` all'applicazione e lo si esegue, si noterà che l'input penna non è limitato a un'area fino a quando l'utente non completa un tratto. Questo perché il `InkCanvas` dispone di una proprietà `DynamicRenderer`, che è un `StylusPlugIn` ed è già un membro della raccolta `StylusPlugIns`. Il `StylusPlugIn` personalizzato aggiunto alla raccolta di `StylusPlugIns` riceve i dati `StylusPoint` dopo che `DynamicRenderer` riceve i dati. Di conseguenza, i dati `StylusPoint` non verranno filtrati fino a quando l'utente non solleva la penna per terminare un tratto. Per filtrare l'input penna quando l'utente lo disegna, è necessario inserire il `FilterPlugin` prima della `DynamicRenderer`.

Nel codice C# seguente viene illustrato un `InkCanvas` personalizzato che filtra l'input penna mentre viene disegnato.

```
public class DynamicallyFilteredInkCanvas : InkCanvas
{
    FilterPlugin filter = new FilterPlugin();

    public DynamicallyFilteredInkCanvas()
        : base()
    {
        int dynamicRenderIndex =
            this.StylusPlugIns.IndexOf(this.DynamicRenderer);

        this.StylusPlugIns.Insert(dynamicRenderIndex, filter);
    }
}
```

Conclusioni

Derivando le proprie classi di `StylusPlugIn` e inserendole in raccolte di `StylusPlugInCollection`, è possibile migliorare significativamente il comportamento dell'input penna digitale. Si ha accesso ai dati `StylusPoint` durante la generazione, offrendo la possibilità di personalizzare l'input di `Stylus`. Poiché si dispone di un accesso di basso livello ai dati `StylusPoint`, è possibile implementare la raccolta di input penna e il rendering con prestazioni ottimali per l'applicazione.

Vedere anche

- [Gestione avanzata dell'input penna](#)
- [Accesso e modifica dell'input penna](#)

Creazione di un controllo di input penna

10/02/2020 • 10 minutes to read • [Edit Online](#)

È possibile creare un controllo personalizzato che esegue il rendering dell'input penna in modo dinamico e statico. Ovvero il rendering dell'input penna quando un utente disegna un tratto, causando la visualizzazione dell'input penna in "Flow" dalla penna del tablet e la visualizzazione dell'input penna dopo l'aggiunta al controllo, tramite la penna del tablet, incollato dagli Appunti o caricato da un file. Per eseguire dinamicamente il rendering dell'input penna, il controllo deve usare un [DynamicRenderer](#). Per eseguire il rendering statico dell'input penna, è necessario eseguire l'override dei metodi di evento stilo ([OnStylusDown](#), [OnStylusMovee](#) [OnStylusUp](#)) per raccogliere dati di [StylusPoint](#), creare tratti e aggiungerli a un [InkPresenter](#) (che esegue il rendering dell'input penna sul controllo).

In questo argomento sono contenute le seguenti sottosezioni:

- [Procedura: raccogliere dati dei punti dello stilo e creare tratti di input penna](#)
- [Procedura: abilitare il controllo per accettare l'input dal mouse](#)
- [Riunendoli](#)
- [Uso di plug-in aggiuntivi e DynamicRenderers](#)
- [Conclusioni](#)

Procedura: raccogliere dati dei punti dello stilo e creare tratti di input penna

Per creare un controllo che raccoglie e gestisce i tratti di input penna, effettuare le operazioni seguenti:

1. Derivare una classe da [Control](#) o da una delle classi derivate da [Control](#), ad esempio [Label](#).

```
using System;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Input.StylusPlugIns;
using System.Windows.Controls;
using System.Windows;
```

```
class InkControl : Label
{
```

```
}
```

2. Aggiungere un [InkPresenter](#) alla classe e impostare la proprietà [Content](#) sul nuovo [InkPresenter](#).

```

InkPresenter ip;

public InkControl()
{
    // Add an InkPresenter for drawing.
    ip = new InkPresenter();
    this.Content = ip;
}

```

3. Allineare il [RootVisual](#) della [DynamicRenderer](#) al [InkPresenter](#) chiamando il metodo [AttachVisuals](#) e aggiungere il [DynamicRenderer](#) alla raccolta di [StylusPlugIns](#). Ciò consente all'[InkPresenter](#) di visualizzare l'input penna quando i dati del punto dello stilo vengono raccolti dal controllo.

```

public InkControl()
{

    // Add a dynamic renderer that
    // draws ink as it "flows" from the stylus.
    dr = new DynamicRenderer();
    ip.AttachVisuals(dr.RootVisual, dr.DrawingAttributes);
    this.StylusPlugIns.Add(dr);
}

```

4. Eseguire l'override del metodo [OnStylusDown](#) . In questo metodo acquisire lo stilo con una chiamata a [Capture](#). Acquisendo lo stilo, il controllo continuerà a ricevere [StylusMove](#) e [StylusUp](#) eventi anche se lo stilo lascia i limiti del controllo. Questa operazione non è strettamente obbligatoria, ma è quasi sempre necessaria per un'esperienza utente ottimale. Creare una nuova [StylusPointCollection](#) per raccogliere i dati [StylusPoint](#). Infine, aggiungere il set iniziale di dati di [StylusPoint](#) al [StylusPointCollection](#).

```

protected override void OnStylusDown(StylusDownEventArgs e)
{
    // Capture the stylus so all stylus input is routed to this control.
    Stylus.Capture(this);

    // Allocate memory for the StylusPointsCollection and
    // add the StylusPoints that have come in so far.
    stylusPoints = new StylusPointCollection();
    StylusPointCollection eventPoints =
        e.GetStylusPoints(this, stylusPoints.Description);

    stylusPoints.Add(eventPoints);
}

```

5. Eseguire l'override del metodo [OnStylusMove](#) e aggiungere i dati [StylusPoint](#) all'oggetto [StylusPointCollection](#) creato in precedenza.

```

protected override void OnStylusMove(StylusEventArgs e)
{
    if (stylusPoints == null)
    {
        return;
    }

    // Add the StylusPoints that have come in since the
    // last call to OnStylusMove.
    StylusPointCollection newStylusPoints =
        e.GetStylusPoints(this, stylusPoints.Description);
    stylusPoints.Add(newStylusPoints);
}

```

- Eseguire l'override del metodo [OnStylusUp](#) e creare una nuova [Stroke](#) con i dati di [StylusPointCollection](#). Aggiungere il nuovo [Stroke](#) creato alla raccolta [Strokes](#) della [InkPresenter](#) e acquisire lo stilo di rilascio.

```

protected override void OnStylusUp(StylusEventArgs e)
{
    if (stylusPoints == null)
    {
        return;
    }

    // Add the StylusPoints that have come in since the
    // last call to OnStylusMove.
    StylusPointCollection newStylusPoints =
        e.GetStylusPoints(this, stylusPoints.Description);
    stylusPoints.Add(newStylusPoints);

    // Create a new stroke from all the StylusPoints since OnStylusDown.
    Stroke stroke = new Stroke(stylusPoints);

    // Add the new stroke to the Strokes collection of the InkPresenter.
    ip.Strokes.Add(stroke);

    // Clear the StylusPointsCollection.
    stylusPoints = null;

    // Release stylus capture.
    Stylus.Capture(null);
}

```

Procedura: abilitare il controllo per accettare l'input dal mouse

Se si aggiunge il controllo precedente all'applicazione, lo si esegue e si usa il mouse come dispositivo di input, si noterà che i tratti non sono salvati in modo permanente. Per salvare in modo permanente i tratti quando il mouse viene usato come dispositivo di input, eseguire le operazioni seguenti:

- Eseguire l'override del [OnMouseLeftButtonDown](#) e creare una nuova [StylusPointCollection](#) ottenere la posizione del mouse quando si è verificato l'evento e creare un [StylusPoint](#) usando i dati del punto e aggiungere il [StylusPoint](#) alla [StylusPointCollection](#).

```

protected override void OnMouseLeftButtonDown(MouseEventArgs e)
{
    base.OnMouseLeftButtonDown(e);

    // If a stylus generated this event, return.
    if (e.StylusDevice != null)
    {
        return;
    }

    // Start collecting the points.
    stylusPoints = new StylusPointCollection();
    Point pt = e.GetPosition(this);
    stylusPoints.Add(new StylusPoint(pt.X, pt.Y));
}

```

- Eseguire l'override del metodo [OnMouseMove](#) . Ottiene la posizione del mouse quando si è verificato l'evento e crea un [StylusPoint](#) usando i dati del punto. Aggiungere la [StylusPoint](#) all'oggetto [StylusPointCollection](#) creato in precedenza.

```

protected override void OnMouseMove(MouseEventArgs e)
{
    base.OnMouseMove(e);

    // If a stylus generated this event, return.
    if (e.StylusDevice != null)
    {
        return;
    }

    // Don't collect points unless the left mouse button
    // is down.
    if (e.LeftButton == MouseButtonState.Released ||
        stylusPoints == null)
    {
        return;
    }

    Point pt = e.GetPosition(this);
    stylusPoints.Add(new StylusPoint(pt.X, pt.Y));
}

```

- Eseguire l'override del metodo [OnMouseLeftButtonUp](#) . Creare una nuova [Stroke](#) con i dati [StylusPointCollection](#) e aggiungere il nuovo [Stroke](#) creato alla raccolta [Strokes](#) del [InkPresenter](#).

```

protected override void OnMouseLeftButtonUp(MouseEventArgs e)
{
    base.OnMouseLeftButtonUp(e);

    // If a stylus generated this event, return.
    if (e.StylusDevice != null)
    {
        return;
    }

    if (stylusPoints == null)
    {
        return;
    }

    Point pt = e.GetPosition(this);
    stylusPoints.Add(new StylusPoint(pt.X, pt.Y));

    // Create a stroke and add it to the InkPresenter.
    Stroke stroke = new Stroke(stylusPoints);
    stroke.DrawingAttributes = dr.DrawingAttributes;
    ip.Strokes.Add(stroke);

    stylusPoints = null;
}

```

Riunendoli

L'esempio seguente è un controllo personalizzato che raccoglie l'input penna quando l'utente usa il mouse o la penna.

```

using System;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Input.StylusPlugIns;
using System.Windows.Controls;
using System.Windows;

// A control for managing ink input
class InkControl : Label
{
    InkPresenter ip;
    DynamicRenderer dr;

    // The StylusPointsCollection that gathers points
    // before Stroke from is created.
    StylusPointCollection stylusPoints = null;

    public InkControl()
    {
        // Add an InkPresenter for drawing.
        ip = new InkPresenter();
        this.Content = ip;

        // Add a dynamic renderer that
        // draws ink as it "flows" from the stylus.
        dr = new DynamicRenderer();
        ip.AttachVisuals(dr.RootVisual, dr.DrawingAttributes);
        this.StylusPlugIns.Add(dr);
    }

    static InkControl()

```

```

{
    // Allow ink to be drawn only within the bounds of the control.
    Type owner = typeof(InkControl);
    ClipToBoundsProperty.OverrideMetadata(owner,
        new FrameworkPropertyMetadata(true));
}

protected override void OnStylusDown(StylusEventArgs e)
{
    // Capture the stylus so all stylus input is routed to this control.
    Stylus.Capture(this);

    // Allocate memory for the StylusPointsCollection and
    // add the StylusPoints that have come in so far.
    stylusPoints = new StylusPointCollection();
    StylusPointCollection eventPoints =
        e.GetStylusPoints(this, stylusPoints.Description);

    stylusPoints.Add(eventPoints);
}

protected override void OnStylusMove(StylusEventArgs e)
{
    if (stylusPoints == null)
    {
        return;
    }

    // Add the StylusPoints that have come in since the
    // last call to OnStylusMove.
    StylusPointCollection newStylusPoints =
        e.GetStylusPoints(this, stylusPoints.Description);
    stylusPoints.Add(newStylusPoints);
}

protected override void OnStylusUp(StylusEventArgs e)
{
    if (stylusPoints == null)
    {
        return;
    }

    // Add the StylusPoints that have come in since the
    // last call to OnStylusMove.
    StylusPointCollection newStylusPoints =
        e.GetStylusPoints(this, stylusPoints.Description);
    stylusPoints.Add(newStylusPoints);

    // Create a new stroke from all the StylusPoints since OnStylusDown.
    Stroke stroke = new Stroke(stylusPoints);

    // Add the new stroke to the Strokes collection of the InkPresenter.
    ip.Strokes.Add(stroke);

    // Clear the StylusPointsCollection.
    stylusPoints = null;

    // Release stylus capture.
    Stylus.Capture(null);
}

protected override void OnMouseLeftButtonDown(MouseEventArgs e)
{
    base.OnMouseLeftButtonDown(e);

    // If a stylus generated this event, return.
    if (e.StylusDevice != null)
    {

```

```

        return;
    }

    // Start collecting the points.
    stylusPoints = new StylusPointCollection();
    Point pt = e.GetPosition(this);
    stylusPoints.Add(new StylusPoint(pt.X, pt.Y));
}

protected override void OnMouseMove(MouseEventArgs e)
{
    base.OnMouseMove(e);

    // If a stylus generated this event, return.
    if (e.StylusDevice != null)
    {
        return;
    }

    // Don't collect points unless the left mouse button
    // is down.
    if (e.LeftButton == MouseButtonState.Released ||
        stylusPoints == null)
    {
        return;
    }

    Point pt = e.GetPosition(this);
    stylusPoints.Add(new StylusPoint(pt.X, pt.Y));
}

protected override void OnMouseLeftButtonUp(MouseButtonEventArgs e)
{
    base.OnMouseLeftButtonUp(e);

    // If a stylus generated this event, return.
    if (e.StylusDevice != null)
    {
        return;
    }

    if (stylusPoints == null)
    {
        return;
    }

    Point pt = e.GetPosition(this);
    stylusPoints.Add(new StylusPoint(pt.X, pt.Y));

    // Create a stroke and add it to the InkPresenter.
    Stroke stroke = new Stroke(stylusPoints);
    stroke.DrawingAttributes = dr.DrawingAttributes;
    ip.Strokes.Add(stroke);

    stylusPoints = null;
}
}

```

Uso di plug-in aggiuntivi e DynamicRenderers

Analogamente a InkCanvas, il controllo personalizzato può disporre di [StylusPlugIn](#) personalizzati e oggetti [DynamicRenderer](#) aggiuntivi. Aggiungerli alla raccolta di [StylusPlugins](#). L'ordine degli oggetti [StylusPlugIn](#) nel [StylusPlugInCollection](#) influiscono sull'aspetto dell'input penna quando ne viene eseguito il rendering. Si

supponga di avere un [DynamicRenderer](#) denominato `dynamicRenderer` e un [StylusPlugIn](#) personalizzato denominato `translatePlugin` che compensa l'input penna dalla penna del tablet. Se `translatePlugin` è il primo [StylusPlugIn](#) nel [StylusPlugInCollection](#) `dynamicRenderer` è il secondo, l'input penna "Flows" verrà sfalsato quando l'utente sposta la penna. Se `dynamicRenderer` è il primo e `translatePlugin` è il secondo, l'input penna non verrà sfalsato fino a quando l'utente non solleva la penna.

Conclusioni

È possibile creare un controllo che raccoglie ed esegue il rendering dell'input penna eseguendo l'override dei metodi di evento dello stilo. Creando un controllo personalizzato, derivando le proprie classi di [StylusPlugIn](#) e inserendole in [StylusPlugInCollection](#), è possibile implementare praticamente qualsiasi comportamento immaginabile con l'input penna digitale. È possibile accedere ai dati [StylusPoint](#) durante la generazione, offrendo la possibilità di personalizzare [Stylus](#) input e di eseguirne il rendering sullo schermo in base alle esigenze dell'applicazione. Poiché si dispone di un accesso di basso livello ai dati [StylusPoint](#), è possibile implementare la raccolta di input penna ed eseguirne il rendering con prestazioni ottimali per l'applicazione.

Vedere anche

- [Gestione avanzata dell'input penna](#)
- [Accesso e modifica dell'input penna](#)

Modello di threading dell'input penna

23/10/2019 • 8 minutes to read • [Edit Online](#)

Uno dei vantaggi dell'input penna su un Tablet PC è che presenta notevoli analogie con la scrittura con una carta e penna regolare. A tale scopo, la penna del tablet PC raccoglie i dati di input con una frequenza molto più elevata rispetto a un mouse ed esegue il rendering dell'input penna mentre l'utente scrive. Thread dell'interfaccia utente dell'applicazione non è sufficiente per la raccolta dei dati della penna e rendering di input penna, perché può essere bloccato. Per risolvere il problema, un WPF applicazione utilizza due thread aggiuntivi quando l'utente scrive tramite input penna.

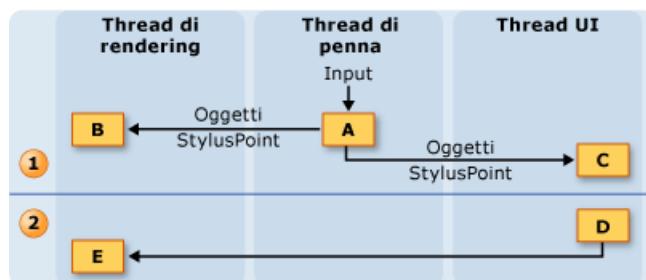
L'elenco seguente descrive il thread che partecipano nella raccolta e il rendering dell'input penna digitale:

- Thread di penna - il thread che accetta l'input dallo stilo. (In realtà, si tratta di un pool di thread, ma in questo argomento fa riferimento a esso come un thread di penna).
- Thread interfaccia utente dell'applicazione - il thread che controlla l'interfaccia utente dell'applicazione.
- Thread di rendering dinamico - il thread che esegue il rendering dell'input penna mentre l'utente consente di disegnare un tratto. Il thread di rendering dinamico è diverso rispetto al thread che esegue il rendering di altri elementi dell'interfaccia utente per l'applicazione, come indicato in Windows Presentation Foundation [modello di Threading](#).

Il modello di input penna non cambiano se l'applicazione usa la [InkCanvas](#) o un controllo personalizzato simile a quello riportato nella [creazione di un controllo Input penna](#). Anche se il threading in termini di questo argomento viene descritto il [InkCanvas](#), gli stessi concetti si applicano quando si crea un controllo personalizzato.

Cenni preliminari sul threading

Quando un utente consente di disegnare un tratto, il diagramma seguente illustra il modello di threading:



1. Azioni che si verificano mentre l'utente lo disegna

- Quando l'utente consente di disegnare un tratto, i punti dello stilo sono disponibili in sul thread di penna. Plug-in dello stilo, tra cui il [DynamicRenderer](#), accettare i punti dello stilo sul thread di penna e ha la possibilità di modificarli prima di [InkCanvas](#) li riceve.
- Il [DynamicRenderer](#) esegue il rendering dei punti dello stilo nel thread di rendering dinamico. Ciò si verifica nello stesso momento del passaggio precedente.
- Il [InkCanvas](#) riceve i punti dello stilo sul thread UI.

2. Azioni che si verificano dopo che l'utente termina il tratto

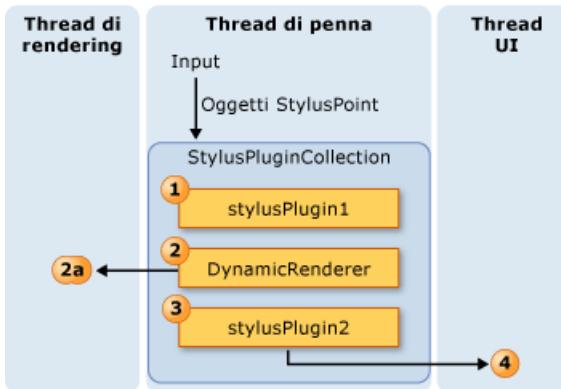
- Quando l'utente completa il disegno del tratto, la [InkCanvas](#) crea un' [Stroke](#) dell'oggetto e lo aggiunge al [InkPresenter](#), che in modo statico ne esegue il rendering.

- b. Gli avvisi di thread dell'interfaccia utente di [DynamicRenderer](#) che in modo statico di rendering del tratto, pertanto la [DynamicRenderer](#) rimuove la rappresentazione visiva del tratto.

Raccolta di input penna e Plug-in dello stilo

Ciascuna [UIElement](#) ha un [StylusPlugInCollection](#). Il [StylusPlugIn](#) oggetti nel [StylusPlugInCollection](#) ricevono e possono modificare i punti dello stilo sul thread di penna. Il [StylusPlugIn](#) gli oggetti ricevono i punti dello stilo in base al relativo ordine nella [StylusPlugInCollection](#).

Il diagramma seguente illustra lo scenario ipotetico in cui il [StylusPlugIns](#) raccolta di un [UIElement](#) contiene [stylusPlugin1](#), una [DynamicRenderer](#), e [stylusPlugin2](#), in questo ordine.



Nel diagramma precedente, il comportamento seguente viene eseguita:

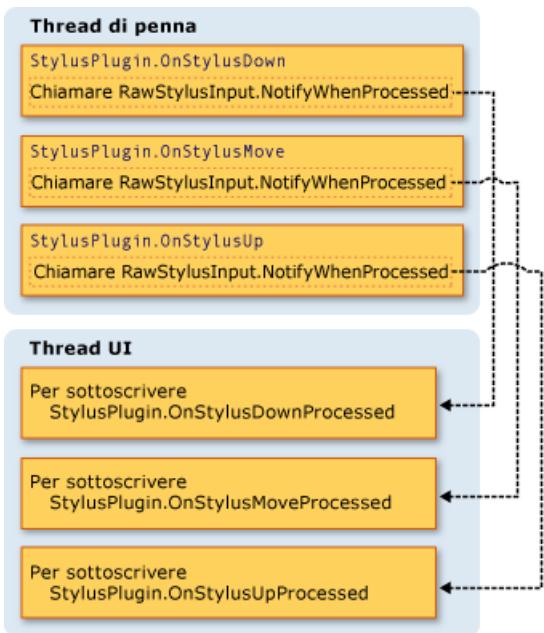
1. [stylusPlugin1](#) Modifica i valori per x e y.
2. [DynamicRenderer](#) riceve i punti dello stilo modificato e ne esegue il rendering nel thread di rendering dinamico.
3. [stylusPlugin2](#) riceve i punti dello stilo modificato e ulteriore modifica i valori per x e y.
4. L'applicazione raccoglie i punti dello stilo e, quando l'utente completa del tratto, in modo statico viene eseguito il rendering del tratto.

Si supponga che [stylusPlugin1](#) limita i punti dello stilo a un rettangolo e [stylusPlugin2](#) converte i punti dello stilo a destra. Nello scenario precedente, il [DynamicRenderer](#) riceve i punti dello stilo limitato, ma non i punti dello stilo tradotti. Quando l'utente lo disegna, entro i limiti del rettangolo di rendering del tratto, ma il tratto non sembra essere tradotto fino a quando l'utente solleva la penna.

Esecuzione di operazioni con uno stilo plug-in sul thread UI

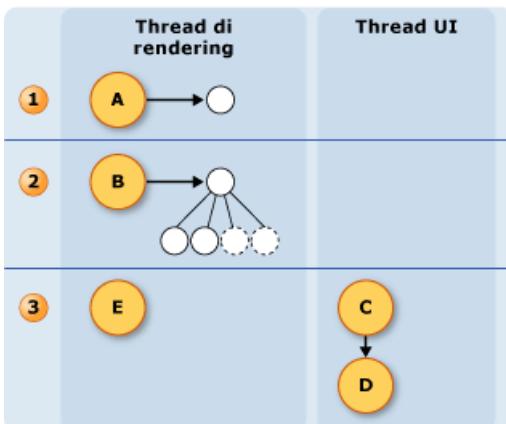
Poiché hit test accurati non possono essere eseguite sul thread di penna, alcuni elementi potrebbero ricevere occasionalmente input dello stilo destinato ad altri elementi. Se è necessario assicurarsi che l'input è stato instradato correttamente prima di eseguire un'operazione, sottoscrivere ed eseguire l'operazione dei [OnStylusDownProcessed](#), [OnStylusMoveProcessed](#), o [OnStylusUpProcessed](#) (metodo). Questi metodi vengono richiamati dal thread dell'applicazione dopo l'esecuzione di hit test accurati. Per eseguire la sottoscrizione a questi metodi, chiamare il [NotifyWhenProcessed](#) metodo nel metodo che si verifica nel thread di penna.

Il diagramma seguente illustra la relazione tra il thread di penna e il thread dell'interfaccia utente per quanto riguarda gli eventi dello stilo di un [StylusPlugIn](#).



Il rendering dell'input penna

Quando l'utente consente di disegnare un tratto, [DynamicRenderer](#) esegue il rendering dell'input penna in un thread separato in modo che sembri "fluire" dalla penna anche quando il thread UI è occupato. Il [DynamicRenderer](#) compila un struttura ad albero visuale nel thread di rendering dinamico che raccoglie i punti dello stilo. Quando l'utente completa il tratto di [DynamicRenderer](#) chiede di ricevere una notifica quando l'applicazione esegue il successivo passaggio di rendering. Dopo l'applicazione ha completato il passaggio successivo per il rendering, il [DynamicRenderer](#) pulisce la struttura ad albero visuale. Il diagramma seguente illustra questo processo.



1. L'utente inizia il tratto.
 - a. Il [DynamicRenderer](#) crea la struttura ad albero visuale.
2. L'utente lo disegna il tratto.
 - a. Il [DynamicRenderer](#) compila la struttura ad albero visuale.
3. L'utente termina il tratto.
 - a. Il [InkPresenter](#) aggiunge il tratto alla relativa struttura ad albero visuale.
 - b. Il livello di integrazione Media (MIL) sottoposto a rendering statico.
 - c. Il [DynamicRenderer](#) pulisce gli oggetti visivi.

Argomenti sulle procedure relative all'input penna

23/10/2019 • 2 minutes to read • [Edit Online](#)

In questa sezione

- [Selezionare l'input penna da un controllo personalizzato](#)
- [Aggiungere dati personalizzati ai dati dell'input penna](#)
- [Cancellare l'input penna da un controllo personalizzato](#)
- [Riconoscere i movimenti dell'applicazione](#)
- [Trascinare l'input penna](#)
- [Eseguire l'associazione dati a un InkCanvas](#)
- [Analizzare l'input penna con i suggerimenti dell'analisi](#)
- [Ruotare l'input penna](#)
- [Disabilitare RealTimeStylus per le applicazioni WPF](#)

Procedura: Selezionare l'input penna da un controllo personalizzato

23/10/2019 • 18 minutes to read • [Edit Online](#)

Aggiungendo un [IncrementalLassoHitTester](#) del controllo personalizzato, è possibile abilitare il controllo in modo che un utente può selezionare l'input penna con lo strumento, analogamente al modo in cui il [InkCanvas](#) seleziona input penna con un lazo.

In questo esempio si presuppone che si ha familiarità con la creazione di un controllo personalizzato compatibile. Per creare un controllo personalizzato che accetta l'input penna, vedere [creazione di un controllo Input penna](#).

Esempio

Quando l'utente lo disegna un lazo il [IncrementalLassoHitTester](#) consente di stimare quali tratti saranno all'interno dei limiti del lazo dopo che l'utente completa del lazo. I tratti che sono determinati all'interno dei confini del lazo possono essere considerati come selezionato. I tratti selezionati possono anche essere deselectati. Se l'utente cambia direzione mentre lo strumento lazo, ad esempio il [IncrementalLassoHitTester](#) possibile deselectare alcune tracce.

Il [IncrementalLassoHitTester](#) genera il [SelectionChanged](#) evento, che consente il controllo personalizzato rispondere mentre l'utente è lo strumento lazo. Ad esempio, è possibile modificare l'aspetto dei tratti come utente selezionati e deselectati.

Gestione della modalità di input penna

È utile per l'utente se lazo viene visualizzato in modo diverso rispetto all'input penna sul controllo. A tale scopo, il controllo personalizzato deve tenere traccia della se l'utente è la scrittura o la selezione di input penna. Il modo più semplice per eseguire questa operazione consiste nel dichiarare un'enumerazione con due valori: una per indicare che l'utente scrive dell'input penna e uno per indicare che l'utente seleziona l'input penna.

```
// Enum that keeps track of whether StrokeCollectionDemo is in ink mode
// or select mode.
public enum InkMode
{
    Ink, Select
}
```

```
' Enum that keeps track of whether StrokeCollectionDemo is in ink mode
' or select mode.
Public Enum InkMode
    Ink
    [Select]
End Enum 'InkMode'
```

Successivamente, aggiungere due [DrawingAttributes](#) alla classe: uno da utilizzare quando l'utente scrive tramite una penna, uno da utilizzare quando l'utente seleziona l'input penna. Nel costruttore, inizializzare il [DrawingAttributes](#) allegando sia [AttributeChanged](#) eventi allo stesso gestore eventi. Impostare quindi le [DrawingAttributes](#) proprietà del [DynamicRenderer](#) all'input penna [DrawingAttributes](#).

```
DrawingAttributes inkDA;  
DrawingAttributes selectDA;
```

```
Private inkDA As DrawingAttributes  
Private selectDA As DrawingAttributes
```

```
// In the constructor.  
// Selection drawing attributes use dark gray ink.  
selectDA = new DrawingAttributes();  
selectDA.Color = Colors.DarkGray;  
  
// Ink drawing attributes use default attributes  
inkDA = new DrawingAttributes();  
inkDA.Width = 5;  
inkDA.Height = 5;  
  
inkDA.AttributeChanged += new PropertyDataChangedEventHandler(DrawingAttributesChanged);  
selectDA.AttributeChanged += new PropertyDataChangedEventHandler(DrawingAttributesChanged);
```

```
' In the constructor.  
' Selection drawing attributes use dark gray ink.  
selectDA = New DrawingAttributes()  
selectDA.Color = Colors.DarkGray  
  
' Ink drawing attributes use default attributes  
inkDA = New DrawingAttributes()  
inkDA.Width = 5  
inkDA.Height = 5  
  
AddHandler inkDA.AttributeChanged, _  
    AddressOf DrawingAttributesChanged  
  
AddHandler selectDA.AttributeChanged, _  
    AddressOf DrawingAttributesChanged
```

Aggiungere una proprietà che espone la modalità di selezione. Quando l'utente modifica la modalità di selezione, impostare il [DrawingAttributes](#) proprietà del [DynamicRenderer](#) appropriato [DrawingAttributes](#) dell'oggetto e quindi ricollegare il [RootVisual](#) proprietà per il [InkPresenter](#).

```

// Property to indicate whether the user is inputting or
// selecting ink.
public InkMode Mode
{
    get
    {
        return mode;
    }

    set
    {
        mode = value;

        // Set the DrawingAttributes of the DynamicRenderer
        if (mode == InkMode.Ink)
        {
            renderer.DrawingAttributes = inkDA;
        }
        else
        {
            renderer.DrawingAttributes = selectDA;
        }

        // Reattach the visual of the DynamicRenderer to the InkPresenter.
        presenter.DetachVisuals(renderer.RootVisual);
        presenter.AttachVisuals(renderer.RootVisual, renderer.DrawingAttributes);
    }
}

```

```

' Property to indicate whether the user is inputting or
' selecting ink.
Public Property Mode() As InkMode
    Get
        Return Mode
    End Get

    Set(ByVal value As InkMode)
        modeState = value

        ' Set the DrawingAttributes of the DynamicRenderer
        If modeState = InkMode.Ink Then
            renderer.DrawingAttributes = inkDA
        Else
            renderer.DrawingAttributes = selectDA
        End If

        ' Reattach the visual of the DynamicRenderer to the InkPresenter.
        presenter.DetachVisuals(renderer.RootVisual)
        presenter.AttachVisuals(renderer.RootVisual, renderer.DrawingAttributes)
    End Set
End Property

```

Esporre il **DrawingAttributes** come proprietà in modo che le applicazioni possono determinare l'aspetto dei tratti input penna e selezione.

```

// Property to allow the user to change the pen's DrawingAttributes.
public DrawingAttributes InkDrawingAttributes
{
    get
    {
        return inkDA;
    }
}

// Property to allow the user to change the Selector's newStroke DrawingAttributes.
public DrawingAttributes SelectDrawingAttributes
{
    get
    {
        return selectDA;
    }
}

```

```

' Property to allow the user to change the pen's DrawingAttributes.
Public ReadOnly Property InkDrawingAttributes() As DrawingAttributes
    Get
        Return inkDA
    End Get
End Property

' Property to allow the user to change the Selector's newStroke DrawingAttributes.
Public ReadOnly Property SelectDrawingAttributes() As DrawingAttributes
    Get
        Return selectDA
    End Get
End Property

```

Quando una proprietà di un [DrawingAttributes](#) oggetto viene modificato, il [RootVisual](#) devono essere ricollegate al [InkPresenter](#). Nel gestore eventi per il [AttributeChanged](#) evento, ricollegare il [RootVisual](#) per il [InkPresenter](#).

```

void DrawingAttributesChanged(object sender, PropertyChangedEventArgs e)
{
    // Reattach the visual of the DynamicRenderer to the InkPresenter
    // whenever the DrawingAttributes change.
    presenter.DetachVisuals(renderer.RootVisual);
    presenter.AttachVisuals(renderer.RootVisual, renderer.DrawingAttributes);
}

```

```

Private Sub DrawingAttributesChanged(ByVal sender As Object, _
                                     ByVal e As PropertyChangedEventArgs)

    ' Reattach the visual of the DynamicRenderer to the InkPresenter
    ' whenever the DrawingAttributes change.
    presenter.DetachVisuals(renderer.RootVisual)
    presenter.AttachVisuals(renderer.RootVisual, _
                           renderer.DrawingAttributes)

End Sub

```

Utilizzo di IncrementalLassoHitTester

Creare e inizializzare un [StrokeCollection](#) che contiene i tratti selezionati.

```
// StylusPointCollection that collects the stylus points from the stylus events.  
StylusPointCollection stylusPoints;
```

```
' StylusPointCollection that collects the stylus points from the stylus events.  
Private stylusPoints As StylusPointCollection
```

Quando l'utente inizia a disegnare un tratto, input penna o il lazo, deselectare tutti i tratti selezionati. Quindi, se l'utente lo disegna un lazo, creare un [IncrementalLassoHitTester](#) chiamando [GetIncrementalLassoHitTester](#), sottoscrivere il [SelectionChanged](#) evento, quindi chiamare [AddPoints](#). Questo codice può essere un metodo separato e chiamata dal [OnStylusDown](#) e [OnMouseDown](#) metodi.

```
private void InitializeHitTester(StylusPointCollection collectedPoints)  
{  
    // Deselect any selected strokes.  
    foreach (Stroke selectedStroke in selectedStrokes)  
    {  
        selectedStroke.DrawingAttributes.Color = inkDA.Color;  
    }  
    selectedStrokes.Clear();  
  
    if (mode == InkMode.Select)  
    {  
        // Remove the previously drawn lasso, if it exists.  
        if (lassoPath != null)  
        {  
            presenter.Strokes.Remove(lassoPath);  
            lassoPath = null;  
        }  
  
        selectionTester =  
            presenter.Strokes.GetIncrementalLassoHitTester(80);  
        selectionTester.SelectionChanged +=  
            new LassoSelectionChangedEventHandler(selectionTester_SelectionChanged);  
        selectionTester.AddPoints(collectedPoints);  
    }  
}
```

```
Private Sub InitializeHitTester(ByVal collectedPoints As StylusPointCollection)  
  
    ' Deselect any selected strokes.  
    Dim selectedStroke As Stroke  
    For Each selectedStroke In selectedStrokes  
        selectedStroke.DrawingAttributes.Color = inkDA.Color  
    Next selectedStroke  
    selectedStrokes.Clear()  
  
    If modeState = InkMode.Select Then  
        ' Remove the previously drawn lasso, if it exists.  
        If Not (lassoPath Is Nothing) Then  
            presenter.Strokes.Remove(lassoPath)  
            lassoPath = Nothing  
        End If  
  
        selectionTester = presenter.Strokes.GetIncrementalLassoHitTester(80)  
        AddHandler selectionTester.SelectionChanged, AddressOf selectionTester_SelectionChanged  
        selectionTester.AddPoints(collectedPoints)  
    End If  
  
End Sub
```

Aggiungere i punti dello stilo dal [IncrementalLassoHitTester](#) mentre l'utente lo disegna lazo. Chiamare il metodo seguente dal [OnStylusMove](#), [OnStylusUp](#), [OnMouseMove](#), e [OnMouseLeftButtonUp](#) metodi.

```
private void AddPointsToHitTester(StylusPointCollection collectedPoints)
{
    if (mode == InkMode.Select &&
        selectionTester != null &&
        selectionTester.IsValid)
    {
        // When the control is selecting strokes, add the
        // stylus packetList to selectionTester.
        selectionTester.AddPoints(collectedPoints);
    }
}
```

```
Private Sub AddPointsToHitTester(ByVal collectedPoints As StylusPointCollection)

If modeState = InkMode.Select AndAlso _
    Not selectionTester Is Nothing AndAlso _
    selectionTester.IsValid Then

    ' When the control is selecting strokes, add the
    ' stylus packetList to selectionTester.
    selectionTester.AddPoints(collectedPoints)
End If

End Sub
```

Gestire il [IncrementalLassoHitTester.SelectionChanged](#) evento a cui rispondere quando l'utente seleziona e deseleziona tratti. Il [LassoSelectionChangedEventArgs](#) classe presenta la [SelectedStrokes](#) e [DeselectedStrokes](#) proprietà che recuperano i tratti che sono state selezionate e deselezionate, rispettivamente.

```
void selectionTester_SelectionChanged(object sender,
    LassoSelectionChangedEventArgs args)
{
    // Change the color of all selected strokes to red.
    foreach (Stroke selectedStroke in args.SelectedStrokes)
    {
        selectedStroke.DrawingAttributes.Color = Colors.Red;
        selectedStrokes.Add(selectedStroke);
    }

    // Change the color of all unselected strokes to
    // their original color.
    foreach (Stroke unselectedStroke in args.DeselectedStrokes)
    {
        unselectedStroke.DrawingAttributes.Color = inkDA.Color;
        selectedStrokes.Remove(unselectedStroke);
    }
}
```

```

Private Sub selectionTester_SelectionChanged(ByVal sender As Object, _
                                         ByVal args As LassoSelectionChangedEventArgs)

    ' Change the color of all selected strokes to red.
    Dim selectedStroke As Stroke
    For Each selectedStroke In args.SelectedStrokes
        selectedStroke.DrawingAttributes.Color = Colors.Red
        selectedStrokes.Add(selectedStroke)
    Next selectedStroke

    ' Change the color of all unselected strokes to
    ' their original color.
    Dim unselectedStroke As Stroke
    For Each unselectedStroke In args.DeselectedStrokes
        unselectedStroke.DrawingAttributes.Color = inkDA.Color
        selectedStrokes.Remove(unselectedStroke)
    Next unselectedStroke

End Sub

```

Quando l'utente termina lo strumento lazo, annullare la sottoscrizione di [SelectionChanged](#) eventi e chiamate [EndHitTesting](#).

```

if (mode == InkMode.Select && lassoPath == null)
{
    // Add the lasso to the InkPresenter and add the packetList
    // to selectionTester.
    lassoPath = newStroke;
    lassoPath.DrawingAttributes = selectDA.Clone();
    presenter.Strokes.Add(lassoPath);
    selectionTester.SelectionChanged -= new LassoSelectionChangedEventHandler
        (selectionTester_SelectionChanged);
    selectionTester.EndHitTesting();
}

```

```

If modeState = InkMode.Select AndAlso lassoPath Is Nothing Then
    ' Add the lasso to the InkPresenter and add the packetList
    ' to selectionTester.
    lassoPath = newStroke
    lassoPath.DrawingAttributes = selectDA.Clone()
    presenter.Strokes.Add(lassoPath)
    RemoveHandler selectionTester.SelectionChanged, _
        AddressOf selectionTester_SelectionChanged
    selectionTester.EndHitTesting()
End If

```

Uso combinato tutti.

Nell'esempio seguente è un controllo personalizzato che consente all'utente di selezionare l'input penna con un lazo.

```

using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
using System.Windows.Input;
using System.Windows.Input.StylusPlugIns;
using System.Windows.Ink;

// Enum that keeps track of whether StrokeCollectionDemo is in ink mode
// or select mode.

```

```

public enum InkMode
{
    Ink, Select
}

// This control allows the user to input and select ink. When the
// user selects ink, the lasso remains visible until they erase, or clip
// the selected strokes, or clear the selection. When the control is
// in selection mode, strokes that are selected turn red.
public class InkSelector : Label
{
    InkMode mode;

    DrawingAttributes inkDA;
    DrawingAttributes selectDA;

    InkPresenter presenter;
    IncrementalLassoHitTester selectionTester;
    StrokeCollection selectedStrokes = new StrokeCollection();

    // StylusPointCollection that collects the stylus points from the stylus events.
    StylusPointCollection stylusPoints;
    // Stroke that represents the lasso.
    Stroke lassoPath;

    DynamicRenderer renderer;

    public InkSelector()
    {
        mode = InkMode.Ink;

        // Use an InkPresenter to display the strokes on the custom control.
        presenter = new InkPresenter();
        this.Content = presenter;

        // In the constructor.
        // Selection drawing attributes use dark gray ink.
        selectDA = new DrawingAttributes();
        selectDA.Color = Colors.DarkGray;

        // ink drawing attributes use default attributes
        inkDA = new DrawingAttributes();
        inkDA.Width = 5;
        inkDA.Height = 5;

        inkDA.AttributeChanged += new PropertyDataChangedEventHandler(DrawingAttributesChanged);
        selectDA.AttributeChanged += new PropertyDataChangedEventHandler(DrawingAttributesChanged);

        // Add a DynmaicRenderer to the control so ink appears
        // to "flow" from the tablet pen.
        renderer = new DynamicRenderer();
        renderer.DrawingAttributes = inkDA;
        this.StylusPlugIns.Add(renderer);
        presenter.AttachVisuals(renderer.RootVisual,
            renderer.DrawingAttributes);
    }

    static InkSelector()
    {
        // Allow ink to be drawn only within the bounds of the control.
        Type owner = typeof(InkSelector);
        ClipToBoundsProperty.OverrideMetadata(owner,
            new FrameworkPropertyMetadata(true));
    }

    // Prepare to collect stylus packets. If Mode is set to Select,
    // get the IncrementalHitTester from the InkPresenter's newStroke
    // StrokeCollection and subscribe to its StrokeHitChanged event.
    protected override void OnStylusDown(StylusDownEventArgs e)

```

```

protected override void OnStylusDown(StylusEventArgs e)
{
    base.OnStylusDown(e);

    Stylus.Capture(this);

    // Create a new StylusPointCollection using the StylusPointDescription
    // from the stylus points in the StylusDownEventArgs.
    stylusPoints = new StylusPointCollection();
    StylusPointCollection eventPoints = e.GetStylusPoints(this, stylusPoints.Description);

    stylusPoints.Add(eventPoints);

    InitializeHitTester(eventPoints);
}

protected override void OnMouseLeftButtonDown(MouseButtonEventArgs e)
{
    base.OnMouseLeftButtonDown(e);

    Mouse.Capture(this);

    if (e.StylusDevice != null)
    {
        return;
    }

    Point pt = eGetPosition(this);

    StylusPointCollection collectedPoints = new StylusPointCollection(new Point[] { pt });

    stylusPoints = new StylusPointCollection();

    stylusPoints.Add(collectedPoints);

    InitializeHitTester(collectedPoints);
}

private void InitializeHitTester(StylusPointCollection collectedPoints)
{
    // Deselect any selected strokes.
    foreach (Stroke selectedStroke in selectedStrokes)
    {
        selectedStroke.DrawingAttributes.Color = inkDA.Color;
    }
    selectedStrokes.Clear();

    if (mode == InkMode.Select)
    {
        // Remove the previously drawn lasso, if it exists.
        if (lassoPath != null)
        {
            presenter.Strokes.Remove(lassoPath);
            lassoPath = null;
        }

        selectionTester =
            presenter.Strokes.GetIncrementalLassoHitTester(80);
        selectionTester.SelectionChanged +=
            new LassoSelectionChangedEventHandler(selectionTester_SelectionChanged);
        selectionTester.AddPoints(collectedPoints);
    }
}

// Collect the stylus packets as the stylus moves.
protected override void OnStylusMove(StylusEventArgs e)
{
    if (stylusPoints == null)
    {
        return;
    }
}

```

```

        }

        StylusPointCollection collectedPoints = e.GetStylusPoints(this, stylusPoints.Description);
        stylusPoints.Add(collectedPoints);
        AddPointsToHitTester(collectedPoints);
    }

protected override void OnMouseMove(MouseEventArgs e)
{
    base.OnMouseMove(e);

    if (e.StylusDevice != null)
    {
        return;
    }

    if (e.LeftButton == MouseButtonState.Released)
    {
        return;
    }

    stylusPoints ??= new StylusPointCollection();

    Point pt = eGetPosition(this);

    StylusPointCollection collectedPoints = new StylusPointCollection(new Point[] { pt });

    stylusPoints.Add(collectedPoints);

    AddPointsToHitTester(collectedPoints);
}

private void AddPointsToHitTester(StylusPointCollection collectedPoints)
{
    if (mode == InkMode.Select &&
        selectionTester != null &&
        selectionTester.IsValid)
    {
        // When the control is selecting strokes, add the
        // stylus packetList to selectionTester.
        selectionTester.AddPoints(collectedPoints);
    }
}

// When the user lifts the stylus, create a Stroke from the
// collected stylus points and add it to the InkPresenter.
// When the control is selecting strokes, add the
// point data to the IncrementalHitTester.
protected override void OnStylusUp(StylusEventArgs e)
{
    stylusPoints ??= new StylusPointCollection();
    StylusPointCollection collectedPoints =
        e.GetStylusPoints(this, stylusPoints.Description);

    stylusPoints.Add(collectedPoints);
    AddPointsToHitTester(collectedPoints);
    AddStrokeToPresenter();
    stylusPoints = null;

    Stylus.Capture(null);
}

protected override void OnMouseLeftButtonUp(MouseEventArgs e)
{
    base.OnMouseLeftButtonUp(e);
}

```

```

        if (e.StylusDevice != null) return;

        if (stylusPoints == null) stylusPoints = new StylusPointCollection();

        Point pt = e.GetPosition(this);

        StylusPointCollection collectedPoints = new StylusPointCollection(new Point[] { pt });

        stylusPoints.Add(collectedPoints);
        AddPointsToHitTester(collectedPoints);
        AddStrokeToPresenter();

        stylusPoints = null;

        Mouse.Capture(null);
    }

    private void AddStrokeToPresenter()
    {
        Stroke newStroke = new Stroke(stylusPoints);

        if (mode == InkMode.Ink)
        {
            // Add the stroke to the InkPresenter.
            newStroke.DrawingAttributes = inkDA.Clone();
            presenter.Strokes.Add(newStroke);
        }

        if (mode == InkMode.Select && lassoPath == null)
        {
            // Add the lasso to the InkPresenter and add the packetList
            // to selectionTester.
            lassoPath = newStroke;
            lassoPath.DrawingAttributes = selectDA.Clone();
            presenter.Strokes.Add(lassoPath);
            selectionTester.SelectionChanged -= new LassoSelectionChangedEventHandler
                (selectionTester_SelectionChanged);
            selectionTester.EndHitTesting();
        }
    }

    void selectionTester_SelectionChanged(object sender,
        LassoSelectionChangedEventArgs args)
    {
        // Change the color of all selected strokes to red.
        foreach (Stroke selectedStroke in args.SelectedStrokes)
        {
            selectedStroke.DrawingAttributes.Color = Colors.Red;
            selectedStrokes.Add(selectedStroke);
        }

        // Change the color of all unselected strokes to
        // their original color.
        foreach (Stroke unselectedStroke in args.DeselectedStrokes)
        {
            unselectedStroke.DrawingAttributes.Color = inkDA.Color;
            selectedStrokes.Remove(unselectedStroke);
        }
    }

    // Property to indicate whether the user is inputting or
    // selecting ink.
    public InkMode Mode
    {
        get
        {
            return mode;
        }
    }
}

```

```

        set
    {
        mode = value;

        // Set the DrawingAttributes of the DynamicRenderer
        if (mode == InkMode.Ink)
        {
            renderer.DrawingAttributes = inkDA;
        }
        else
        {
            renderer.DrawingAttributes = selectDA;
        }

        // Reattach the visual of the DynamicRenderer to the InkPresenter.
        presenter.DetachVisuals(renderer.RootVisual);
        presenter.AttachVisuals(renderer.RootVisual, renderer.DrawingAttributes);
    }
}

void DrawingAttributesChanged(object sender, PropertyDataChangedEventArgs e)
{
    // Reattach the visual of the DynamicRenderer to the InkPresenter
    // whenever the DrawingAttributes change.
    presenter.DetachVisuals(renderer.RootVisual);
    presenter.AttachVisuals(renderer.RootVisual, renderer.DrawingAttributes);
}

// Property to allow the user to change the pen's DrawingAttributes.
public DrawingAttributes InkDrawingAttributes
{
    get
    {
        return inkDA;
    }
}

// Property to allow the user to change the Selector's newStroke DrawingAttributes.
public DrawingAttributes SelectDrawingAttributes
{
    get
    {
        return selectDA;
    }
}
}

```

```

Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Media
Imports System.Windows.Input
Imports System.Windows.Input.StylusPlugIns
Imports System.Windows.Ink

' Enum that keeps track of whether StrokeCollectionDemo is in ink mode
' or select mode.
Public Enum InkMode
    Ink
    [Select]
End Enum 'InkMode

' This control allows the user to input and select ink. When the
' user selects ink, the lasso remains visible until they erase, or clip
' the selected strokes, or clear the selection. When the control is
' in selection mode, strokes that are selected turn red.
Public Class InkSelector
    Inherits Label

```

```

Private modeState As InkMode

Private inkDA As DrawingAttributes
Private selectDA As DrawingAttributes

Private presenter As InkPresenter
Private selectionTester As IncrementalLassoHitTester
Private selectedStrokes As New StrokeCollection()

' StylusPointCollection that collects the stylus points from the stylus events.
Private stylusPoints As StylusPointCollection

' Stroke that represents the lasso.
Private lassoPath As Stroke

Private renderer As DynamicRenderer

Public Sub New()
    modeState = InkMode.Ink

    ' Use an InkPresenter to display the strokes on the custom control.
    presenter = New InkPresenter()
    Me.Content = presenter

    ' In the constructor.
    ' Selection drawing attributes use dark gray ink.
    selectDA = New DrawingAttributes()
    selectDA.Color = Colors.DarkGray

    ' ink drawing attributes use default attributes
    inkDA = New DrawingAttributes()
    inkDA.Width = 5
    inkDA.Height = 5

    AddHandler inkDA.AttributeChanged, _
        AddressOf DrawingAttributesChanged

    AddHandler selectDA.AttributeChanged, _
        AddressOf DrawingAttributesChanged

    ' Add a DynmaicRenderer to the control so ink appears
    ' to "flow" from the tablet pen.
    renderer = New DynamicRenderer()
    renderer.DrawingAttributes = inkDA
    Me.StylusPlugIns.Add(renderer)
    presenter.AttachVisuals(renderer.RootVisual, _
        renderer.DrawingAttributes)

End Sub

Shared Sub New()
    ' Allow ink to be drawn only within the bounds of the control.
    Dim owner As Type = GetType(InkSelector)
    ClipToBoundsProperty.OverrideMetadata(owner, _
        New FrameworkPropertyMetadata(True))

End Sub

' Prepare to collect stylus packets. If Mode is set to Select,
' get the IncrementalHitTester from the InkPresenter's newStroke
' StrokeCollection and subscribe to its StrokeHitChanged event.
Protected Overrides Sub OnStylusDown(ByVal e As StylusDownEventArgs)
    MyBase.OnStylusDown(e)

    Stylus.Capture(Me)

```

```

' Create a new StylusPointCollection using the StylusPointDescription
' from the stylus points in the StylusDownEventArgs.
stylusPoints = New StylusPointCollection()
Dim eventPoints As StylusPointCollection = e.GetStylusPoints(Me, stylusPoints.Description)

stylusPoints.Add(eventPoints)

InitializeHitTester(eventPoints)

End Sub

Protected Overrides Sub OnMouseLeftButtonDown(ByVal e As MouseButtonEventArgs)
 MyBase.OnMouseLeftButtonDown(e)

 Mouse.Capture(Me)

 If Not (e.StylusDevice Is Nothing) Then
 Return
 End If

 Dim pt As Point = eGetPosition(Me)

 Dim collectedPoints As New StylusPointCollection(New Point() {pt})
 stylusPoints = New StylusPointCollection()

 stylusPoints.Add(collectedPoints)

 InitializeHitTester(collectedPoints)

End Sub

Private Sub InitializeHitTester(ByVal collectedPoints As StylusPointCollection)

 ' Deselect any selected strokes.
 Dim selectedStroke As Stroke
 For Each selectedStroke In selectedStrokes
 selectedStroke.DrawingAttributes.Color = inkDA.Color
 Next selectedStroke
 selectedStrokes.Clear()

 If modeState = InkMode.Select Then
 ' Remove the previously drawn lasso, if it exists.
 If Not (lassoPath Is Nothing) Then
 presenter.Strokes.Remove(lassoPath)
 lassoPath = Nothing
 End If

 selectionTester = presenter.Strokes.GetIncrementalLassoHitTester(80)
 AddHandler selectionTester.SelectionChanged, AddressOf selectionTester_SelectionChanged
 selectionTester.AddPoints(collectedPoints)
 End If

End Sub

' Collect the stylus packets as the stylus moves.
Protected Overrides Sub OnStylusMove(ByVal e As StylusEventArgs)

 If stylusPoints Is Nothing Then
 Return
 End If

 Dim collectedPoints As StylusPointCollection = e.GetStylusPoints(Me, stylusPoints.Description)
 stylusPoints.Add(collectedPoints)
 AddPointsToHitTester(collectedPoints)

End Sub

```

```

Protected Overrides Sub OnMouseMove(ByVal e As MouseEventArgs)
    MyBase.OnMouseMove(e)

    If Not (e.StylusDevice Is Nothing) Then
        Return
    End If

    If e.LeftButton = MouseButtons.Released Then
        Return
    End If

    If stylusPoints Is Nothing Then
        stylusPoints = New StylusPointCollection()
    End If

    Dim pt As Point = e.GetPosition(Me)

    Dim collectedPoints As New StylusPointCollection(New Point() {pt})

    stylusPoints.Add(collectedPoints)

    AddPointsToHitTester(collectedPoints)

End Sub

Private Sub AddPointsToHitTester(ByVal collectedPoints As StylusPointCollection)

    If modeState = InkMode.Select AndAlso _
        Not selectionTester Is Nothing AndAlso _
        selectionTester.IsValid Then

        ' When the control is selecting strokes, add the
        ' stylus packetList to selectionTester.
        selectionTester.AddPoints(collectedPoints)
    End If

End Sub

' When the user lifts the stylus, create a Stroke from the
' collected stylus points and add it to the InkPresenter.
' When the control is selecting strokes, add the
' point data to the IncrementalHitTester.

Protected Overrides Sub OnStylusUp(ByVal e As StylusEventArgs)

    If stylusPoints Is Nothing Then
        stylusPoints = New StylusPointCollection()
    End If

    Dim collectedPoints As StylusPointCollection = _
        e.GetStylusPoints(Me, stylusPoints.Description)

    stylusPoints.Add(collectedPoints)
    AddPointsToHitTester(collectedPoints)
    AddStrokeToPresenter()

    stylusPoints = Nothing

    Stylus.Capture(Nothing)

End Sub

Protected Overrides Sub OnMouseLeftButtonUp(ByVal e As MouseButtonEventArgs)
    MyBase.OnMouseLeftButtonUp(e)

```

```

If Not (e.StylusDevice Is Nothing) Then
    Return
End If
If stylusPoints Is Nothing Then
    stylusPoints = New StylusPointCollection()
End If
Dim pt As Point = e.GetPosition(Me)

Dim collectedPoints As New StylusPointCollection(New Point() {pt})

stylusPoints.Add(collectedPoints)
AddPointsToHitTester(collectedPoints)
AddStrokeToPresenter()

stylusPoints = Nothing

Mouse.Capture(Nothing)

End Sub

Private Sub AddStrokeToPresenter()
    Dim newStroke As New Stroke(stylusPoints)

    If modeState = InkMode.Ink Then
        ' Add the stroke to the InkPresenter.
        newStroke.DrawingAttributes = inkDA.Clone()
        presenter.Strokes.Add(newStroke)
    End If

    If modeState = InkMode.Select AndAlso lassoPath Is Nothing Then
        ' Add the lasso to the InkPresenter and add the packetList
        ' to selectionTester.
        lassoPath = newStroke
        lassoPath.DrawingAttributes = selectDA.Clone()
        presenter.Strokes.Add(lassoPath)
        RemoveHandler selectionTester.SelectionChanged, _
            AddressOf selectionTester_SelectionChanged
        selectionTester.EndHitTesting()
    End If
End Sub

Private Sub selectionTester_SelectionChanged(ByVal sender As Object, _
    ByVal args As LassoSelectionChangedEventArgs)

    ' Change the color of all selected strokes to red.
    Dim selectedStroke As Stroke
    For Each selectedStroke In args.SelectedStrokes
        selectedStroke.DrawingAttributes.Color = Colors.Red
        selectedStrokes.Add(selectedStroke)
    Next selectedStroke

    ' Change the color of all unselected strokes to
    ' their original color.
    Dim unselectedStroke As Stroke
    For Each unselectedStroke In args.DeselectedStrokes
        unselectedStroke.DrawingAttributes.Color = inkDA.Color
        selectedStrokes.Remove(unselectedStroke)
    Next unselectedStroke

End Sub

' Property to indicate whether the user is inputting or
' selecting ink.
Public Property Mode() As InkMode
    Get
        Return Mode
    End Get
End Property

```

```

    Reattach mode
End Get

Set(ByVal value As InkMode)
modeState = value

' Set the DrawingAttributes of the DynamicRenderer
If modeState = InkMode.Ink Then
    renderer.DrawingAttributes = inkDA
Else
    renderer.DrawingAttributes = selectDA
End If

' Reattach the visual of the DynamicRenderer to the InkPresenter.
presenter.DetachVisuals(renderer.RootVisual)
presenter.AttachVisuals(renderer.RootVisual, renderer.DrawingAttributes)
End Set
End Property

Private Sub DrawingAttributesChanged(ByVal sender As Object, _
                                      ByVal e As PropertyDataChangedEventArgs)

    ' Reattach the visual of the DynamicRenderer to the InkPresenter
    ' whenever the DrawingAttributes change.
    presenter.DetachVisuals(renderer.RootVisual)
    presenter.AttachVisuals(renderer.RootVisual, _
                           renderer.DrawingAttributes)

End Sub

' Property to allow the user to change the pen's DrawingAttributes.
Public ReadOnly Property InkDrawingAttributes() As DrawingAttributes
    Get
        Return inkDA
    End Get
End Property

' Property to allow the user to change the Selector's newStroke DrawingAttributes.
Public ReadOnly Property SelectDrawingAttributes() As DrawingAttributes
    Get
        Return selectDA
    End Get
End Property

End Class

```

Vedere anche

- [IncrementalLassoHitTester](#)
- [StrokeCollection](#)
- [StylusPointCollection](#)
- [Creazione di un controllo di input penna](#)

Procedura: Aggiungere dati personalizzati ai dati dell'input penna

23/10/2019 • 3 minutes to read • [Edit Online](#)

È possibile aggiungere dati personalizzati all'input penna che verrà salvato quando l'input penna viene salvata come formato di input penna serializzato (Serialized Format). È possibile salvare i dati personalizzati per il [DrawingAttributes](#), il [StrokeCollection](#), o [Stroke](#). La possibilità di salvare i dati personalizzati nelle tre oggetti ti offre la possibilità di decidere il modo migliore per salvare i dati. Tutte le tre classi usano metodi simili per archiviare e accedere ai dati personalizzati.

Solo i tipi seguenti possono essere salvati come dati personalizzati:

- [Boolean](#)
- [Boolean\[\]](#)
- [Byte](#)
- [Byte\[\]](#)
- [Char](#)
- [Char\[\]](#)
- [DateTime](#)
- [DateTime\[\]](#)
- [Decimal](#)
- [Decimal\[\]](#)
- [Double](#)
- [Double\[\]](#)
- [Int16](#)
- [Int16\[\]](#)
- [Int32](#)
- [Int32\[\]](#)
- [Int64](#)
- [Int64\[\]](#)
- [Single](#)
- [Single\[\]](#)
- [String](#)
- [UInt16](#)
- [UInt16\[\]](#)
- [UInt32](#)

- [UInt32\[\]](#)
- [UInt64](#)
- [UInt64\[\]](#)

Esempio

Nell'esempio seguente viene illustrato come aggiungere e recuperare i dati personalizzati da un [StrokeCollection](#).

```
Guid timestamp = new Guid("12345678-9012-3456-7890-123456789012");

// Add a timestamp to the StrokeCollection.
private void AddTimestamp()
{
    inkCanvas1.Strokes.AddPropertyData(timestamp, DateTime.Now);
}

// Get the timestamp of the StrokeCollection.
private void GetTimestamp()
{
    if (inkCanvas1.Strokes.ContainsPropertyData(timestamp))
    {
        object date = inkCanvas1.Strokes.GetPropertyData(timestamp);

        if (date is DateTime)
        {
            MessageBox.Show("This StrokeCollection's timestamp is " +
                ((DateTime)date).ToString());
        }
        else
        {
            MessageBox.Show(
                "The StrokeCollection does not have a timestamp.");
        }
    }
}
```

L'esempio seguente crea un'applicazione che visualizza un [InkCanvas](#) e due pulsanti. Il pulsante, `switchAuthor`, consente due penne utilizzabile dai due diversi autori. Il pulsante `changePenColors` modifica il colore di ogni tratto nel [InkCanvas](#) in base all'autore. L'applicazione definisce due [DrawingAttributes](#) degli oggetti e aggiunge una proprietà personalizzata per ognuno dei quali indica l'autore che ha tracciato la [Stroke](#). Quando l'utente fa clic `changePenColors`, l'applicazione modifica l'aspetto del tratto a seconda del valore della proprietà personalizzata.

```

<Window x:Class="Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Adding Custom Data to Ink" Height="500" Width="700"
    >
    <DockPanel Name="root">

        <StackPanel Background="DarkSlateBlue">
            <Button Name="switchAuthor" Click="switchAuthor_click" >
                Switch to student's pen
            </Button>
            <Button Name="changePenColors" Click="changeColor_click" >
                Change the color of the pen ink
            </Button>
        </StackPanel>
        <InkCanvas Name="inkCanvas1">
        </InkCanvas>
    </DockPanel>
</Window>

```

```

using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Windows.Ink;

/// <summary>
/// Interaction logic for Window1.xaml
/// </summary>

public partial class Window1 : Window
{
    Guid authorGuid = new Guid("12345678-9012-3456-7890-123456789012");
    DrawingAttributes teachersDA = new DrawingAttributes();
    DrawingAttributes studentsDA = new DrawingAttributes();
    string teacher = "teacher";
    string student = "student";
    bool useStudentPen = false;

    public Window1()
    {
        InitializeComponent();

        teachersDA.Color = Colors.Red;
        teachersDA.Width = 5;
        teachersDA.Height = 5;
        teachersDA.AddPropertyData(authorGuid, teacher);

        studentsDA.Color = Colors.Blue;
        studentsDA.Width = 5;
        studentsDA.Height = 5;
        studentsDA.AddPropertyData(authorGuid, student);

        inkCanvas1.DefaultDrawingAttributes = teachersDA;
    }

    // Switch between using the 'pen' DrawingAttributes and the
    // 'highlighter' DrawingAttributes.
    void switchAuthor_click(Object sender, RoutedEventArgs e)
    {
        useStudentPen = !useStudentPen;
    }
}

```

```
if (useStudentPen)
{
    switchAuthor.Content = "Use teacher's pen";
    inkCanvas1.DefaultDrawingAttributes = studentsDA;
}
else
{
    switchAuthor.Content = "Use student's pen";
    inkCanvas1.DefaultDrawingAttributes = teachersDA;
}

// Change the color of the ink that on the InkCanvas that used the pen.
void changeColor_click(Object sender, RoutedEventArgs e)
{
    foreach (Stroke s in inkCanvas1.Strokes)
    {
        if (s.DrawingAttributes.ContainsPropertyData(authorGuid))
        {
            object data = s.DrawingAttributes.GetPropertyData(authorGuid);

            if ((data is string) && ((string)data == teacher))
            {
                s.DrawingAttributes.Color = Colors.Black;
            }
            if ((data is string) && ((string)data == student))
            {
                s.DrawingAttributes.Color = Colors.Green;
            }
        }
    }
}
```

Procedura: Cancellare l'input penna da un controllo personalizzato

23/10/2019 • 5 minutes to read • [Edit Online](#)

Il [IncrementalStrokeHitTester](#) determina se il tratto attualmente tracciato interseca un altro tratto. Ciò è utile per la creazione di un controllo che consente di cancellare le parti di un tratto, il modo un utente può in un' [InkCanvas](#) quando il [EditingMode](#) è impostata su [EraseByPoint](#).

Esempio

L'esempio seguente crea un controllo personalizzato che consente all'utente di cancellare le parti dei tratti. Questo esempio crea un controllo che contiene l'input penna quando viene inizializzato. Per creare un controllo che consente di raccogliere input penna, vedere [creazione di un controllo Input penna](#).

```
using System.Windows;
using System.Windows.Controls;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.IO;

// This control initializes with ink already on it and allows the
// user to erase the ink with the tablet pen or mouse.
public class InkEraser : Label
{
    IncrementalStrokeHitTester eraseTester;

    InkPresenter presenter;

    string strokesString =
        @"ALwHAxdiEETLgIgERYQBGwIAJAFGhAEbAgAkAQUBOBkgMgkA9P8CAekiOkUzCQD4n"
        + "wIBWiA6RTgIAP4DAAAAgH8RAACAPx8JEQAAAAAAAPA/CiUh/A6N4HR0AivFX8Vs"
        + "IfsiuyLSaIeDSLwabiHm0GgUDi+KZkACjsQh/A9t4IC5VJpfLhaIxxyxIh7Dncnh"
        + "+6e7qODwoER1PAw8EpGoJAg61IKjCYXBYDA4DAIHF67KIHAAojB4fwMteBn+RKB"
        + "lziaIfwWTeCwePqbM8WgIeeQCDQ0FRvcIAKNA+H8B8XgQHkUbjQTnGuaZns3l4h"
        + "/Dwt4a0YKBB0A94D6HRCaiGnp5CS8LExMLB1t0gYIAKUB0H8KnXhU71Mold+tcbi"
        + "kChkqu2EtPYxp9bmYCH8HDhg4ZhMIwRyMHH+4Jt8nleX8c0/D/AkYwxJGiHkkQgm"
        + "Ch9CqcFhMDQCbwAwuR2eAACmgdh/EpF41A6XMUfhMXgMHgVDxBFPrKpZII5EINA"
        + "OA64M+J4Lw1CIoAh/B2x4PS4bQodAopEI5IJBki4waEx2Qy+dy+ayHgleEmmHH8c"
        + "e3MZOCGw5Twd3CwsHAWMCgRAEAgElKwOHZKBApaGIfxezeL0uN02N8IzwaGEpNIJ"
        + "ZxHnELy0j0GfyuU6Fgmhp1IgIfwYgeDHeaI1vj0tZgcHgHAYb9hUCgEFgsPm1xnM"
        + "ZkYhsnYJgZeZh4uAgCgnSBI0Jv40AgwCmkgh/GrR41X4dGoRJL9EKra5HKY7IZ3C"
        + "4fj/M06olSoU8kkehUbh8jkMdCH8IJXhAxhMCK8JuN1mNyh0YiEumUwn2wMRxyHw"
        + "2TzWmzeb020zGKxMITwIhnrzbb44zRhGEKRhCM4zrr6sQKXRWH8kuXkmPj0DiXC"
        + "gcJbC9HZZgkKgUG4bLh3YrwJHAYw2CAh/CiN4Tq7DOZr4BB/AFtdOWW5P2h1Wkzv"
        + "14+YwqXF8d5fZ7ih51QkbB4LqrLAYDBIDABA4B04nAIACpvIIfy4BeXA2DRSRQ1L"
        + "oHHsYQ/KMX1sv18rn8Xkcdg+G9NVaUWimUDYk9Ah/BoF4M0YBCqZPyqk8dwLf7hd"
        + "YNBjFLKBnqZTqNubWsh19VoM1reFYZYQEBGUdAwKEjYuDQKBgICBgcAGIOAg4nI"
        + "80ACloSh/BF14Gf/I0t6FXF8F4ToPCzz1PwP4+B+DhmQ0847rfDeCcG8eKh/EZV"
        + "4i9eZt8A9nUF8VxaUe5gr17YrPaHfpRKJNx4yHmUuj1vicwmMBEAjUVgKB61A=";

    public InkEraser()
    {
        presenter = new InkPresenter();
        this.Content = presenter;

        // Create a StrokeCollection the string and add it to
        StrokeCollectionConverter converter =
            new StrokeCollectionConverter();
    }
}
```

```

        new StrokeCollectionConverter(),
    }

    if (converter.CanConvertFrom(typeof(string)))
    {
        StrokeCollection newStrokes =
            converter.ConvertFrom(strokesString) as StrokeCollection;
        presenter.Strokes.Clear();
        presenter.Strokes.Add(newStrokes);
    }
}

protected override void OnStylusDown(StylusDownEventArgs e)
{
    base.OnStylusDown(e);
    StylusPointCollection points = e.GetStylusPoints(this);

    InitializeEraserHitTester(points);
}

protected override void OnMouseLeftButtonDown(MouseButtonEventArgs e)
{
    base.OnMouseLeftButtonDown(e);

    if (e.StylusDevice != null)
    {
        return;
    }

    Point pt = eGetPosition(this);

    StylusPointCollection collectedPoints = new StylusPointCollection(new Point[] { pt });

    InitializeEraserHitTester(collectedPoints);
}

// Prepare to collect stylus packets. Get the
// IncrementalHitTester from the InkPresenter's
// StrokeCollection and subscribe to its StrokeHitChanged event.
private void InitializeEraserHitTester(StylusPointCollection points)
{
    EllipseStylusShape eraserTip = new EllipseStylusShape(3, 3, 0);
    eraseTester =
        presenter.Strokes.GetIncrementalStrokeHitTester(eraserTip);
    eraseTester.StrokeHit += new StrokeHitEventHandler(eraseTester_StrokeHit);
    eraseTester.AddPoints(points);
}

protected override void OnStylusMove(StylusEventArgs e)
{
    StylusPointCollection points = e.GetStylusPoints(this);

    AddPointsToEraserHitTester(points);
}

protected override void OnMouseMove(MouseEventArgs e)
{
    base.OnMouseMove(e);

    if (e.StylusDevice != null)
    {
        return;
    }

    if (e.LeftButton == MouseButtonState.Released)
    {
        return;
    }

    Point pt = eGetPosition(this);
}

```

```

        StylusPointCollection collectedPoints = new StylusPointCollection(new Point[] { pt });

        AddPointsToEraserHitTester(collectedPoints);
    }

    // Collect the StylusPackets as the stylus moves.
    private void AddPointsToEraserHitTester(StylusPointCollection points)
    {
        if (eraseTester.IsValid)
        {
            eraseTester.AddPoints(points);
        }
    }

    // Unsubscribe from the StrokeHitChanged event when the
    // user lifts the stylus.
    protected override void OnStylusUp(StylusEventArgs e)
    {
        StylusPointCollection points = e.GetStylusPoints(this);

        StopEraseHitTesting(points);
    }

    protected override void OnMouseLeftButtonUp(MouseEventArgs e)
    {
        base.OnMouseLeftButtonUp(e);

        if (e.StylusDevice != null)
        {
            return;
        }

        Point pt = e.GetPosition(this);

        StylusPointCollection collectedPoints = new StylusPointCollection(new Point[] { pt });

        StopEraseHitTesting(collectedPoints);
    }

    private void StopEraseHitTesting(StylusPointCollection points)
    {
        eraseTester.AddPoints(points);
        eraseTester.StrokeHit -= new
            StrokeHitEventHandler(eraseTester_StrokeHit);
        eraseTester.EndHitTesting();
    }

    // When the stylus intersects a stroke, erase that part of
    // the stroke. When the stylus dissects a stroke, the
    // Stroke.Erase method returns a StrokeCollection that contains
    // the two new strokes.
    void eraseTester_StrokeHit(object sender,
        StrokeHitEventArgs args)
    {
        StrokeCollection eraseResult =
            args.GetPointEraseResults();
        StrokeCollection strokesToReplace = new StrokeCollection();
        strokesToReplace.Add(args.HitStroke);

        // Replace the old stroke with the new one.
        if (eraseResult.Count > 0)
        {
            presenter.Strokes.Replace(strokesToReplace, eraseResult);
        }
        else
        {
            presenter.Strokes.Remove(strokesToReplace);
        }
    }
}

```

```
    }  
}
```

```
Imports System.Windows  
Imports System.Windows.Controls  
Imports System.Windows.Ink  
Imports System.Windows.Input  
Imports System.Windows.Media  
Imports System.IO  
  
' This control initializes with ink already on it and allows the  
' user to erase the ink with the tablet pen or mouse.  
  
Public Class InkEraser  
    Inherits Label  
    Private eraseTester As IncrementalStrokeHitTester  
  
    Private presenter As InkPresenter  
  
    ' The base-64 encoded string that contains ink data  
    ' in ink serialized format (ISF).  
    Private strokesString As String = _  
        "ALwHAxdIEETLgIgERYQBGwIAJAFGhAEbAgAkAQUBOBkgMgkA9P8CAekiOkUzCQD4n" _  
        & "wIBWiA6RTgIAP4DAAAAGh8RAACAPx8JEQAAAAAAAPA/CiUh/A6N4HR0AivFX8Vs" _  
        & "IfsiuyLSaIeDSLwabiHm0GgUDi+KZkACjsQh/A9t4IC5VJpfLhaIxxyXlh7Dncnh" _  
        & "+6e7qODwoER1PAw8EpGoJagh61IKjCYXBVD4DAIHf67KIHAAojB4fwMteBn+RKB" _  
        & "lziaIfwlTeCwePqbM8WgIeeQCDQoFRvcIAKNA+h8B8XgQHkUbjQTtnguaZns314h" _  
        & "/Dwt4a0YKBB0A94D6HRCAiGnp5CS8LExMLB1t0gYIAKUBO8KnXhU7lMold+tcbi" _  
        & "kChkqu2EtPYxp9bmYCH8HDhg4ZhMIwRyMHH+4Jt8nleX8c0/D/AkYwxJGiHkkQgM" _  
        & "Ch9CqcFhMDQCBwAwuR2eAACmgdh/EpF41A6XMUfhMXgMHgVdxBFpRKpZII5EINA" _  
        & "OA64M+J4Lw1Ci0Ah/B2x4PS4bQodAopEI5IJBki4waEx2Qy+dy+ayHgleEmmHH8c" _  
        & "e3MZOOGw5TwD3CwsHAWMCgRAEAgElKwOHZKBApaGifxezeL0uN02N8IzwaGEpNIJ" _  
        & "ZxHnElYoj0GfyuU6FgmhpIgIfwYgeDHeaI1vj0tZgcHgHAYb9hUCgEFgsPm1xm" _  
        & "ZkYhsnYJgZeZh4uAgCgnSBl0jv40AgwCmkgh/GrR41X4dGoRjl9EKra5HKY7IZ3C" _  
        & "4fj/M06o1SoU8kkehUbh8jkMdCh8IJXhAXhMck8JuNlmNyh0YiEumUwn2wMRxyHw" _  
        & "2TzWmzeb020zGKxMITwIhnrzbb44zRhGEKRhCM4zrr6sQKXRWH8kuXkmPj0DiXC" _  
        & "gcJbC9HZgkKgUG4bLh3YrwJHAYw2CaH/CiN4Tq7DOzr4BB/AFtd0W5P2h1Wkzv" _  
        & "14+YwqXF8d5fZ7ih51QkbB4LqrLAYDBIDABA4B04nAICApvIIfy4BeXA2DRSrQll" _  
        & "oHhsYQ/KMX1sv18rn8Xkcdg+G9NVaUWimUDYk9Ah/Bof4M0YBCqZPYqk8dwLf7hd" _  
        & "YNBJFLKBnqZTqNubWsh19VoM1reFYZYQEBUGUsDAwKEjYuDQKBgICBgcAGIOAg4nI" _  
        & "80ACloSh/BF14Gf/I0t6FXfF8F4ToPCzz1PwP4+B+DHmQ0847rfDeCcG8eKh/EZV" _  
        & "4i9eZt8A9nUF8VxaUe5gr17YrPaHfpRKJNx4yHmUuj1vicwmMBEAjUVgKB61A="  
  
    Public Sub New()  
        presenter = New InkPresenter()  
        Me.Content = presenter  
  
        ' Create a StrokeCollection the string and add it to  
        Dim converter As New StrokeCollectionConverter()  
  
        If converter.CanConvertFrom(GetType(String)) Then  
            Dim newStrokes As StrokeCollection = converter.ConvertFrom(strokesString)  
  
            presenter.Strokes.Clear()  
            presenter.Strokes.Add(newStrokes)  
        End If  
  
    End Sub  
  
    Protected Overrides Sub OnStylusDown(ByVal e As StylusDownEventArgs)  
        MyBase.OnStylusDown(e)  
        Dim points As StylusPointCollection = e.GetStylusPoints(Me)  
  
        InitializeEraserHitTester(points)  
    End Sub
```

```

Protected Overrides Sub OnMouseLeftButtonDown(ByVal e As MouseButtonEventArgs)
    MyBase.OnMouseLeftButtonDown(e)

    If Not (e.StylusDevice Is Nothing) Then
        Return
    End If

    Dim pt As Point = e.GetPosition(Me)

    Dim collectedPoints As New StylusPointCollection(New Point() {pt})

    InitializeEraserHitTester(collectedPoints)

End Sub

' Get the IncrementalHitTester from the InkPresenter's
' StrokeCollection and subscribe to its StrokeHitChanged event.
Private Sub InitializeEraserHitTester(ByVal points As StylusPointCollection)

    Dim eraserTip As New EllipseStylusShape(3, 3, 0)
    eraseTester = presenter-strokes.GetIncrementalStrokeHitTester(eraserTip)
    AddHandler eraseTester.StrokeHit, AddressOf eraseTester_StrokeHit
    eraseTester.AddPoints(points)

End Sub

Protected Overrides Sub OnStylusMove(ByVal e As StylusEventArgs)
    Dim points As StylusPointCollection = e.GetStylusPoints(Me)

    AddPointsToEraserHitTester(points)

End Sub

Protected Overrides Sub OnMouseMove(ByVal e As MouseEventArgs)
    MyBase.OnMouseMove(e)

    If Not (e.StylusDevice Is Nothing) Then
        Return
    End If

    If e.LeftButton = MouseButtons.Released Then
        Return
    End If

    Dim pt As Point = e.GetPosition(Me)

    Dim collectedPoints As New StylusPointCollection(New Point() {pt})

    AddPointsToEraserHitTester(collectedPoints)

End Sub

' Collect the StylusPackets as the stylus moves.
Private Sub AddPointsToEraserHitTester(ByVal points As StylusPointCollection)

    If eraseTester.IsValid Then
        eraseTester.AddPoints(points)
    End If

End Sub

' Unsubscribe from the StrokeHitChanged event when the
' user lifts the stylus.

```

```

Protected Overrides Sub OnStylusUp(ByVal e As StylusEventArgs)
    Dim points As StylusPointCollection = e.GetStylusPoints(Me)
    StopEraseHitTesting(points)
End Sub

Protected Overrides Sub OnMouseLeftButtonUp(ByVal e As MouseButtonEventArgs)
    MyBase.OnMouseLeftButtonUp(e)

    If Not (e.StylusDevice Is Nothing) Then
        Return
    End If

    Dim pt As Point = eGetPosition(Me)

    Dim collectedPoints As New StylusPointCollection(New Point() {pt})
    StopEraseHitTesting(collectedPoints)
End Sub

Private Sub StopEraseHitTesting(ByVal points As StylusPointCollection)
    eraseTester.AddPoints(points)
    RemoveHandler eraseTester.StrokeHit, AddressOf eraseTester_StrokeHit
    eraseTester.EndHitTesting()
End Sub

' When the stylus intersects a stroke, erase that part of
' the stroke. When the stylus dissects a stroke, the
' Stroke.Erase method returns a StrokeCollection that contains
' the two new strokes.
Private Sub eraseTester_StrokeHit(ByVal sender As Object, ByVal args As StrokeHitEventArgs)

    Dim eraseResult As StrokeCollection = args.GetPointEraseResults()
    Dim strokesToReplace As New StrokeCollection()
    strokesToReplace.Add(args.HitStroke)

    ' Replace the old stroke with the new one.
    If eraseResult.Count > 0 Then
        presenter.Strokes.Replace(strokesToReplace, eraseResult)
    Else
        presenter.Strokes.Remove(strokesToReplace)
    End If
End Sub
End Class

```

Procedura: Riconoscere i movimenti dell'applicazione

23/10/2019 • 2 minutes to read • [Edit Online](#)

Nell'esempio seguente viene illustrato come cancellare l'input penna quando un utente apporta una **ScratchOut** movimenti su un' **InkCanvas**. Questo esempio si presuppone un' **InkCanvas**, denominato `inkCanvas1`, viene dichiarata nel file XAML.

Esempio

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Ink;
using System.Collections.ObjectModel;

public partial class Window1 : Window
{
    public Window1()
    {
        InitializeComponent();

        if (inkCanvas1.IsGestureRecognizerAvailable)
        {
            inkCanvas1.EditingMode = InkCanvasEditingStyle.InkAndGesture;
            inkCanvas1.Gesture += new InkCanvasGestureEventHandler(inkCanvas1_Gesture);
            inkCanvas1.SetEnabledGestures(
                new ApplicationGesture[] { ApplicationGesture.ScratchOut });
        }
    }

    void inkCanvas1_Gesture(object sender, InkCanvasGestureEventArgs e)
    {
        ReadOnlyCollection<GestureRecognitionResult> gestureResults =
            e.GetGestureRecognitionResults();

        // Check the first recognition result for a gesture.
        if ((gestureResults[0].RecognitionConfidence ==
            RecognitionConfidence.Strong) &&
            (gestureResults[0].ApplicationGesture ==
            ApplicationGesture.ScratchOut))
        {
            StrokeCollection hitStrokes = inkCanvas1.Strokes.HitTest(
                e.Strokes.GetBounds(), 10);

            if (hitStrokes.Count > 0)
            {
                inkCanvas1.Strokes.Remove(hitStrokes);
            }
        }
    }
}
```

```

Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Ink
Imports System.Collections.ObjectModel

Class Window1
    Inherits Window

    Public Sub New()
        InitializeComponent()

        If inkCanvas1.IsGestureRecognizerAvailable Then
            inkCanvas1.EditingMode = InkCanvasEditingMode.InkAndGesture
            AddHandler inkCanvas1.Gesture, AddressOf inkCanvas1_Gesture
            inkCanvas1.SetEnabledGestures(New ApplicationGesture() {ApplicationGesture.ScratchOut})
        End If

    End Sub

    Private Sub inkCanvas1_Gesture(ByVal sender As Object, ByVal e As InkCanvasGestureEventArgs)

        Dim gestureResults As ReadOnlyCollection(Of GestureRecognitionResult) = _
            e.GetGestureRecognitionResults()

        ' Check the first recognition result for a gesture.
        If gestureResults(0).RecognitionConfidence = _
            RecognitionConfidence.Strong AndAlso _
            gestureResults(0).ApplicationGesture = _
            ApplicationGesture.ScratchOut Then

            Dim hitStrokes As StrokeCollection = _
                inkCanvas1.Strokes.HitTest(e.Strokes.GetBounds(), 10)

            If hitStrokes.Count > 0 Then
                inkCanvas1.Strokes.Remove(hitStrokes)
            End If
        End If

    End Sub
End Class

```

Vedere anche

- [ApplicationGesture](#)
- [InkCanvas](#)
- [Gesture](#)

Procedura: Trascinare l'input penna

23/10/2019 • 3 minutes to read • [Edit Online](#)

Esempio

L'esempio seguente crea un'applicazione che consente all'utente di trascinare i tratti selezionati da una [InkCanvas](#) a altro.

```
<Window x:Class="Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="InkDragDropSample" Height="500" Width="700"
    >
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>

    <InkCanvas Name="ic1" AllowDrop="True"
        Grid.Column="0" Grid.Row="0"
        Margin="10,10,10,10" Background="AliceBlue"
        PreviewMouseDown="InkCanvas_PreviewMouseDown"
        Drop="InkCanvas_Drop"/>

    <InkCanvas Name="ic2" AllowDrop="True"
        Grid.Column="1" Grid.Row="0"
        Margin="10,10,10,10" Background="Beige"
        PreviewMouseDown="InkCanvas_PreviewMouseDown"
        Drop="InkCanvas_Drop"/>

    <CheckBox Grid.Row="1"
        Checked="switchToSelect" Unchecked="switchToInk">
        Select Mode
    </CheckBox>
</Grid>
</Window>
```

```
using System;
using System.IO;
using System.Windows;
using System.Windows.Ink;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Input;
using System.Windows.Media;

public partial class Window1 : Window
{
    public Window1()
    {
        InitializeComponent();
    }

    void InkCanvas_PreviewMouseDown(object sender, MouseEventArgs e)
```

```

    {
        InkCanvas ic = (InkCanvas)sender;

        Point pt = e.GetPosition(ic);

        // If the user is moving selected strokes, prepare the strokes to be
        // moved to another InkCanvas.
        if (ic.HitTestSelection(pt) ==
            InkCanvasSelectionHitResult.Selection)
        {
            StrokeCollection selectedStrokes = ic.GetSelectedStrokes();
            StrokeCollection strokesToMove = selectedStrokes.Clone();

            // Remove the offset of the selected strokes so they
            // are positioned when the strokes are dropped.
            Rect inkBounds = strokesToMove.GetBounds();
            TranslateStrokes(strokesToMove, -inkBounds.X, -inkBounds.Y);

            // Perform drag and drop.
            MemoryStream ms = new MemoryStream();
            strokesToMove.Save(ms);
            DataObject dataObject = new DataObject(
                StrokeCollection.InkSerializedFormat, ms);

            DragDropEffects effects =
                DragDrop.DoDragDrop(ic, dataObject,
                    DragDropEffects.Move);

            if ((effects & DragDropEffects.Move) ==
                DragDropEffects.Move)
            {
                // Remove the selected strokes
                // from the current InkCanvas.
                ic.Strokes.Remove(selectedStrokes);
            }
        }
    }

    void InkCanvas_Drop(object sender, DragEventArgs e)
    {
        // Get the strokes that were moved.
        InkCanvas ic = (InkCanvas)sender;
        MemoryStream ms = (MemoryStream)e.Data.GetData(
            StrokeCollection.InkSerializedFormat);
        ms.Position = 0;
        StrokeCollection strokes = new StrokeCollection(ms);

        // Translate the strokes to the position at which
        // they were dropped.
        Point pt = e.GetPosition(ic);
        TranslateStrokes(strokes, pt.X, pt.Y);

        // Add the strokes to the InkCanvas and keep them selected.
        ic.Strokes.Add(strokes);
        ic.Select(strokes);
    }

    // Helper method that translates the specified strokes.
    void TranslateStrokes(StrokeCollection strokes,
        double x, double y)
    {
        Matrix mat = new Matrix();
        mat.Translate(x, y);
        strokes.Transform(mat, false);
    }

    void switchToSelect(object sender, RoutedEventArgs e)
    {
        ic1.EditingMode = InkCanvasEditingMode.Select;
    }
}

```

```

        ic1.EditingMode = InkCanvasEditingStyle.Select,
        ic2.EditingMode = InkCanvasEditingStyle.Select;
    }

    void switchToInk(object sender, RoutedEventArgs e)
    {
        ic1.EditingMode = InkCanvasEditingStyle.Ink;
        ic2.EditingMode = InkCanvasEditingStyle.Ink;
    }
}

```

```

Imports System.IO
Imports System.Windows
Imports System.Windows.Ink
Imports System.Windows.Controls
Imports System.Windows.Data
Imports System.Windows.Input
Imports System.Windows.Media

Class Window1
    Inherits Window

    Public Sub New()
        InitializeComponent()
    End Sub

    Private Sub InkCanvas_PreviewMouseDown(ByVal sender As Object, _
                                         ByVal e As MouseButtonEventArgs)

        Dim ic As InkCanvas = CType(sender, InkCanvas)

        Dim pt As Point = e.GetPosition(ic)

        ' If the user is moving selected strokes, prepare the strokes to be
        ' moved to another InkCanvas.
        If ic.HitTestSelection(pt) = InkCanvasSelectionHitResult.Selection Then

            Dim selectedStrokes As StrokeCollection = _
                ic.GetSelectedStrokes()

            Dim strokesToMove As StrokeCollection = _
                selectedStrokes.Clone()

            ' Remove the offset of the selected strokes so they
            ' are positioned when the strokes are dropped.
            Dim inkBounds As Rect = strokesToMove.GetBounds()
            TranslateStrokes(strokesToMove, -inkBounds.X, -inkBounds.Y)

            ' Perform drag and drop.
            Dim ms As New MemoryStream()
            strokesToMove.Save(ms)

            Dim dataObject As New DataObject _
                (StrokeCollection.InkSerializedFormat, ms)

            Dim effects As DragDropEffects = _
                DragDrop.DoDragDrop(ic, dataObject, DragDropEffects.Move)

            If (effects And DragDropEffects.Move) = DragDropEffects.Move Then

                ' Remove the selected strokes from the current InkCanvas.
                ic.Strokes.Remove(selectedStrokes)
            End If
        End If
    End Sub

```

```

Private Sub InkCanvas_Drop(ByVal sender As Object, _
    ByVal e As DragEventArgs)

    ' Get the strokes that were moved.
    Dim ic As InkCanvas = CType(sender, InkCanvas)
    Dim ms As MemoryStream = CType(e.Data.GetData( _
        StrokeCollection.InkSerializedFormat), _
        MemoryStream)

    ms.Position = 0
    Dim strokes As New StrokeCollection(ms)

    ' Translate the strokes to the position at which
    ' they were dropped.
    Dim pt As Point = e.GetPosition(ic)
    TranslateStrokes(strokes, pt.X, pt.Y)

    ' Add the strokes to the InkCanvas and keep them selected.
    ic.Strokes.Add(strokes)
    ic.Select(strokes)

End Sub

' Helper method that translates the specified strokes.
Sub TranslateStrokes(ByVal strokes As StrokeCollection, _
    ByVal x As Double, ByVal y As Double)

    Dim mat As New Matrix()
    mat.Translate(x, y)
    strokes.Transform(mat, False)

End Sub

Private Sub switchToSelect(ByVal sender As Object, _
    ByVal e As RoutedEventArgs)

    ic1.EditingMode = InkCanvasEditingStyle.Select
    ic2.EditingMode = InkCanvasEditingStyle.Select

End Sub

Private Sub switchToInk(ByVal sender As Object, _
    ByVal e As RoutedEventArgs)

    ic1.EditingMode = InkCanvasEditingStyle.Ink
    ic2.EditingMode = InkCanvasEditingStyle.Ink

End Sub

End Class

```

Procedura: Eseguire l'associazione dati a un InkCanvas

23/10/2019 • 2 minutes to read • [Edit Online](#)

Esempio

Nell'esempio seguente viene illustrato come associare le [Strokes](#) proprietà di un [InkCanvas](#) a un altro [InkCanvas](#).

```
<InkCanvas Background="LightGray"
    Canvas.Top="0" Canvas.Left="0"
    Height="400" Width="200" Name="ic"/>

<!-- Bind the Strokes of the second InkCavas to the first InkCanvas
    and mirror the strokes along the Y axis.-->
<InkCanvas Background="LightBlue"
    Canvas.Top="0" Canvas.Left="200"
    Height="400" Width="200"
    Strokes="{Binding ElementName=ic, Path=Strokes}">
    <InkCanvas.LayoutTransform>
        <ScaleTransform ScaleX="-1" ScaleY="1" />
    </InkCanvas.LayoutTransform>
</InkCanvas>
```

Nell'esempio seguente viene illustrato come associare il [DefaultDrawingAttributes](#) proprietà a un'origine dati.

```
<Canvas.Resources>
    <!--Define an array containing some DrawingAttributes.-->
    <x:Array x:Key="MyDrawingAttributes" x:Type="{x:Type DrawingAttributes}">
        <DrawingAttributes Color="Black" FitToCurve="true" Width="3" Height="3"/>
        <DrawingAttributes Color="Blue" FitToCurve="false" Width="5" Height="5"/>
        <DrawingAttributes Color="Red" FitToCurve="true" Width="7" Height="7"/>
    </x:Array>

    <!--Create a DataTemplate to display the DrawingAttributes shown above-->
    <DataTemplate DataType="{x:Type DrawingAttributes}">
        <Border Width="80" Height="{Binding Path=Height}">
            <Border.Background>
                <SolidColorBrush Color="{Binding Path=Color}"/>
            </Border.Background>
        </Border>
    </DataTemplate>
</Canvas.Resources>
```

```

<!--Bind the InkCanvas' DefaultDrawingAtributes to
    a Listbox, called lbDrawingAttributes.-->
<InkCanvas Name="inkCanvas1" Background="LightGreen"
    Canvas.Top="400" Canvas.Left="0"
    Height="400" Width="400"
    DefaultDrawingAttributes="{Binding
        ElementName=lbDrawingAttributes, Path=SelectedItem}"
    >
</InkCanvas>

<!--Use the array, MyDrawingAttributes, to populate a ListBox-->
<ListBox Name="lbDrawingAttributes"
    Canvas.Top="400" Canvas.Left="450"
    Height="100" Width="100"
    ItemsSource="{StaticResource MyDrawingAttributes}" />

```

L'esempio seguente dichiara due [InkCanvas](#) degli oggetti in XAML e stabilisce l'associazione dati tra queste e altre origini dati. Il primo [InkCanvas](#), denominato [ic](#), è associato a due origini dati. Il [EditMode](#) e [DefaultDrawingAttributes](#) delle proprietà nel [ic](#) sono associati a [ListBox](#) oggetti, che a sua volta sono associati a matrici definite nel XAML. Il [EditMode](#), [DefaultDrawingAttributes](#), e [Strokes](#) delle proprietà del secondo [InkCanvas](#) sono associati al primo [InkCanvas](#), [ic](#).

```

<Canvas>
    <Canvas.Resources>
        <!--Define an array containing the InkEditingStyle Values.-->
        <x:Array x:Key="MyEditingModes" x:Type="{x:Type InkCanvasEditingStyle}">
            <x:Static Member="InkCanvasEditingStyle.Ink"/>
            <x:Static Member="InkCanvasEditingStyle.Select"/>
            <x:Static Member="InkCanvasEditingStyle.EraseByPoint"/>
            <x:Static Member="InkCanvasEditingStyle.EraseByStroke"/>
        </x:Array>

        <!--Define an array containing some DrawingAttributes.-->
        <x:Array x:Key="MyDrawingAttributes"
            x:Type="{x:Type DrawingAttributes}">
            <DrawingAttributes Color="Black" FitToCurve="true"
                Width="3" Height="3"/>
            <DrawingAttributes Color="Blue" FitToCurve="false"
                Width="5" Height="5"/>
            <DrawingAttributes Color="Red" FitToCurve="true"
                Width="7" Height="7"/>
        </x:Array>

        <!--Create a DataTemplate to display the
            DrawingAttributes shown above-->
        <DataTemplate DataType="{x:Type DrawingAttributes}" >
            <Border Width="80" Height="{Binding Path=Height}">
                <Border.Background>
                    <SolidColorBrush Color="{Binding Path=Color}"/>
                </Border.Background>
            </Border>
        </DataTemplate>
    </Canvas.Resources>

    <!--Bind the first InkCavas' DefaultDrawingAtributes to a
        Listbox, called lbDrawingAttributes, and its EditingMode to
        a ListBox called lbEditingStyle.-->
    <InkCanvas Name="ic" Background="LightGray"
        Canvas.Top="0" Canvas.Left="0"
        Height="400" Width="200"
        DefaultDrawingAttributes="{Binding
            ElementName=lbDrawingAttributes, Path=SelectedItem}"
        EditingMode=
            "{Binding ElementName=lbEditingStyle, Path=SelectedItem}">

```

```
</InkCanvas>

<!--Bind the Strokes, DefaultDrawingAttributes, and, EditingMode properties of
the second InkCavas the first InkCanvas.-->
<InkCanvas Background="LightBlue"
    Canvas.Top="0" Canvas.Left="200"
    Height="400" Width="200"
    Strokes="{Binding ElementName=ic, Path=Strokes}"
    DefaultDrawingAttributes="{Binding
        ElementName=ic, Path=DefaultDrawingAttributes}"
    EditingMode="{Binding ElementName=ic, Path=EditingMode}">

    <InkCanvas.LayoutTransform>
        <ScaleTransform ScaleX="-1" ScaleY="1" />
    </InkCanvas.LayoutTransform>

</InkCanvas>

<!--Use the array, MyEditingModes, to populate a ListBox-->
<ListBox Name="lbEditingMode"
    Canvas.Top="0" Canvas.Left="450"
    Height="100" Width="100"
    ItemsSource="{StaticResource MyEditingModes}" />

<!--Use the array, MyDrawingAttributes, to populate a ListBox-->
<ListBox Name="lbDrawingAttributes"
    Canvas.Top="150" Canvas.Left="450"
    Height="100" Width="100"
    ItemsSource="{StaticResource MyDrawingAttributes}" />

</Canvas>
```

Procedura: Analizzare l'input penna con i suggerimenti dell'analisi

23/10/2019 • 6 minutes to read • [Edit Online](#)

Un' [System.Windows.Ink.AnalysisHintNode](#) fornisce un suggerimento per il [InkAnalyzer](#) al quale è associato. Si applica l'hint per l'area specificata dal [System.Windows.Ink.ContextNode.Location%2A](#) proprietà delle [System.Windows.Ink.AnalysisHintNode](#) e fornisce un contesto aggiuntivo per l'utilità di analisi dell'input penna migliorare la precisione del riconoscimento. Il [InkAnalyzer](#) Applica queste informazioni di contesto quando l'analisi dell'input penna ottenuto dall'interno dell'area del suggerimento.

Esempio

L'esempio seguente è un'applicazione che usa più [System.Windows.Ink.AnalysisHintNode](#) gli oggetti in un form che accetta l'input penna. L'applicazione usa la [System.Windows.Ink.AnalysisHintNode.Factoid%2A](#) proprietà per fornire informazioni sul contesto per ogni voce nel form. L'applicazione usa l'analisi in background per analizzare l'input penna e cancella il formato dell'input penna di tutti i cinque secondi dopo che l'utente interrompe l'aggiunta dell'input penna.

```
<Window x:Class="FormAnalyzer"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="FormAnalyzer"
    SizeToContent="WidthAndHeight"
    >
    <StackPanel Orientation="Vertical">
        <InkCanvas Name="xaml_writingCanvas" Height="500" Width="840"
            StrokeCollected="RestartAnalysis" >
            <Grid>
                <Grid.Resources>
                    <Style TargetType="{x:Type Label}">
                        <Setter Property="FontSize" Value="20"/>
                        <Setter Property="FontFamily" Value="Arial"/>
                    </Style>

                    <Style TargetType="{x:Type TextBlock}">
                        <Setter Property="FontSize" Value="18"/>
                        <Setter Property="VerticalAlignment" Value="Center"/>
                    </Style>
                </Grid.Resources>

                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="100"/></ColumnDefinition>
                    <ColumnDefinition Width="160"/></ColumnDefinition>
                    <ColumnDefinition Width="160"/></ColumnDefinition>
                    <ColumnDefinition Width="100"/></ColumnDefinition>
                    <ColumnDefinition Width="160"/></ColumnDefinition>
                    <ColumnDefinition Width="160"/></ColumnDefinition>
                </Grid.ColumnDefinitions>

                <Grid.RowDefinitions>
                    <RowDefinition Height="100"/></RowDefinition>
                    <RowDefinition Height="100"/></RowDefinition>
                    <RowDefinition Height="100"/></RowDefinition>
                    <RowDefinition Height="100"/></RowDefinition>
                    <RowDefinition Height="100"/></RowDefinition>
                </Grid.RowDefinitions>
            </Grid>
        </InkCanvas>
    </StackPanel>
</Window>
```

```

<Label Grid.Row="0" Grid.Column="0">Title</Label>
<Label Grid.Row="1" Grid.Column="0">Director</Label>
<Label Grid.Row="2" Grid.Column="0">Starring</Label>
<Label Grid.Row="3" Grid.Column="0">Rating</Label>
<Label Grid.Row="3" Grid.Column="3">Year</Label>
<Label Grid.Row="4" Grid.Column="0">Genre</Label>

<TextBlock Name="xaml_blockTitle"
           Grid.Row="0" Grid.Column="1"
           Grid.ColumnSpan="5"/>
<TextBlock Name="xaml_blockDirector"
           Grid.Row="1" Grid.Column="1"
           Grid.ColumnSpan="5"/>
<TextBlock Name="xaml_blockStarring"
           Grid.Row="2" Grid.Column="1"
           Grid.ColumnSpan="5"/>
<TextBlock Name="xaml_blockRating"
           Grid.Row="3" Grid.Column="1"
           Grid.ColumnSpan="2"/>
<TextBlock Name="xaml_blockYear"
           Grid.Row="3" Grid.Column="4"
           Grid.ColumnSpan="2"/>
<TextBlock Name="xaml_blockGenre"
           Grid.Row="4" Grid.Column="1"
           Grid.ColumnSpan="5"/>

<Line Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="6"
      StrokeThickness="2" Stroke="Black"
      X1="0" Y1="100" X2="840" Y2="100" />
<Line Grid.Row="1" Grid.Column="0" Grid.ColumnSpan="6"
      StrokeThickness="2" Stroke="Black"
      X1="0" Y1="100" X2="840" Y2="100" />
<Line Grid.Row="2" Grid.Column="0" Grid.ColumnSpan="6"
      StrokeThickness="2" Stroke="Black"
      X1="0" Y1="100" X2="840" Y2="100" />
<Line Grid.Row="3" Grid.Column="0" Grid.ColumnSpan="6"
      StrokeThickness="2" Stroke="Black"
      X1="0" Y1="100" X2="840" Y2="100" />
<Line Grid.Row="3" Grid.Column="0" Grid.ColumnSpan="6"
      StrokeThickness="2" Stroke="Black"
      X1="420" Y1="0" X2="420" Y2="100" />
</Grid>
</InkCanvas>
</StackPanel>
</Window>

```

```

using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Ink;
using System.Windows.Threading;

public partial class FormAnalyzer : Window
{
    private InkAnalyzer analyzer;

    private AnalysisHintNode hintNodeTitle;
    private AnalysisHintNode hintNodeDirector;
    private AnalysisHintNode hintNodeStarring;
    private AnalysisHintNode hintNodeRating;
    private AnalysisHintNode hintNodeYear;
    private AnalysisHintNode hintNodeGenre;

    // Timer that raises an event to
    // clear the InkCanvas.
    private DispatcherTimer strokeRemovalTimer;

```

```

private const int CLEAR_STROKES_DELAY = 5;

public FormAnalyzer()
{
    InitializeComponent();
}

protected override void OnContentRendered(EventArgs e)
{
    base.OnContentRendered(e);

    // Initialize the Analyzer.
    analyzer = new InkAnalyzer();
    analyzer.ResultsUpdated += 
        new ResultsUpdatedEventHandler(analyzer_ResultsUpdated);

    // Add analysis hints for each form area.
    // Use the absolute Width and Height of the Grid's
    // RowDefinition and ColumnDefinition properties defined in XAML,
    // to calculate the bounds of the AnalysisHintNode objects.
    hintNodeTitle = analyzer.CreateAnalysisHint(
        new Rect(100, 0, 740, 100));
    hintNodeDirector = analyzer.CreateAnalysisHint(
        new Rect(100, 100, 740, 100));
    hintNodeStarring = analyzer.CreateAnalysisHint(
        new Rect(100, 200, 740, 100));
    hintNodeRating = analyzer.CreateAnalysisHint(
        new Rect(100, 300, 320, 100));
    hintNodeYear = analyzer.CreateAnalysisHint(
        new Rect(520, 300, 320, 100));
    hintNodeGenre = analyzer.CreateAnalysisHint(
        new Rect(100, 400, 740, 100));

    //Set the factoids on the hints.
    hintNodeTitleFactoid = "(!IS_DEFAULT)";
    hintNodeDirectorFactoid = "(!IS_PERSONALNAME_FULLNAME)";
    hintNodeStarringFactoid = "(!IS_PERSONALNAME_FULLNAME)";
    hintNodeRatingFactoid = "(!IS_DEFAULT)";
    hintNodeYearFactoid = "(!IS_DATE_YEAR)";
    hintNodeGenreFactoid = "(!IS_DEFAULT)";
}

/// <summary>
/// InkCanvas.StrokeCollected event handler. Begins
/// ink analysis and starts the timer to clear the strokes.
/// If five seconds pass without a Stroke being added,
/// the strokes on the InkCanvas will be cleared.
/// </summary>
/// <param am name="sender">InkCanvas that raises the
/// StrokeCollected event.</param>
/// <param name="args">Contains the event data.</param>
private void RestartAnalysis(object sender,
    InkCanvasStrokeCollectedEventArgs args)
{

    // If strokeRemovalTimer is enabled, stop it.
    if (strokeRemovalTimer != null && strokeRemovalTimer.IsEnabled)
    {
        strokeRemovalTimer.Stop();
    }

    // Restart the timer to clear the strokes in five seconds
    strokeRemovalTimer = new DispatcherTimer(
        TimeSpan.FromSeconds(CLEAR_STROKES_DELAY),
        DispatcherPriority.Normal,
        ClearCanvas,
        Dispatcher.CurrentDispatcher);

    // Add the new stroke to the InkAnalyzer and
}

```

```

// begin background analysis.
analyzer.AddStroke(args.Stroke);
analyzer.BackgroundAnalyze();
}

/// <summary>
/// Analyzer.ResultsUpdated event handler.
/// </summary>
/// <param name="sender">InkAnalyzer that raises the
/// event.</param>
/// <param name="e">Event data</param>
/// <remarks>This method checks each AnalysisHint for
/// analyzed ink and then populated the TextBlock that
/// corresponds to the area on the form.</remarks>
void analyzer_ResultsUpdated(object sender, ResultsUpdatedEventArgs e)
{
    string recoText;

    recoText = hintNodeTitle.GetRecognizedString();
    if (recoText != "") xaml_blockTitle.Text = recoText;

    recoText = hintNodeDirector.GetRecognizedString();
    if (recoText != "") xaml_blockDirector.Text = recoText;

    recoText = hintNodeStarring.GetRecognizedString();
    if (recoText != "") xaml_blockStarring.Text = recoText;

    recoText = hintNodeRating.GetRecognizedString();
    if (recoText != "") xaml_blockRating.Text = recoText;

    recoText = hintNodeYear.GetRecognizedString();
    if (recoText != "") xaml_blockYear.Text = recoText;

    recoText = hintNodeGenre.GetRecognizedString();
    if (recoText != "") xaml_blockGenre.Text = recoText;
}

//Clear the canvas, but leave the current strokes in the analyzer.
private void ClearCanvas(object sender, EventArgs args)
{
    strokeRemovalTimer.Stop();

    xaml_writingCanvas-strokes.Clear();
}
}

```

```

Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Ink
Imports System.Windows.Threading

Class FormAnalyzer
    Inherits Window

    Private analyzer As InkAnalyzer

    Private hintNodeTitle As AnalysisHintNode
    Private hintNodeDirector As AnalysisHintNode
    Private hintNodeStarring As AnalysisHintNode
    Private hintNodeRating As AnalysisHintNode
    Private hintNodeYear As AnalysisHintNode
    Private hintNodeGenre As AnalysisHintNode

    ' Timer that raises an event to
    ' clear the InkCanvas.

```

```

Private strokeRemovalTimer As DispatcherTimer

Private Const CLEAR_STROKES_DELAY As Integer = 5

Public Sub New()

    InitializeComponent()

End Sub

Protected Overrides Sub OnContentRendered(ByVal e As EventArgs)
    MyBase.OnContentRendered(e)

    ' Initialize the Analyzer.
    analyzer = New InkAnalyzer()
    AddHandler analyzer.ResultsUpdated, AddressOf analyzer_ResultsUpdated

    ' Add analysis hints for each form area.
    ' Use the absolute Width and Height of the Grid's
    ' RowDefinition and ColumnDefinition properties defined in XAML,
    ' to calculate the bounds of the AnalysisHintNode objects.
    hintNodeTitle = analyzer.CreateAnalysisHint(New Rect(100, 0, 740, 100))
    hintNodeDirector = analyzer.CreateAnalysisHint(New Rect(100, 100, 740, 100))
    hintNodeStarring = analyzer.CreateAnalysisHint(New Rect(100, 200, 740, 100))
    hintNodeRating = analyzer.CreateAnalysisHint(New Rect(100, 300, 320, 100))
    hintNodeYear = analyzer.CreateAnalysisHint(New Rect(520, 300, 320, 100))
    hintNodeGenre = analyzer.CreateAnalysisHint(New Rect(100, 400, 740, 100))

    'Set the factoids on the hints.
    hintNodeTitle.Factoid = "(!IS_DEFAULT)"
    hintNodeDirector.Factoid = "(!IS_PERSONALNAME_FULLNAME)"
    hintNodeStarring.Factoid = "(!IS_PERSONALNAME_FULLNAME)"
    hintNodeRating.Factoid = "(!IS_DEFAULT)"
    hintNodeYear.Factoid = "(!IS_DATE_YEAR)"
    hintNodeGenre.Factoid = "(!IS_DEFAULT)"

End Sub

' InkCanvas.StrokeCollected event handler. Begins
' ink analysis and starts the timer to clear the strokes.
' If five seconds pass without a Stroke being added,
' the strokes on the InkCanvas will be cleared.
' <param name="sender">InkCanvas that raises the
' StrokeCollected event.</param>
' <param name="args">Contains the event data.</param>
Private Sub RestartAnalysis(ByVal sender As Object, ByVal args As InkCanvasStrokeCollectedEventArgs)

    ' If strokeRemovalTimer is enabled, stop it.
    If Not (strokeRemovalTimer Is Nothing) AndAlso strokeRemovalTimer.IsEnabled Then
        strokeRemovalTimer.Stop()
    End If

    ' Restart the timer to clear the strokes in five seconds
    strokeRemovalTimer = New DispatcherTimer(
        TimeSpan.FromSeconds(CLEAR_STROKES_DELAY), _
        DispatcherPriority.Normal, _
        AddressOf ClearCanvas, _
        System.Windows.Threading.Dispatcher.CurrentDispatcher)

    ' Add the new stroke to the InkAnalyzer and
    ' begin background analysis.
    analyzer.AddStroke(args.Stroke)
    analyzer.BackgroundAnalyze()

End Sub

' Analyzer.ResultsUpdated event handler.
' <param name="sender">InkAnalyzer that raises the
' event.</param>

```

```

' <param name="e">Event data</param>
' <remarks>This method checks each AnalysisHint for
' analyzed ink and then populated the TextBlock that
' corresponds to the area on the form.</remarks>
Private Sub analyzer_ResultsUpdated(ByVal sender As Object, ByVal e As ResultsUpdatedEventArgs)

    Dim recoText As String

    recoText = hintNodeTitle.GetRecognizedString()
    If recoText <> "" Then
        xaml_blockTitle.Text = recoText
    End If

    recoText = hintNodeDirector.GetRecognizedString()
    If recoText <> "" Then
        xaml_blockDirector.Text = recoText
    End If

    recoText = hintNodeStarring.GetRecognizedString()
    If recoText <> "" Then
        xaml_blockStarring.Text = recoText
    End If

    recoText = hintNodeRating.GetRecognizedString()
    If recoText <> "" Then
        xaml_blockRating.Text = recoText
    End If

    recoText = hintNodeYear.GetRecognizedString()
    If recoText <> "" Then
        xaml_blockYear.Text = recoText
    End If

    recoText = hintNodeGenre.GetRecognizedString()
    If recoText <> "" Then
        xaml_blockGenre.Text = recoText
    End If

End Sub

'Clear the canvas, but leave the current strokes in the analyzer.
Private Sub ClearCanvas(ByVal sender As Object, ByVal args As EventArgs)

    strokeRemovalTimer.Stop()

    xaml_writingCanvas.Strokes.Clear()

End Sub
End Class

```

Procedura: Ruotare l'input penna

23/10/2019 • 7 minutes to read • [Edit Online](#)

Esempio

Nell'esempio seguente consente di copiare l'input penna da un [InkCanvas](#) a un [Canvas](#) che contiene un [InkPresenter](#). Quando l'applicazione input penna viene copiato, anche ruotato di 90 gradi in senso orario.

```
<Canvas>
    <InkCanvas Name="inkCanvas1" Background="LightBlue"
        Height="200" Width="200"
        Canvas.Top="20" Canvas.Left="20" />

    <Border Name="canvas1" Background="LightGreen"
        Height="200" Width="200" ClipToBounds="True"
        Canvas.Top="20" Canvas.Left="240" >
        <InkPresenter Name="inkPresenter1"/>
    </Border>
    <Button Click="button_Click"
        Canvas.Top="240" Canvas.Left="170">
        Copy and Rotate Strokes
    </Button>
</Canvas>
```

```
// Button.Click event handler that rotates the strokes
// and copies them to a Canvas.
private void button_Click(object sender, RoutedEventArgs e)
{
    StrokeCollection copiedStrokes = inkCanvas1.Strokes.Clone();
    Matrix rotatingMatrix = new Matrix();
    double canvasLeft = Canvas.GetLeft(inkCanvas1);
    double canvasTop = Canvas.GetTop(inkCanvas1);
    Point rotatePoint = new Point(canvas1.Width / 2, canvas1.Height / 2);

    rotatingMatrix.RotateAt(90, rotatePoint.X, rotatePoint.Y);
    copiedStrokes.Transform(rotatingMatrix, false);
    inkPresenter1.Strokes = copiedStrokes;
}
```

Esempio

L'esempio seguente è una classe personalizzata [Adorner](#) che consente di ruotare i tratti su un [InkPresenter](#).

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Controls.Primitives;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Shapes;
using System.Windows.Ink;

public class RotatingStrokesAdorner : Adorner
{
    // The Thumb to drag to rotate the strokes.
    Thumb rotateHandle;
```

```

// The surrounding boarder.
Path outline;

VisualCollection visualChildren;

// The center of the strokes.
Point center;
double lastAngle;

RotateTransform rotation;

const int HANDLEMARGIN = 10;

// The bounds of the Strokes;
Rect strokeBounds = Rect.Empty;

public RotatingStrokesAdorner(UIElement adornedElement)
    : base(adornedElement)
{

    visualChildren = new VisualCollection(this);
    rotateHandle = new Thumb();
    rotateHandle.Cursor = Cursors.SizeNWSE;
    rotateHandle.Width = 20;
    rotateHandle.Height = 20;
    rotateHandle.Background = Brushes.Blue;

    rotateHandle.DragDelta += new DragDeltaEventHandler(rotateHandle_DragDelta);
    rotateHandle.DragCompleted += new DragCompletedEventHandler(rotateHandle_DragCompleted);

    outline = new Path();
    outline.Stroke = Brushes.Blue;
    outline.StrokeThickness = 1;

    visualChildren.Add(outline);
    visualChildren.Add(rotateHandle);

    strokeBounds = AdornedStrokes.GetBounds();
}

/// <summary>
/// Draw the rotation handle and the outline of
/// the element.
/// </summary>
/// <param name="finalSize">The final area within the
/// parent that this element should use to arrange
/// itself and its children.</param>
/// <returns>The actual size used. </returns>
protected override Size ArrangeOverride(Size finalSize)
{
    if (strokeBounds.IsEmpty)
    {
        return finalSize;
    }

    center = new Point(strokeBounds.X + strokeBounds.Width / 2,
                      strokeBounds.Y + strokeBounds.Height / 2);

    // The rectangle that determines the position of the Thumb.
    Rect handleRect = new Rect(strokeBounds.X,
                               strokeBounds.Y - (strokeBounds.Height / 2 +
                                                 HANDLEMARGIN),
                               strokeBounds.Width, strokeBounds.Height);

    if (rotation != null)
    {
        handleRect.Transform(rotation.Value);
    }
}

```



```

    {
        if (rotation == null)
        {
            return;
        }

        // Rotate the strokes to match the new angle.
        Matrix mat = new Matrix();
        mat.RotateAt(rotation.Angle - lastAngle, center.X, center.Y);
        AdornedStrokes.Transform(mat, true);

        // Save the angle of the last rotation.
        lastAngle = rotation.Angle;

        // Redraw rotateHandle.
        this.InvalidateArrange();
    }

    /// <summary>
    /// Gets the strokes of the adorned element
    /// (in this case, an InkPresenter).
    /// </summary>
    private StrokeCollection AdornedStrokes
    {
        get
        {
            return ((InkPresenter)AdornedElement).Strokes;
        }
    }

    // Override the VisualChildrenCount and
    // GetVisualChild properties to interface with
    // the adorner's visual collection.
    protected override int VisualChildrenCount
    {
        get { return visualChildren.Count; }
    }

    protected override Visual GetVisualChild(int index)
    {
        return visualChildren[index];
    }
}

```

```

Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Controls.Primitives
Imports System.Windows.Documents
Imports System.Windows.Input
Imports System.Windows.Media
Imports System.Windows.Shapes
Imports System.Windows.Ink

Public Class RotatingStrokesAdorner
    Inherits Adorner

    ' The Thumb to drag to rotate the strokes.
    Private rotateHandle As Thumb

    ' The surrounding boarder.
    Private outline As Path

    Private visualChildren As VisualCollection

    ' The center of the strokes.

```

```

    THE CENTER OF THE STROKES.
Private center As Point
Private lastAngle As Double

Private rotation As RotateTransform

Private Const HANDLEMARGIN As Integer = 10

' The bounds of the Strokes;
Private strokeBounds As Rect = Rect.Empty


Public Sub New(ByVal adornedElement As UIElement)
 MyBase.New(adornedElement)

visualChildren = New VisualCollection(Me)
rotateHandle = New Thumb()
rotateHandle.Cursor = Cursors.SizeNWSE
rotateHandle.Width = 20
rotateHandle.Height = 20
rotateHandle.Background = Brushes.Blue

AddHandler rotateHandle.DragDelta, _
    AddressOf rotateHandle_DragDelta
AddHandler rotateHandle.DragCompleted, _
    AddressOf rotateHandle_DragCompleted

outline = New Path()
outline.Stroke = Brushes.Blue
outline.StrokeThickness = 1

visualChildren.Add(outline)
visualChildren.Add(rotateHandle)

strokeBounds = AdornedStrokes.GetBounds()

End Sub

''' <summary>
''' Draw the rotation handle and the outline of
''' the element.
''' </summary>
''' <param name="finalSize">The final area within the
''' parent that this element should use to arrange
''' itself and its children.</param>
''' <returns>The actual size used. </returns>
Protected Overrides Function ArrangeOverride(ByVal finalSize As Size) _
As Size

If strokeBounds.IsEmpty Then
    Return finalSize
End If

center = New Point(strokeBounds.X + strokeBounds.Width / 2, _
    strokeBounds.Y + strokeBounds.Height / 2)

' The rectangle that determines the position of the Thumb.
Dim handleRect As New Rect(strokeBounds.X, _
    strokeBounds.Y - (strokeBounds.Height / 2 + _
    HANDLEMARGIN), _
    strokeBounds.Width, strokeBounds.Height)

If Not (rotation Is Nothing) Then
    handleRect.Transform(rotation.Value)
End If

' Draws the thumb and the rectangle around the strokes.
rotateHandle.Arrange(handleRect)
outline.Data = New RectangleGeometry(strokeBounds)

```

```

outline.Data = New RectangleGeometry(strokeThickness)
outline.Arrange(New Rect(finalSize))
Return finalSize

End Function 'ArrangeOverride

''' <summary>
''' Rotates the rectangle representing the
''' strokes' bounds as the user drags the
''' Thumb.
''' </summary>
Private Sub rotateHandle_DragDelta(ByVal sender As Object, _
                                   ByVal e As DragDeltaEventArgs)

    'Find the angle of which to rotate the shape. Use the right
    'triangle that uses the center and the mouse's position
    'as vertices for the hypotenuse.
    Dim pos As Point = Mouse.GetPosition(Me)

    Dim deltaX As Double = pos.X - center.X
    Dim deltaY As Double = pos.Y - center.Y

    If deltaY.Equals(0) Then
        Return
    End If

    Dim tan As Double = deltaX / deltaY
    Dim angle As Double = Math.Atan(tan)

    ' Convert to degrees.
    angle = angle * 180 / Math.PI

    ' If the mouse crosses the vertical center,
    ' find the complementary angle.
    If deltaY > 0 Then
        angle = 180 - Math.Abs(angle)
    End If

    ' Rotate left if the mouse moves left and right
    ' if the mouse moves right.
    If deltaX < 0 Then
        angle = -Math.Abs(angle)
    Else
        angle = Math.Abs(angle)
    End If

    If Double.IsNaN(angle) Then
        Return
    End If

    ' Apply the rotation to the strokes' outline.
    rotation = New RotateTransform(angle, center.X, center.Y)
    outline.RenderTransform = rotation

End Sub

''' <summary>
''' Rotates the strokes to the same angle as outline.
''' </summary>
Private Sub rotateHandle_DragCompleted(ByVal sender As Object, _
                                       ByVal e As DragCompletedEventArgs)

    If rotation Is Nothing Then
        Return
    End If

    ' Rotate the strokes to match the new angle.
    ...

```

```

        Dim mat As New Matrix()
        mat.RotateAt(rotation.Angle - lastAngle, center.X, center.Y)
        AdornedStrokes.Transform(mat, True)

        ' Save the angle of the last rotation.
        lastAngle = rotation.Angle

        ' Redraw rotateHandle.
        Me.InvalidateArrange()

    End Sub

    ''' <summary>
    ''' Gets the strokes of the adorned element
    ''' (in this case, an InkPresenter).
    ''' </summary>
    Private ReadOnly Property AdornedStrokes() As StrokeCollection
        Get
            Return CType(AdornedElement, InkPresenter).Strokes
        End Get
    End Property

    ' Override the VisualChildrenCount and
    ' GetVisualChild properties to interface with
    ' the adorner's visual collection.

    Protected Overrides ReadOnly Property VisualChildrenCount() As Integer
        Get
            Return visualChildren.Count
        End Get
    End Property

    Protected Overrides Function GetVisualChild(ByVal index As Integer) As Visual
        Return visualChildren(index)
    End Function 'GetVisualChild
End Class

```

L'esempio seguente è un Extensible Application Markup Language (XAML) file che definisce un [InkPresenter](#) e popolarla con input penna. Il `Window_Loaded` gestore dell'evento aggiunge lo strumento decorativo personalizzato per il [InkPresenter](#).

```

<Window x:Class="Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Rotating Strokes Adorner" Height="500" Width="500"
    Loaded="Window_Loaded"
    >
    <InkPresenter Name="inkPresenter1" >
        <InkPresenter.Strokes>
            ALMDAwRIEEU1BQE4GSAyCQD0/wIB6SI6RTMJAPifAgFaIDpFOAgA/gMAAACAfxEAAIA/
            HwkJAAAAAAA8D8K1wE1h/CPd4SB4NA40icCjcGjcClcDj8Lh8DgUSkUmmU6nUmoUuk
            0ukUCQKVyehz+rzuly+bzORx+BReRQ+RTaRCH8JyXhPbgcPicPh8Pg80h0qk1SoVGrV
            Oo0mi0Xi8rm9Xr9Dqc/p87pc/k8XicHicOj1CoVktVmV1GqUaiUHlYg8e14akXK7m7T
            cSJgQggheym5zx6+PACk4dhPwg/fhCbxy8dp4p2tqnqxyvbP085z1X1aswhvCd94Tq
            55DRUGi4+Tk60Ln4KLko0ejo6ig5KTioOPCD9LlHmrzNxMRCcC3ec8+fe4AKQBmE/Cw
            9+FkPNvl0dkrYsWa+acp3Z8er0IT8JaX4S6+FbFilbHNvvPXNjbFqluxghKc5DkwrVF
            GEEIJ1w5eLKYAKShuF+Dnr40a8HVHXNPFFFFho8VFkqsMRYuuuvJxiF+F9r4Xx8HFiqs
            FNcirnweDw9+LvvvixdV0+GhONmlj3wjN0cSCEYTnfLy4oA
        </InkPresenter.Strokes>
    </InkPresenter>
</Window>

```

```
void Window_Loaded(object sender, RoutedEventArgs e)
{
    // Add the rotating strokes adorner to the InkPresenter.
    adornerLayer = AdornerLayer.GetAdornerLayer(inkPresenter1);
    adorner = new RotatingStrokesAdorner(inkPresenter1);

    adornerLayer.Add(adorner);
}
```

```
Private Sub Window_Loaded(ByVal sender As Object, ByVal e As RoutedEventArgs)

    ' Add the rotating strokes adorner to the InkPresenter.
    adornerLayer = adornerLayer.GetAdornerLayer(inkPresenter1)
    adorner = New RotatingStrokesAdorner(inkPresenter1)

    adornerLayer.Add(adorner)

End Sub
```

Disabilitare RealTimeStylus per le applicazioni WPF

29/01/2020 • 2 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) ha integrato il supporto per l'elaborazione dell'input tocco di Windows 7. Il supporto deriva dall'input dello stilo in tempo reale della piattaforma Tablet come [OnStylusDown](#), [OnStylusUp](#) e [OnStylusMove](#) eventi. Windows 7 fornisce inoltre input multitocco come messaggi della finestra WM_TOUCH Win32. Queste due API si escludono a vicenda nello stesso HWND. L'abilitazione dell'input tocco tramite la piattaforma Tablet (impostazione predefinita per le applicazioni WPF) Disabilita i messaggi di WM_TOUCH. Di conseguenza, per usare WM_TOUCH per ricevere messaggi di tocco da una finestra WPF, è necessario disabilitare il supporto dello stilo incorporato in WPF. Questa operazione è applicabile in uno scenario come una finestra WPF che ospita un componente che usa WM_TOUCH.

Per disabilitare l'ascolto dell'input dello stilo in WPF, rimuovere il supporto per Tablet aggiunto dalla finestra WPF.

Esempio

Nell'esempio di codice seguente viene illustrato come rimuovere il supporto predefinito della piattaforma Tablet utilizzando la reflection.

```
public static void DisableWPFTabletSupport()
{
    // Get a collection of the tablet devices for this window.
    TabletDeviceCollection devices = System.Windows.Input.Tablet.TabletDevices;

    if (devices.Count > 0)
    {
        // Get the Type of InputManager.
        Type inputManagerType = typeof(System.Windows.Input.InputManager);

        // Call the StylusLogic method on the InputManager.Current instance.
        object stylusLogic = inputManagerType.InvokeMember("StylusLogic",
            BindingFlags.GetProperty | BindingFlags.Instance | BindingFlags.NonPublic,
            null, InputManager.Current, null);

        if (stylusLogic != null)
        {
            // Get the type of the stylusLogic returned from the call to StylusLogic.
            Type stylusLogicType = stylusLogic.GetType();

            // Loop until there are no more devices to remove.
            while (devices.Count > 0)
            {
                // Remove the first tablet device in the devices collection.
                stylusLogicType.InvokeMember("OnTabletRemoved",
                    BindingFlags.InvokeMethod | BindingFlags.Instance | BindingFlags.NonPublic,
                    null, stylusLogic, new object[] { (uint)0 });
            }
        }
    }
}
```

Vedere anche

- [Intercettazione dell'input dello stilo](#)

Trascinamento e rilascio

23/10/2019 • 2 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) fornisce un'infrastruttura di trascinamento della selezione altamente flessibile che supporta il trascinamento e WPF l'eliminazione di dati all'interno di applicazioni, nonché di altre applicazioni Windows.

In questa sezione

[Cenni preliminari sul trascinamento della selezione](#)

[Dati e oggetti dati](#)

[Procedura dettagliata: Abilitazione del trascinamento della selezione in un controllo utente](#)

[Procedure relative alle proprietà](#)

Riferimenti

[DataFormat](#)

[DataObject](#)

[DragDrop](#)

[DragDropEffects](#)

[DragEventHandler](#)

[TextDataFormat](#)

Sezioni correlate

Cenni preliminari sul trascinamento della selezione

23/11/2019 • 31 minutes to read • [Edit Online](#)

Questo argomento fornisce una panoramica del supporto per il trascinamento della selezione nelle applicazioni Windows Presentation Foundation (WPF). Per trascinamento della selezione si intende di solito un metodo di trasferimento dei dati, in cui si usa un mouse (o un altro dispositivo di puntamento) per selezionare uno o più oggetti, si trascinano questi oggetti su un obiettivo di rilascio desiderato nell'interfaccia utente e li si rilascia.

Supporto del trascinamento della selezione in WPF

Nelle operazioni di trascinamento della selezione in genere sono coinvolte due parti: un'origine di trascinamento da cui ha origine l'oggetto trascinato e un obiettivo di rilascio che riceve l'oggetto rilasciato. L'origine di trascinamento e l'obiettivo di rilascio possono essere elementi dell'interfaccia utente della stessa applicazione o di un'applicazione diversa.

Il tipo e il numero di oggetti che possono essere manipolati con un trascinamento della selezione sono del tutto arbitrari. Ad esempio file, cartelle e selezioni di contenuto sono alcuni degli oggetti più comuni manipolati con operazioni di trascinamento della selezione.

Le particolari azioni eseguite durante un'operazione di trascinamento della selezione sono specifiche dell'applicazione e spesso dipendono dal contesto. Ad esempio, il trascinamento di una selezione di file da una cartella a un'altra nello stesso dispositivo di archiviazione consente di spostare i file per impostazione predefinita, mentre il trascinamento dei file da una condivisione Universal Naming Convention (UNC) a una cartella locale copia i file per impostazione predefinita.

Le funzionalità di trascinamento della selezione fornite da WPF sono estremamente flessibili e personalizzabili, per poter supportare svariati scenari di trascinamento. Il trascinamento della selezione supporta la manipolazione di oggetti in una sola applicazione o tra applicazioni diverse. Anche il trascinamento e la rimozione tra WPF applicazioni e altre applicazioni Windows sono completamente supportati.

In WPF, qualsiasi [UIElement](#) o [ContentElement](#) può partecipare al trascinamento della selezione. Gli eventi e i metodi necessari per le operazioni di trascinamento della selezione vengono definiti nella classe [DragDrop](#). Le classi [UIElement](#) e [ContentElement](#) contengono alias per gli eventi associati [DragDrop](#) in modo che gli eventi appaiano nell'elenco dei membri della classe quando un oggetto [UIElement](#) o [ContentElement](#) viene ereditato come elemento di base. I gestori eventi associati a questi eventi vengono associati all'evento associato [DragDrop](#) sottostante e ricevono la stessa istanza di dati dell'evento. Per altre informazioni, vedere l'evento [UIElement.Drop](#).

IMPORTANT

Il trascinamento della selezione OLE non funziona nell'area Internet.

Trasferimento dati

Il trascinamento della selezione rientra nell'ambito del trasferimento dei dati. Il trasferimento dei dati include operazioni di trascinamento della selezione e di copia e incolla. Un'operazione di trascinamento della selezione è analoga a un'operazione di copia e incolla o taglia e incolla usata per trasferire dati da un oggetto o un'applicazione a un'altra usando gli Appunti di sistema. Entrambi i tipi di operazioni richiedono:

- Un oggetto di origine che fornisce i dati.
- Un modo per archiviare temporaneamente i dati trasferiti.

- Un oggetto di destinazione che riceve i dati.

In un'operazione di copia e incolla, vengono usati gli Appunti di sistema per archiviare temporaneamente i dati trasferiti. In un'operazione di trascinamento della selezione, viene usato un oggetto [DataObject](#) per archiviare i dati. Concettualmente, un oggetto dati è costituito da una o più coppie di un [Object](#) contenente i dati effettivi e dal corrispondente identificatore di formato dati.

L'origine di trascinamento avvia un'operazione di trascinamento della selezione chiamando il metodo [DragDrop.DoDragDrop](#) statico e passandogli i dati trasferiti. Il metodo [DoDragDrop](#) eseguirà il wrapping automatico dei dati in [DataObject](#), se necessario. Per un maggiore controllo sul formato dei dati, è possibile eseguire il wrapping dei dati in un [DataObject](#) prima di passarli al metodo [DoDragDrop](#). L'obiettivo di rilascio è responsabile dell'estrazione dei dati dal [DataObject](#). Per altre informazioni sull'uso degli oggetti dati, vedere [Dati e oggetti dati](#).

L'origine e la destinazione di un'operazione di trascinamento della selezione sono elementi dell'interfaccia utente, ma i dati effettivamente trasferiti non hanno di solito una rappresentazione visiva. È possibile scrivere il codice per fornire una rappresentazione visiva dei dati trascinati, come accade quando si trascinano i file in Esplora risorse. Per impostazione predefinita, viene fornito un feedback all'utente modificando il cursore per rappresentare l'effetto che l'operazione di trascinamento della selezione avrà sui dati, ad esempio se i dati verranno spostati o copiati.

Effetti di trascinamento della selezione

Le operazioni di trascinamento della selezione possono avere effetti diversi sui dati trasferiti. Ad esempio, è possibile copiare i dati oppure spostarli. WPF definisce un'enumerazione [DragDropEffects](#) che è possibile usare per specificare l'effetto di un'operazione di trascinamento della selezione. Nell'origine di trascinamento, è possibile specificare gli effetti che verranno consentiti dall'origine nel metodo [DoDragDrop](#). Nell'obiettivo di rilascio, è possibile specificare l'effetto previsto per la destinazione nella proprietà [Effects](#) della classe [DragEventArgs](#). Quando l'obiettivo di rilascio specifica l'effetto previsto nell'evento [DragOver](#), queste informazioni vengono passate nuovamente all'origine di trascinamento nell'evento [GiveFeedback](#). L'origine di trascinamento usa queste informazioni per informare l'utente dell'effetto sui dati previsto per l'obiettivo di rilascio. Quando i dati vengono rilasciati, l'obiettivo di rilascio specifica il vero e proprio effetto nell'evento [Drop](#). Queste informazioni vengono passate all'origine di trascinamento come valore restituito del metodo [DoDragDrop](#). Se l'obiettivo di trascinamento restituisce un effetto non presente nell'elenco di origini di trascinamento di [allowedEffects](#), l'operazione di trascinamento della selezione viene annullata e il trasferimento dati non viene eseguito.

È importante ricordare che in WPF, i valori [DragDropEffects](#) vengono usati solo per fornire la comunicazione tra l'origine di trascinamento e l'obiettivo di rilascio in relazione agli effetti dell'operazione di trascinamento della selezione. L'effetto vero e proprio dell'operazione di trascinamento della selezione dipende dal codice scritto nell'applicazione.

L'obiettivo di rilascio, ad esempio, potrebbe specificare che l'effetto di rilascio dei dati è lo spostamento dei dati. Tuttavia, per spostare i dati, è necessario sia aggiungerli all'elemento di destinazione che rimuoverli dall'elemento di origine. L'elemento di origine potrebbe indicare che consente lo spostamento dei dati, ma, se non si specifica il codice per rimuovere i dati dall'elemento di origine, come risultato finale i dati verranno copiati e non spostati.

Eventi del trascinamento della selezione

Le operazioni di trascinamento della selezione supportano un modello basato sugli eventi. Sia l'origine di trascinamento che l'obiettivo di rilascio usano un set standard di eventi per gestire le operazioni di trascinamento della selezione. Le tabelle seguenti riepilogano gli eventi di trascinamento della selezione standard. Sono eventi associati nella [DragDrop](#) classe. Per altre informazioni sugli eventi associati, vedere [Cenni preliminari sugli eventi associati](#).

Eventi di origine di trascinamento

EVENT	RIEPILOGO
GiveFeedback	Questo evento si verifica continuamente durante un'operazione di trascinamento della selezione e consente all'origine di trascinamento di fornire informazioni all'utente. Per fornire questo feedback, in genere viene modificato l'aspetto del puntatore del mouse per indicare gli effetti consentiti dall'obiettivo di rilascio. Si tratta di un evento di bubbling.
QueryContinueDrag	Questo evento si verifica quando gli stati della tastiera o dei pulsanti del mouse cambiano durante un'operazione di trascinamento della selezione e consente all'origine di rilascio di annullare l'operazione di trascinamento a seconda degli stati di tastiera/pulsanti. Si tratta di un evento di bubbling.
PreviewGiveFeedback	Versione tunneling di GiveFeedback .
PreviewQueryContinueDrag	Versione tunneling di QueryContinueDrag .

Eventi dell'obiettivo di rilascio

EVENT	RIEPILOGO
DragEnter	Questo evento si verifica quando un oggetto viene trascinato entro il limite dell'obiettivo di rilascio. Si tratta di un evento di bubbling.
DragLeave	Questo evento si verifica quando un oggetto viene trascinato al di fuori del limite dell'obiettivo di rilascio. Si tratta di un evento di bubbling.
DragOver	Questo evento si verifica continuamente mentre un oggetto viene trascinato (spostato) entro il limite dell'obiettivo di rilascio. Si tratta di un evento di bubbling.
Drop	Questo evento si verifica quando un oggetto viene rilasciato sull'obiettivo di rilascio. Si tratta di un evento di bubbling.
PreviewDragEnter	Versione tunneling di DragEnter .
PreviewDragLeave	Versione tunneling di DragLeave .
PreviewDragOver	Versione tunneling di DragOver .
PreviewDrop	Versione tunneling di Drop .

Per gestire gli eventi di trascinamento della selezione per le istanze di un oggetto, aggiungere i gestori per gli eventi elencati nelle tabelle precedenti. Per gestire gli eventi di trascinamento della selezione a livello di classe, eseguire l'override dei corrispondenti metodi `On*Event` e `On*PreviewEvent` virtuali. Per altre informazioni, vedere [Gestione delle classi degli eventi indirizzati tramite classi di base dei controlli](#).

Implementazione del trascinamento della selezione

Un elemento dell'interfaccia utente può essere un'origine di trascinamento, un obiettivo di rilascio o entrambi. Per implementare il trascinamento della selezione di base, è necessario scrivere il codice per avviare l'operazione di

trascinamento e per elaborare i dati rilasciati. È possibile migliorare l'esperienza di trascinamento della selezione gestendo gli eventi di trascinamento facoltativi.

Per implementare il trascinamento della selezione di base, completare le seguenti attività:

- Identificare l'elemento che sarà un'origine di trascinamento. Un'origine di trascinamento può essere un oggetto [UIElement](#) o [ContentElement](#).
- Creare un gestore eventi nell'origine di trascinamento che avvierà l'operazione di trascinamento della selezione. L'evento è in genere [MouseMove](#).
- Nel gestore eventi dell'origine di trascinamento chiamare il metodo [DoDragDrop](#) per avviare l'operazione di trascinamento della selezione. Nella chiamata a [DoDragDrop](#) specificare l'origine di trascinamento, i dati da trasferire e gli effetti consentiti.
- Identificare l'elemento che sarà un obiettivo di rilascio. Un obiettivo di rilascio può essere un oggetto [UIElement](#) o [ContentElement](#).
- Nell'obiettivo di rilascio impostare la proprietà [AllowDrop](#) su `true`.
- Nell'obiettivo di rilascio creare un gestore dell'evento [Drop](#) per elaborare i dati rilasciati.
- Nel gestore dell'evento [Drop](#) estrarre i dati da [DragEventArgs](#) usando i metodi [GetDataPresent](#) e [GetData](#).
- Nel gestore dell'evento [Drop](#) usare i dati per eseguire l'operazione di trascinamento della selezione desiderata.

È possibile migliorare l'implementazione del trascinamento della selezione creando un [DataObject](#) personalizzato e gestendo eventi di origine di trascinamento e obiettivo di rilascio facoltativi, come mostrato nelle attività seguenti:

- Per trasferire dati personalizzati o più elementi di dati, creare un [DataObject](#) da passare al metodo [DoDragDrop](#).
- Per eseguire azioni aggiuntive durante un trascinamento, gestire gli eventi [DragEnter](#), [DragOver](#) e [DragLeave](#) nell'obiettivo di rilascio.
- Per modificare l'aspetto del puntatore del mouse, gestire l'evento [GiveFeedback](#) nell'origine di trascinamento.
- Per modificare la modalità con cui annullare l'operazione di trascinamento della selezione, gestire l'evento [QueryContinueDrag](#) nell'origine di trascinamento.

Esempi di trascinamento della selezione

Questa sezione descrive come implementare il trascinamento della selezione per un elemento [Ellipse](#). [Ellipse](#) è sia un'origine di trascinamento che un obiettivo di rilascio. I dati trasferiti sono la rappresentazione di stringa della proprietà [Fill](#) dell'ellisse. Il codice XAML seguente mostra l'elemento [Ellipse](#) e gli eventi relativi al trascinamento della selezione gestiti. Per i passaggi completi per implementare il trascinamento della selezione, vedere [Procedura dettagliata: abilitare il trascinamento della selezione in un controllo utente](#).

```
<Ellipse Height="50" Width="50" Fill="Green"
    MouseMove="ellipse_MouseMove"
    GiveFeedback="ellipse_GiveFeedback"
    AllowDrop="True"
    DragEnter="ellipse_DragEnter" DragLeave="ellipse_DragLeave"
    DragOver="ellipse_DragOver" Drop="ellipse_Drop" />
```

Impostazione di un elemento come origine di trascinamento

Un oggetto che è un'origine di trascinamento è responsabile di:

- Identificare quando si verifica un trascinamento.
- Avviare l'operazione di trascinamento della selezione.
- Identificare i dati da trasferire.
- Specificare gli effetti che l'operazione di trascinamento della selezione può avere sui dati trasferiti.

L'origine di trascinamento può anche fornire all'utente feedback relativo alle azioni consentite (spostamento, copia, nessuna) e può annullare l'operazione di trascinamento della selezione in base a un input utente aggiuntivo, ad esempio la pressione di ESC durante il trascinamento.

È responsabilità dell'applicazione determinare quando si verifica un trascinamento e quindi avviare l'operazione di trascinamento della selezione chiamando il metodo [DoDragDrop](#). Questo di solito avviene quando si verifica un evento [MouseMove](#) sull'elemento da trascinare mentre viene premuto un pulsante del mouse. Il seguente esempio mostra come avviare un'operazione di trascinamento e rilascio dal gestore dell'evento [MouseMove](#) di un elemento [Ellipse](#) per impostarlo come origine di trascinamento. I dati trasferiti sono la rappresentazione di stringa della proprietà [Fill](#) dell'ellisse.

```
private void ellipse_MouseMove(object sender, MouseEventArgs e)
{
    Ellipse ellipse = sender as Ellipse;
    if (ellipse != null && e.LeftButton == MouseButtons.Left)
    {
        DragDrop.DoDragDrop(ellipse,
                            ellipse.Fill.ToString(),
                            DragDropEffects.Copy);
    }
}
```

```
Private Sub Ellipse_MouseMove(ByVal sender As System.Object, ByVal e As System.Windows.Input.MouseEventArgs)
    Dim ellipse = TryCast(sender, Ellipse)
    If ellipse IsNot Nothing AndAlso e.LeftButton = MouseButtons.Left Then
        DragDrop.DoDragDrop(ellipse, ellipse.Fill.ToString(), DragDropEffects.Copy)
    End If
End Sub
```

Nel gestore dell'evento [MouseMove](#) chiamare il metodo [DoDragDrop](#) per avviare l'operazione di trascinamento della selezione. Il metodo [DoDragDrop](#) accetta tre parametri:

- `dragSource` : riferimento all'oggetto dipendenza che rappresenta l'origine dei dati trasferiti. si tratta in genere dell'origine dell'evento [MouseMove](#).
- `data` : oggetto contenente i dati trasferiti, di cui è stato eseguito il wrapper in un [DataObject](#).
- `allowedEffects` : uno dei [DragDropEffects](#) valori di enumerazione che specifica gli effetti consentiti dell'operazione di trascinamento della selezione.

Qualsiasi oggetto serializzabile può essere passato nel parametro `data`. Se il wrapping dei dati non è già stato eseguito in un [DataObject](#), verrà eseguito automaticamente in un nuovo [DataObject](#). Per passare più elementi di dati, è necessario creare il [DataObject](#) manualmente e passarlo al metodo [DoDragDrop](#). Per altre informazioni, vedere [Dati e oggetti dati](#).

Il parametro `allowedEffects` viene usato per specificare quali azioni l'origine di trascinamento consentirà di eseguire all'obiettivo di rilascio con i dati trasferiti. I valori comuni per un'origine di trascinamento sono [Copy](#), [Move](#) e [All](#).

NOTE

L'obiettivo di rilascio può anche specificare gli effetti previsti in risposta ai dati rilasciati. Ad esempio, se l'obiettivo di rilascio non riconosce il tipo di dati da rilasciare, può rifiutare i dati impostando gli effetti consentiti su `None`. In genere esegue questa operazione nel gestore dell'evento `DragOver`.

Un'origine di trascinamento può facoltativamente gestire gli eventi `GiveFeedback` e `QueryContinueDrag`. Questi eventi hanno gestori predefiniti che vengono usati solo se non si contrasseggano gli eventi come gestiti. In genere si ignorano questi eventi a meno che non esista un'esigenza specifica di modificarne il comportamento predefinito.

L'evento `GiveFeedback` viene generato continuamente mentre l'origine di trascinamento viene trascinata. Il gestore predefinito per questo evento controlla se l'origine di trascinamento sia su un obiettivo di rilascio valido. Se lo è, controlla gli effetti consentiti dell'obiettivo di rilascio. Fornisce quindi feedback all'utente finale sugli effetti di rilascio consentiti. A questo scopo il cursore del mouse viene in genere sostituito con un cursore di non rilascio, di copia o di spostamento. Si dovrebbe gestire questo evento solo se è necessario usare cursori personalizzati per fornire feedback all'utente. Se si gestisce questo evento, assicurarsi di contrassegnarlo come gestito in modo che il gestore predefinito non esegua l'override del gestore.

L'evento `QueryContinueDrag` viene generato continuamente mentre l'origine di trascinamento viene trascinata. È possibile gestire questo evento per determinare quale azione termina l'operazione di trascinamento e rilascio in base allo stato dei tasti ESC, MAIUSC, CTRL e ALT, oltre che allo stato dei pulsanti del mouse. Il gestore predefinito per questo evento annulla l'operazione di trascinamento della selezione se viene premuto ESC e rilascia i dati se il pulsante del mouse viene rilasciato.

Caution

Questi eventi vengono generati continuamente durante l'operazione di trascinamento della selezione. È quindi consigliabile evitare attività a elevato utilizzo di risorse nei gestori eventi. Ad esempio, usare un cursore nella cache invece di creare un nuovo cursore ogni volta che viene generato l'evento `GiveFeedback`.

Impostazione di un elemento come obiettivo di rilascio

Un oggetto che è un obiettivo di rilascio è responsabile di:

- Specificare che è un obiettivo di rilascio valido.
- Rispondere all'origine di trascinamento quando viene trascinata sulla destinazione.
- Controllare che i dati trasferiti siano in un formato che può ricevere.
- Elaborare i dati rilasciati.

Per specificare che un elemento è un obiettivo di rilascio, si imposta la proprietà `AllowDrop` su `true`. Gli eventi dell'obiettivo di rilascio verranno quindi generati nell'elemento per poterli gestire. Durante un'operazione di trascinamento della selezione, nell'obiettivo di rilascio si verifica la sequenza di eventi seguente:

1. `DragEnter`
2. `DragOver`
3. `DragLeave` o `Drop`

L'evento `DragEnter` si verifica quando i dati vengono trascinati entro il limite dell'obiettivo di rilascio. In genere si gestisce questo evento per fornire un'anteprima degli effetti dell'operazione di trascinamento e rilascio, se appropriato per l'applicazione. Non impostare la proprietà `DragEventArgs.Effects` nell'evento `DragEnter`, perché verrà sovrascritta nell'evento `DragOver`.

Il seguente esempio mostra il gestore dell'evento `DragEnter` per un elemento `Ellipse`. Questo codice visualizza in anteprima gli effetti dell'operazione di trascinamento della selezione salvando il pennello `Fill` corrente. Usa quindi

il metodo [GetDataPresent](#) per controllare se il [DataObject](#) che viene trascinato sull'ellisse contiene dati stringa convertibili in un [Brush](#). In tal caso, i dati vengono estratti con il metodo [GetData](#). Vengono quindi convertiti in un [Brush](#) e applicati all'ellisse. La modifica viene ripristinata nel gestore dell'evento [DragLeave](#). Se i dati non possono essere convertiti in un [Brush](#), non viene eseguita alcuna azione.

```
private Brush _previousFill = null;
private void ellipse_DragEnter(object sender, DragEventArgs e)
{
    Ellipse ellipse = sender as Ellipse;
    if (ellipse != null)
    {
        // Save the current Fill brush so that you can revert back to this value in DragLeave.
        _previousFill = ellipse.Fill;

        // If the DataObject contains string data, extract it.
        if (e.Data.GetDataPresent(DataFormats.StringFormat))
        {
            string dataString = (string)e.Data.GetData(DataFormats.StringFormat);

            // If the string can be converted into a Brush, convert it.
            BrushConverter converter = new BrushConverter();
            if (converter.IsValid(dataString))
            {
                Brush newFill = (Brush)converter.ConvertFromString(dataString);
                ellipse.Fill = newFill;
            }
        }
    }
}
```

```
Private _previousFill As Brush = Nothing
Private Sub Ellipse_DragEnter(ByVal sender As System.Object, ByVal e As System.Windows.DragEventArgs)
    Dim ellipse = TryCast(sender, Ellipse)
    If ellipse IsNot Nothing Then
        ' Save the current Fill brush so that you can revert back to this value in DragLeave.
        _previousFill = ellipse.Fill

        ' If the DataObject contains string data, extract it.
        If e.Data.GetDataPresent(DataFormats.StringFormat) Then
            Dim dataString = e.Data.GetData(DataFormats.StringFormat)

            ' If the string can be converted into a Brush, convert it.
            Dim converter As New BrushConverter()
            If converter.IsValid(dataString) Then
                Dim newFill As Brush = CType(converter.ConvertFromString(dataString), Brush)
                ellipse.Fill = newFill
            End If
        End If
    End If
End Sub
```

L'evento [DragOver](#) si verifica in modo continuo mentre i dati vengono trascinati sull'obiettivo di rilascio. Questo evento è associato all'evento [GiveFeedback](#) nell'origine di trascinamento. Nel gestore dell'evento [DragOver](#), si usano di solito i metodi [GetDataPresent](#) e [GetData](#) per controllare se i dati trasferiti sono in un formato che l'obiettivo di rilascio può elaborare. È anche possibile controllare se vengono premuti tasti di modifica, che in genere indicano se l'utente intende eseguire un'azione di copia o spostamento. Dopo questi controlli, si imposta la proprietà [DragEventArgs.Effects](#) per notificare all'origine di trascinamento quale effetto avrà il rilascio dei dati. L'origine di trascinamento riceve queste informazioni negli argomenti dell'evento [GiveFeedback](#) e può impostare un cursore appropriato per fornire feedback all'utente.

Il seguente esempio mostra il gestore dell'evento [DragOver](#) per un elemento [Ellipse](#). Questo codice verifica se il

[DataObject](#) che viene trascinato sull'ellisse contiene dati stringa convertibili in un [Brush](#). In tal caso, imposta la proprietà [DragEventArgs.Effects](#) su [Copy](#). Questo indica all'origine di trascinamento che i dati possono essere copiati nell'ellisse. Se i dati non possono essere convertiti in un [Brush](#), la proprietà [DragEventArgs.Effects](#) viene impostata su [None](#). Questo indica all'origine di trascinamento che l'ellisse non è un obiettivo valido per i dati.

```
private void ellipse_DragOver(object sender, DragEventArgs e)
{
    e.Effects = DragDropEffects.None;

    // If the DataObject contains string data, extract it.
    if (e.Data.GetDataPresent(DataFormats.StringFormat))
    {
        string dataString = (string)e.Data.GetData(DataFormats.StringFormat);

        // If the string can be converted into a Brush, allow copying.
        BrushConverter converter = new BrushConverter();
        if (converter.IsValid(dataString))
        {
            e.Effects = DragDropEffects.Copy | DragDropEffects.Move;
        }
    }
}
```

```
Private Sub Ellipse_DragOver(ByVal sender As System.Object, ByVal e As System.Windows.DragEventArgs)
    e.Effects = DragDropEffects.None

    ' If the DataObject contains string data, extract it.
    If e.Data.GetDataPresent(DataFormats.StringFormat) Then
        Dim dataString = e.Data.GetData(DataFormats.StringFormat)

        ' If the string can be converted into a Brush, convert it.
        Dim converter As New BrushConverter()
        If converter.IsValid(dataString) Then
            e.Effects = DragDropEffects.Copy Or DragDropEffects.Move
        End If
    End If
End Sub
```

L'evento [DragLeave](#) si verifica quando i dati vengono trascinati all'esterno del limite della destinazione senza essere rilasciati. Si gestisce questo evento per annullare qualsiasi operazione eseguita nel gestore dell'evento [DragEnter](#).

Il seguente esempio mostra il gestore dell'evento [DragLeave](#) per un elemento [Ellipse](#). Questo codice annulla l'anteprima eseguita nel gestore dell'evento [DragEnter](#) applicando l'oggetto [Brush](#) salvato all'ellisse.

```
private void ellipse_DragLeave(object sender, DragEventArgs e)
{
    Ellipse ellipse = sender as Ellipse;
    if (ellipse != null)
    {
        ellipse.Fill = _previousFill;
    }
}
```

```

Private Sub Ellipse_DragLeave(ByVal sender As System.Object, ByVal e As System.Windows.DragEventArgs)
    Dim ellipse = TryCast(sender, Ellipse)
    If ellipse IsNot Nothing Then
        ellipse.Fill = _previousFill
    End If
End Sub

```

L'evento **Drop** si verifica quando i dati vengono rilasciati sull'obiettivo di rilascio. Per impostazione predefinita, questo avviene quando viene rilasciato il pulsante del mouse. Nel gestore dell'evento **Drop**, si usa il metodo **GetData** per estrarre i dati trasferiti dal **DataObject** ed eseguire l'elaborazione dati richiesta dall'applicazione. L'evento **Drop** termina l'operazione di trascinamento della selezione.

Il seguente esempio mostra il gestore dell'evento **Drop** per un elemento **Ellipse**. Questo codice applica gli effetti dell'operazione di trascinamento della selezione ed è simile al codice nel gestore dell'evento **DragEnter**. Verifica se il **DataObject** che viene trascinato sull'ellisse contiene dati stringa convertibili in un **Brush**. In tal caso, **Brush** viene applicato all'ellisse. Se i dati non possono essere convertiti in un **Brush**, non viene eseguita alcuna azione.

```

private void ellipse_Drop(object sender, DragEventArgs e)
{
    Ellipse ellipse = sender as Ellipse;
    if (ellipse != null)
    {
        // If the DataObject contains string data, extract it.
        if (e.Data.GetDataPresent(DataFormats.StringFormat))
        {
            string dataString = (string)e.Data.GetData(DataFormats.StringFormat);

            // If the string can be converted into a Brush,
            // convert it and apply it to the ellipse.
            BrushConverter converter = new BrushConverter();
            if (converter.IsValid(dataString))
            {
                Brush newFill = (Brush)converter.ConvertFromString(dataString);
                ellipse.Fill = newFill;
            }
        }
    }
}

```

```

Private Sub Ellipse_Drop(ByVal sender As System.Object, ByVal e As System.Windows.DragEventArgs)
    Dim ellipse = TryCast(sender, Ellipse)
    If ellipse IsNot Nothing Then

        ' If the DataObject contains string data, extract it.
        If e.Data.GetDataPresent(DataFormats.StringFormat) Then
            Dim dataString = e.Data.GetData(DataFormats.StringFormat)

            ' If the string can be converted into a Brush, convert it.
            Dim converter As New BrushConverter()
            If converter.IsValid(dataString) Then
                Dim newFill As Brush = CType(converter.ConvertFromString(dataString), Brush)
                ellipse.Fill = newFill
            End If
        End If
    End Sub

```

Vedere anche

- [Clipboard](#)

- Procedura dettagliata: Abilitare il trascinamento della selezione in un controllo utente
- Procedure relative
- Trascinamento della selezione

Dati e oggetti dati

23/10/2019 • 13 minutes to read • [Edit Online](#)

I dati trasferiti come parte di un'operazione di trascinamento della selezione vengono archiviati in un oggetto dati. Concettualmente, un oggetto dati è costituito da una o più delle coppie seguenti:

- Oggetto [Object](#) che contiene i dati effettivi.
- Identificatore del formato dati corrispondente.

I dati possono essere costituiti da qualsiasi elemento che può essere rappresentato come [Objectbase](#). Il formato dei dati corrispondente è una stringa [Type](#) o che fornisce un suggerimento sul formato in cui si trovano i dati. Gli oggetti dati supportano l'hosting di più coppie di formato dati/dati; Ciò consente a un singolo oggetto dati di fornire dati in più formati.

Oggetti dati

Tutti gli oggetti dati devono implementare [IDataObject](#) l'interfaccia, che fornisce il set di metodi standard seguente che consentono e facilitano il trasferimento dei dati.

METODO	RIEPILOGO
GetData	Recupera un oggetto dati in un formato dati specificato.
GetDataPresent	Verifica se i dati sono disponibili in o possono essere convertiti in un formato specificato.
GetFormats	Restituisce un elenco di formati in cui i dati in questo oggetto dati sono archiviati o possono essere convertiti in.
SetData	Archivia i dati specificati in questo oggetto dati.

WPF fornisce un'implementazione di base [IDataObject](#) di [DataObject](#) nella classe. La classe [DataObject](#) Stock è sufficiente per molti scenari comuni di trasferimento dei dati.

Sono disponibili diversi formati predefiniti, ad esempio bitmap, CSV, file, HTML, RTF, stringhe, testo e audio. Per informazioni sui formati di dati predefiniti forniti con WPF, vedere l'argomento di riferimento alla [DataFormats](#) classe.

Gli oggetti dati includono in genere una funzionalità per la conversione automatica dei dati archiviati in un formato diverso, durante l'estrazione dei dati; Questa funzionalità viene definita conversione automatica. Quando si eseguono query per i formati di dati disponibili in un oggetto dati, è possibile filtrare i formati di dati convertibili automaticamente dai formati di [GetFormats\(Boolean\)](#) dati [GetDataPresent\(String, Boolean\)](#) nativi chiamando il metodo `autoConvert` o e `false` specificando il parametro come. Quando si aggiungono dati a un oggetto dati [SetData\(String, Object, Boolean\)](#) con il metodo, la conversione automatica dei dati può essere proibita `autoConvert` impostando `false` il parametro su.

Utilizzo di oggetti dati

In questa sezione vengono descritte le tecniche comuni per la creazione e l'utilizzo di oggetti dati.

Creazione di nuovi oggetti dati

La [DataObject](#) classe fornisce diversi costruttori di overload che facilitano il popolamento di una [DataObject](#) nuova istanza con una singola coppia dati/formato dati.

Nell'esempio di codice seguente viene creato un nuovo oggetto dati e viene utilizzato uno dei costruttori [DataObject](#)di overload ([DataObject\(String, Object\)](#)) per inizializzare l'oggetto dati con una stringa e un formato dati specificato. In questo caso, il formato dati è specificato da una stringa. la [DataFormats](#) classe fornisce un set di stringhe di tipo predefinite. La conversione automatica dei dati archiviati è consentita per impostazione predefinita.

```
string stringData = "Some string data to store...";  
string dataFormat = DataFormats.UnicodeText;  
DataObject dataObject = new DataObject(dataFormat, stringData);
```

```
Dim stringData As String = "Some string data to store..."  
Dim dataFormat As String = DataFormats.UnicodeText  
Dim dataObject As New DataObject(dataFormat, stringData)
```

Per altri esempi di codice per la creazione di un oggetto dati, vedere [creare un oggetto dati](#).

Archiviazione dei dati in più formati

Un singolo oggetto dati è in grado di archiviare i dati in più formati. L'uso strategico di più formati di dati all'interno di un singolo oggetto dati potenzialmente rende l'oggetto dati utilizzabile da una varietà più ampia di destinazioni di rilascio rispetto a quando è possibile rappresentare solo un singolo formato di dati. Si noti che, in generale, un'origine di trascinamento deve essere indipendente dai formati di dati che possono essere utilizzati da potenziali destinazioni di rilascio.

Nell'esempio seguente viene illustrato come utilizzare il [SetData\(String, Object\)](#) metodo per aggiungere dati a un oggetto dati in più formati.

```
DataObject dataObject = new DataObject();  
string sourceData = "Some string data to store...";  
  
// Encode the source string into Unicode byte arrays.  
byte[] unicodeText = Encoding.Unicode.GetBytes(sourceData); // UTF-16  
byte[] utf8Text = Encoding.UTF8.GetBytes(sourceData);  
byte[] utf32Text = Encoding.UTF32.GetBytes(sourceData);  
  
// The DataFormats class does not provide data format fields for denoting  
// UTF-32 and UTF-8, which are seldom used in practice; the following strings  
// will be used to identify these "custom" data formats.  
string utf32DateFormat = "UTF-32";  
string utf8DateFormat = "UTF-8";  
  
// Store the text in the data object, letting the data object choose  
// the data format (which will be DataFormats.Text in this case).  
dataObject.SetData(sourceData);  
// Store the Unicode text in the data object. Text data can be automatically  
// converted to Unicode (UTF-16 / UCS-2) format on extraction from the data object;  
// Therefore, explicitly converting the source text to Unicode is generally unnecessary, and  
// is done here as an exercise only.  
dataObject.SetData(DataFormats.UnicodeText, unicodeText);  
// Store the UTF-8 text in the data object...  
dataObject.SetData(utf8DateFormat, utf8Text);  
// Store the UTF-32 text in the data object...  
dataObject.SetData(utf32DateFormat, utf32Text);
```

```

Dim dataObject As New DataObject()
Dim sourceData As String = "Some string data to store..."

' Encode the source string into Unicode byte arrays.
Dim unicodeText() As Byte = Encoding.Unicode.GetBytes(sourceData) ' UTF-16
Dim utf8Text() As Byte = Encoding.UTF8.GetBytes(sourceData)
Dim utf32Text() As Byte = Encoding.UTF32.GetBytes(sourceData)

' The DataFormats class does not provide data format fields for denoting
' UTF-32 and UTF-8, which are seldom used in practice; the following strings
' will be used to identify these "custom" data formats.
Dim utf32DateFormat As String = "UTF-32"
Dim utf8DateFormat As String = "UTF-8"

' Store the text in the data object, letting the data object choose
' the data format (which will be DataFormats.Text in this case).
dataObject.SetData(sourceData)

' Store the Unicode text in the data object. Text data can be automatically
' converted to Unicode (UTF-16 / UCS-2) format on extraction from the data object;
' Therefore, explicitly converting the source text to Unicode is generally unnecessary, and
' is done here as an exercise only.
dataObject.SetData(DataFormats.UnicodeText, unicodeText)
' Store the UTF-8 text in the data object...
dataObject.SetData(utf8DateFormat, utf8Text)
' Store the UTF-32 text in the data object...
dataObject.SetData(utf32DateFormat, utf32Text)

```

Esecuzione di query su un oggetto dati per i formati disponibili

Poiché un singolo oggetto dati può contenere un numero arbitrario di formati di dati, gli oggetti dati includono le funzionalità per il recupero di un elenco di formati di dati disponibili.

Il codice di esempio seguente usa [GetFormats](#) l'overload per ottenere una matrice di stringhe che indica tutti i formati di dati disponibili in un oggetto dati (sia nativi che tramite conversione automatica).

```

DataObject dataObject = new DataObject("Some string data to store...");

// Get an array of strings, each string denoting a data format
// that is available in the data object. This overload of GetDataFormats
// returns all available data formats, native and auto-convertible.
string[] dataFormats = dataObject.GetFormats();

// Get the number of data formats present in the data object, including both
// auto-convertible and native data formats.
int numberOfDataFormats = dataFormats.Length;

// To enumerate the resulting array of data formats, and take some action when
// a particular data format is found, use a code structure similar to the following.
foreach (string dataFormat in dataFormats)
{
    if (dataFormat == DataFormats.Text)
    {
        // Take some action if/when data in the Text data format is found.
        break;
    }
    else if(dataFormat == DataFormats.StringFormat)
    {
        // Take some action if/when data in the string data format is found.
        break;
    }
}

```

```

Dim dataObject As New DataObject("Some string data to store...")

' Get an array of strings, each string denoting a data format
' that is available in the data object. This overload of GetDataFormats
' returns all available data formats, native and auto-convertible.
Dim dataFormats() As String = dataObject.GetFormats()

' Get the number of data formats present in the data object, including both
' auto-convertible and native data formats.
Dim numberOfDataFormats As Integer = dataFormats.Length

' To enumerate the resulting array of data formats, and take some action when
' a particular data format is found, use a code structure similar to the following.
For Each dataFormat As String In dataFormats
    If dataFormat = System.Windows.DataFormats.Text Then
        ' Take some action if/when data in the Text data format is found.
        Exit For
    ElseIf dataFormat = System.Windows.DataFormats.StringFormat Then
        ' Take some action if/when data in the string data format is found.
        Exit For
    End If
Next dataFormat

```

Per altri esempi di codice che esegue query su un oggetto dati per i formati di dati disponibili, vedere [elencare i formati di dati in un oggetto dati](#). Per esempi di esecuzione di query su un oggetto dati per la presenza di un particolare formato dati, vedere [determinare se un formato di dati è presente in un oggetto dati](#).

Recupero di dati da un oggetto dati

Il recupero di dati da un oggetto dati in un particolare formato prevede semplicemente la chiamata a [GetData](#) uno dei metodi e la specifica del formato dati desiderato. Uno dei [GetDataPresent](#) metodi può essere utilizzato per verificare la presenza di un particolare formato dati. [GetData](#) Restituisce i dati in un [Object](#) oggetto; a seconda del formato dati, è possibile eseguire il cast di questo oggetto a un contenitore specifico del tipo.

Nell'esempio di codice seguente viene [GetDataPresent\(String\)](#) utilizzato l'overload per verificare se è disponibile un formato dati specificato (nativo o tramite conversione automatica). Se il formato specificato è disponibile, nell'esempio vengono recuperati i dati utilizzando [GetData\(String\)](#) il metodo.

```

DataObject dataObject = new DataObject("Some string data to store...");

string desiredFormat = DataFormats.UnicodeText;
byte[] data = null;

// Use the GetDataPresent method to check for the presence of a desired data format.
// This particular overload of GetDataPresent looks for both native and auto-convertible
// data formats.
if (dataObject.GetDataPresent(desiredFormat))
{
    // If the desired data format is present, use one of the GetData methods to retrieve the
    // data from the data object.
    data = dataObject.GetData(desiredFormat) as byte[];
}

```

```
Dim dataObject As New DataObject("Some string data to store...")  
  
Dim desiredFormat As String = DataFormats.UnicodeText  
Dim data() As Byte = Nothing  
  
' Use the GetDataPresent method to check for the presence of a desired data format.  
' This particular overload of GetDataPresent looks for both native and auto-convertible  
' data formats.  
If dataObject.GetDataPresent(desiredFormat) Then  
    ' If the desired data format is present, use one of the GetData methods to retrieve the  
    ' data from the data object.  
    data = TryCast(dataObject.GetData(desiredFormat), Byte())  
End If
```

Per altri esempi di codice che recupera dati da un oggetto dati, vedere [recuperare dati in un particolare formato dati](#).

Rimozione di dati da un oggetto dati

I dati non possono essere rimossi direttamente da un oggetto dati. Per rimuovere efficacemente i dati da un oggetto dati, attenersi alla procedura seguente:

1. Creare un nuovo oggetto dati che conterrà solo i dati che si desidera mantenere.
2. "Copiare" i dati desiderati dall'oggetto dati precedente al nuovo oggetto dati. Per copiare i dati, utilizzare uno dei [GetData](#) metodi per recuperare un [Object](#) oggetto che contiene i dati non elaborati, [SetData](#) quindi utilizzare uno dei metodi per aggiungere i dati al nuovo oggetto dati.
3. Sostituire l'oggetto dati precedente con quello nuovo.

NOTE

I [SetData](#) metodi aggiungono solo dati a un oggetto dati, ma non sostituiscono i dati, anche se i dati e il formato dati sono esattamente uguali a quelli di una chiamata precedente. La [SetData](#) chiamata di due volte per gli stessi dati e il formato dati comporterà la presenza del formato dati/dati due volte nell'oggetto dati.

Procedura dettagliata: Abilitare il trascinamento della selezione in un controllo utente

23/10/2019 • 27 minutes to read • [Edit Online](#)

Questa procedura dettagliata illustra come creare un controllo utente personalizzato che possa partecipare al trasferimento di dati tramite trascinamento in Windows Presentation Foundation (WPF).

In questa procedura dettagliata verrà creato un oggetto WPF [UserControl](#) personalizzato che rappresenta una forma circolare. Si implemetteranno funzionalità nel controllo per consentire il trasferimento di dati tramite trascinamento. Ad esempio, se si trascina da un controllo Circle a un altro, i dati del colore di riempimento vengono copiati dal controllo Circle di origine a quello di destinazione. Se si trascina da un controllo Circle a un [TextBox](#) oggetto, la rappresentazione di stringa del colore di riempimento viene copiata [TextBox](#) in. Si creerà anche una piccola applicazione che contiene due controlli Panel e un oggetto [TextBox](#) per testare la funzionalità di trascinamento della selezione. Si scriverà codice che consente ai pannelli di elaborare i dati Circle rilasciati per permettere di spostare o copiare cerchi dalla raccolta Children di un pannello all'altro.

Questa procedura dettagliata illustra le attività seguenti:

- Creare un controllo utente personalizzato.
- Consentire al controllo utente di essere un'origine di trascinamento.
- Consentire al controllo utente di essere un obiettivo di rilascio.
- Consentire a un pannello di ricevere dati rilasciati dal controllo utente.

Prerequisiti

Per completare la procedura dettagliata, è necessario Visual Studio.

Creare il progetto dell'applicazione

In questa sezione verrà creata l'infrastruttura dell'applicazione, che include una pagina principale con due pannelli e un oggetto [TextBox](#).

1. Creare un nuovo progetto di applicazione WPF in Visual Basic o Visual C# denominato `DragDropExample`.
Per altre informazioni, vedere [Procedura dettagliata: Prima applicazione desktop WPF](#).
2. Aprire `MainWindow.xaml`.
3. Aggiungere il markup seguente tra i tag di apertura [Grid](#) e di chiusura.

Questo markup crea l'interfaccia utente per l'applicazione di prova.

```
<Grid.ColumnDefinitions>
    <ColumnDefinition />
    <ColumnDefinition />
</Grid.ColumnDefinitions>
<StackPanel Grid.Column="0"
    Background="Beige">
    <TextBox Width="Auto" Margin="2"
        Text="green"/>
</StackPanel>
<StackPanel Grid.Column="1"
    Background="Bisque">
</StackPanel>
```

Aggiungere un nuovo controllo utente al progetto

In questa sezione si aggiungerà un nuovo controllo utente al progetto.

1. Scegliere **Aggiungi controllo utente** dal menu Progetto.
2. Nella finestra di dialogo **Aggiungi nuovo elemento** modificare il nome in `Circle.xaml`, quindi fare clic su **Aggiungi**.

Circle.xaml e il relativo code-behind vengono aggiunti al progetto.

3. Aprire Circle.xaml.

Questo file conterrà gli elementi dell'interfaccia utente del controllo utente.

4. Aggiungere il markup seguente alla radice **Grid** per creare un controllo utente semplice con un cerchio blu come interfaccia utente.

```
<Ellipse x:Name="circleUI"
    Height="100" Width="100"
    Fill="Blue" />
```

5. Aprire Circle.xaml.cs o Circle.xaml.vb.

6. In C#aggiungere il codice seguente dopo il costruttore senza parametri per creare un costruttore di copia. In Visual Basic aggiungere il codice seguente per creare un costruttore senza parametri e un costruttore di copia.

Per consentire la copia del controllo utente, aggiungere un metodo di costruttore di copia nel file code-behind. Nel controllo utente Circle semplificato si copieranno solo il riempimento e le dimensioni del controllo utente.

```
public Circle(Circle c)
{
    InitializeComponent();
    this.circleUI.Height = c.circleUI.Height;
    this.circleUI.Width = c.circleUI.Width;
    this.circleUI.Fill = c.circleUI.Fill;
}
```

```

Public Sub New()
    ' This call is required by the designer.
    InitializeComponent()
End Sub

Public Sub New(ByVal c As Circle)
    InitializeComponent()
    Me.circleUI.Height = c.circleUI.Height
    Me.circleUI.Width = c.circleUI.Height
    Me.circleUI.Fill = c.circleUI.Fill
End Sub

```

Aggiungere il controllo utente alla finestra principale

1. Aprire MainWindow.xaml.
2. Aggiungere il codice XAML seguente al tag [Window](#) di apertura per creare un riferimento allo spazio dei nomi XML all'applicazione corrente.

```
xmlns:local="clr-namespace:DragDropExample"
```

3. Nel primo [StackPanel](#), aggiungere il codice XAML seguente per creare due istanze del controllo utente Circle nel primo pannello.

```
<local:Circle Margin="2" />
<local:Circle Margin="2" />
```

Il codice XAML completo per il pannello è analogo al seguente.

```

<StackPanel Grid.Column="0"
            Background="Beige">
    <TextBox Width="Auto" Margin="2"
             Text="green"/>
    <local:Circle Margin="2" />
    <local:Circle Margin="2" />
</StackPanel>
<StackPanel Grid.Column="1"
            Background="Bisque">
</StackPanel>

```

Implementare gli eventi di origine del trascinamento nel controllo utente

In questa sezione verrà eseguito l'override del [OnMouseMove](#) metodo e verrà avviata l'operazione di trascinamento della selezione.

Se viene avviato un trascinamento (viene premuto un pulsante del mouse e il mouse viene spostato), i dati verranno inseriti in un pacchetto [DataObject](#) da trasferire in un oggetto. In questo caso, il controllo Circle assemblerà tre elementi di dati: una rappresentazione di stringa del colore di riempimento, una rappresentazione doppia dell'altezza e una copia di se stesso.

Per avviare un'operazione di riempimento

1. Aprire Circle.xaml.cs o Circle.xaml.vb.
2. Aggiungere la seguente [OnMouseMove](#) sostituzione per fornire la gestione delle classi [MouseMove](#) per

l'evento.

```
protected override void OnMouseMove(MouseEventArgs e)
{
    base.OnMouseMove(e);
    if (e.LeftButton == MouseButtonState.Pressed)
    {
        // Package the data.
        DataObject data = new DataObject();
        data.SetData(DataFormats.StringFormat, circleUI.Fill.ToString());
        data.SetData("Double", circleUI.Height);
        data.SetData("Object", this);

        // Initiate the drag-and-drop operation.
        DragDrop.DoDragDrop(this, data, DragDropEffects.Copy | DragDropEffects.Move);
    }
}
```

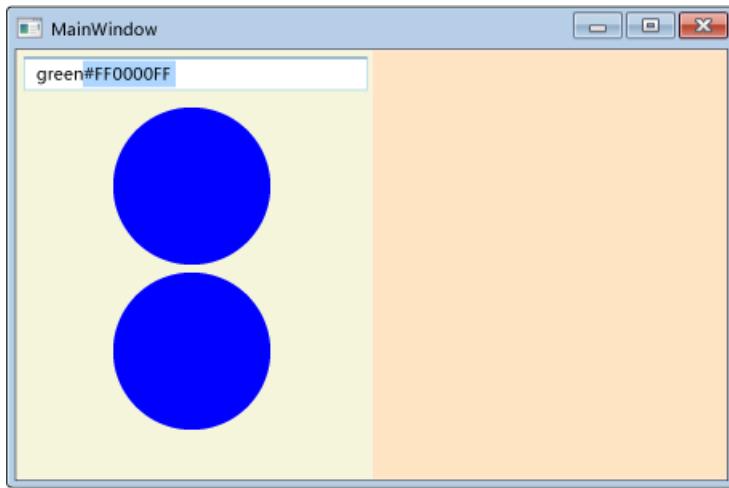
```
Protected Overrides Sub OnMouseMove(ByVal e As System.Windows.Input.MouseEventArgs)
    MyBase.OnMouseMove(e)
    If e.LeftButton = MouseButtonState.Pressed Then
        ' Package the data.
        Dim data As New DataObject
        data.SetData(DataFormats.StringFormat, circleUI.Fill.ToString())
        data.SetData("Double", circleUI.Height)
        data.SetData("Object", Me)

        ' Initiate the drag-and-drop operation.
        DragDrop.DoDragDrop(Me, data, DragDropEffects.Copy Or DragDropEffects.Move)
    End If
End Sub
```

Questo [OnMouseMove](#) override esegue le attività seguenti:

- Controlla se il pulsante sinistro del mouse viene premuto mentre il mouse è in movimento.
- Inserisce i dati Circle in un [DataObject](#)oggetto. In questo caso, il controllo Circle assembla tre elementi dati: una rappresentazione di stringa del colore di riempimento, una rappresentazione doppia dell'altezza e una copia di se stesso.
- Chiama il metodo [DragDrop.DoDragDrop](#) statico per avviare l'operazione di trascinamento della selezione. Passare i tre parametri seguenti al [DoDragDrop](#) metodo:
 - `dragSource` - Un riferimento a questo controllo.
 - `data` : Oggetto [DataObject](#) creato nel codice precedente.
 - `allowedEffects` : Le operazioni di trascinamento della selezione consentite [Copy](#) , [Move](#)ovvero o.

3. Premere **F5** per compilare ed eseguire l'applicazione.
4. Fare clic su uno dei controlli Circle e trascinarlo sui pannelli, sull'altro cerchio e [TextBox](#)su. Quando si trascina su [TextBox](#), il cursore cambia per indicare uno spostamento.
5. Quando si trascina un cerchio sul controllo [TextBox](#), premere il tasto **CTRL** . Notare come il cursore cambia aspetto per indicare una copia.
6. Trascinare e rilasciare un cerchio nell'oggetto [TextBox](#). La rappresentazione di stringa del colore di riempimento del cerchio viene aggiunta all'oggetto [TextBox](#).



Per impostazione predefinita, il cursore cambierà aspetto durante un'operazione di trascinamento per indicare l'effetto che avrà il rilascio dei dati. È possibile personalizzare il feedback fornito all'utente gestendo l'[GiveFeedback](#) evento e impostando un cursore diverso.

Inviare commenti e suggerimenti all'utente

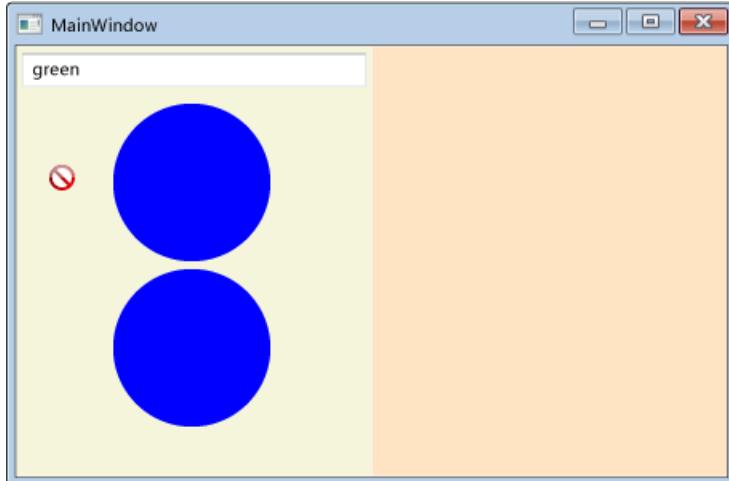
1. Aprire Circle.xaml.cs o Circle.xaml.vb.
2. Aggiungere la seguente [OnGiveFeedback](#) sostituzione per fornire la gestione delle classi [GiveFeedback](#) per l'evento.

```
protected override void OnGiveFeedback(GiveFeedbackEventArgs e)
{
    base.OnGiveFeedback(e);
    // These Effects values are set in the drop target's
    // DragOver event handler.
    if (e.Effects.HasFlag(DragDropEffects.Copy))
    {
        Mouse.SetCursor(Cursors.Cross);
    }
    else if (e.Effects.HasFlag(DragDropEffects.Move))
    {
        Mouse.SetCursor(Cursors.Pen);
    }
    else
    {
        Mouse.SetCursor(Cursors.No);
    }
    e.Handled = true;
}
```

```
Protected Overrides Sub OnGiveFeedback(ByVal e As System.Windows.GiveFeedbackEventArgs)
    MyBase.OnGiveFeedback(e)
    ' These Effects values are set in the drop target's
    ' DragOver event handler.
    If e.Effects.HasFlag(DragDropEffects.Copy) Then
        Mouse.SetCursor(Cursors.Cross)
    ElseIf e.Effects.HasFlag(DragDropEffects.Move) Then
        Mouse.SetCursor(Cursors.Pen)
    Else
        Mouse.SetCursor(Cursors.No)
    End If
    e.Handled = True
End Sub
```

Questo [OnGiveFeedback](#) override esegue le attività seguenti:

- Verifica i **Effects** valori impostati nel gestore eventi dell'obiettivo di **DragOver** rilascio.
 - Imposta un cursore personalizzato in base al **Effects** valore. Il cursore ha lo scopo di fornire un feedback visivo all'utente sull'effetto che avrà il rilascio dei dati.
3. Premere **F5** per compilare ed eseguire l'applicazione.
4. Trascinare uno dei controlli Circle sui pannelli, sull'altro cerchio e sull'oggetto **TextBox**. Si noti che i cursori sono ora i cursori personalizzati specificati nella **OnGiveFeedback** sostituzione.



5. Selezionare il testo `green` **TextBox**.
6. Trascinare il testo `green` su un controllo Circle. Notare che vengono visualizzati i cursori predefiniti per indicare gli effetti dell'operazione di trascinamento. Il cursore di feedback viene sempre impostato dall'origine di trascinamento.

Implementare eventi di destinazione di rilascio nel controllo utente

In questa sezione si specificherà che il controllo utente è un obiettivo di rilascio, si eseguirà l'override dei metodi che consentono al controllo utente di essere un obiettivo di rilascio e si elaboreranno i dati rilasciati sul controllo.

Per consentire al controllo utente di essere un obiettivo di rilascio

1. Aprire **Circle.xaml**.
2. Nel tag di **UserControl** apertura aggiungere la **AllowDrop** proprietà e impostarla su `true`.

```
<UserControl x:Class="DragDropWalkthrough.Circle"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="300"
    AllowDrop="True">
```

Il **OnDrop** metodo viene chiamato quando la **AllowDrop** proprietà è impostata `true` su e i dati dell'origine di trascinamento vengono rilasciati sul controllo utente **Circle**. In questo metodo si elaboreranno i dati rilasciati e li si applicheranno al controllo **Circle**.

Per elaborare i dati rilasciati

1. Aprire **Circle.xaml.cs** o **Circle.xaml.vb**.
2. Aggiungere la seguente **OnDrop** sostituzione per fornire la gestione delle classi **Drop** per l'evento.

```

protected override void OnDrop(DragEventArgs e)
{
    base.OnDrop(e);

    // If the DataObject contains string data, extract it.
    if (e.Data.GetDataPresent(DataFormats.StringFormat))
    {
        string dataString = (string)e.Data.GetData(DataFormats.StringFormat);

        // If the string can be converted into a Brush,
        // convert it and apply it to the ellipse.
        BrushConverter converter = new BrushConverter();
        if (converter.IsValid(dataString))
        {
            Brush newFill = (Brush)converter.ConvertFromString(dataString);
            circleUI.Fill = newFill;

            // Set Effects to notify the drag source what effect
            // the drag-and-drop operation had.
            // (Copy if CTRL is pressed; otherwise, move.)
            if (e.KeyStates.HasFlag(DragDropKeyStates.ControlKey))
            {
                e.Effects = DragDropEffects.Copy;
            }
            else
            {
                e.Effects = DragDropEffects.Move;
            }
        }
    }
    e.Handled = true;
}

```

```

Protected Overrides Sub OnDrop(ByVal e As System.Windows.DragEventArgs)
    MyBase.OnDrop(e)
    ' If the DataObject contains string data, extract it.
    If e.Data.GetDataPresent(DataFormats.StringFormat) Then
        Dim dataString As String = e.Data.GetData(DataFormats.StringFormat)

        ' If the string can be converted into a Brush,
        ' convert it and apply it to the ellipse.
        Dim converter As New BrushConverter
        If converter.IsValid(dataString) Then
            Dim newFill As Brush = converter.ConvertFromString(dataString)
            circleUI.Fill = newFill

            ' Set Effects to notify the drag source what effect
            ' the drag-and-drop operation had.
            ' (Copy if CTRL is pressed; otherwise, move.)
            If e.KeyStates.HasFlag(DragDropKeyStates.ControlKey) Then
                e.Effects = DragDropEffects.Copy
            Else
                e.Effects = DragDropEffects.Move
            End If
        End If
        e.Handled = True
    End Sub

```

Questo [OnDrop](#) override esegue le attività seguenti:

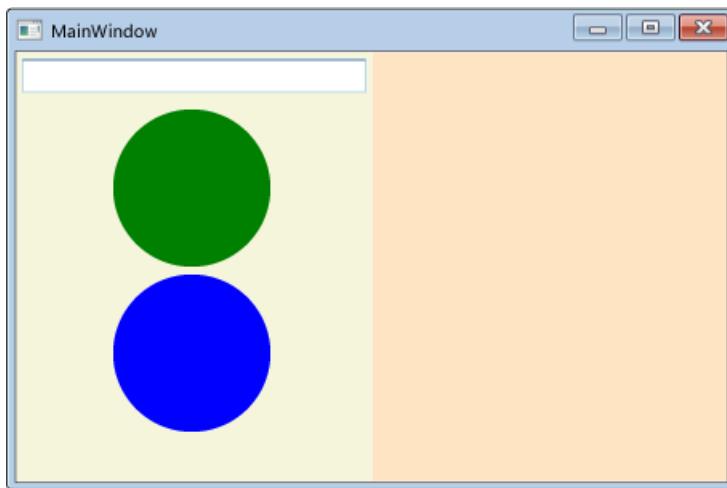
- Usa il [GetDataPresent](#) metodo per verificare se i dati trascinati contengono un oggetto String.
- Usa il [GetData](#) metodo per estrarre i dati di stringa, se presenti.

- Usa un [BrushConverter](#) oggetto per tentare di convertire la stringa in [Brush](#)un oggetto.
- Se la conversione ha esito positivo, applica il pennello [Fill](#) all'oggetto [Ellipse](#) dell'oggetto che fornisce l'interfaccia utente del controllo Circle.
- Contrassegna l' [Drop](#) evento come gestito. È necessario contrassegnare l'evento di trascinamento come gestito per segnalare agli altri elementi che ricevono l'evento che questo è stato gestito dal controllo utente Circle.

3. Premere **F5** per compilare ed eseguire l'applicazione.

4. Selezionare il testo `green` [TextBox](#)in.

5. Trascinare il testo su un controllo Circle e rilasciarlo. Il controllo Circle passa da blu a verde.



6. Digitare il testo `green` [TextBox](#)in.

7. Selezionare il testo `gre` [TextBox](#)in.

8. Trascinarlo su un controllo Circle e rilasciarlo. Notare che il cursore cambia aspetto per indicare che è consentito il trascinamento ma il colore del controllo Circle non cambia poiché `gre` non è un colore valido.

9. Trascinare dal controllo Circle verde e rilasciare sul controllo Circle blu. Il controllo Circle passa da blu a verde. Si noti che il cursore viene visualizzato a seconda che il [TextBox](#) o il cerchio sia l'origine del trascinamento.

L'impostazione [AllowDrop](#) della proprietà `true` su e l'elaborazione dei dati eliminati è tutto ciò che è necessario per consentire a un elemento di essere un obiettivo di rilascio. Tuttavia, per offrire un'esperienza utente migliore, è necessario gestire anche gli [DragEnter](#) eventi [DragLeave](#), e [DragOver](#). In questi eventi, è possibile eseguire controlli e offrire feedback all'utente prima che i dati vengono rilasciati.

Quando i dati vengono rilasciati sul controllo utente Circle, il controllo deve notificare all'origine di trascinamento se può elaborare i dati trascinati. Se il controllo non è in grado di elaborare i dati, deve rifiutare il rilascio. A tale scopo, si gestirà l' [DragOver](#) evento e si imposterà la [Effects](#) proprietà.

Per verificare se il rilascio dei dati è consentito

1. Aprire Circle.xaml.cs o Circle.xaml.vb.

2. Aggiungere la seguente [OnDragOver](#) sostituzione per fornire la gestione delle classi [DragOver](#) per l'evento.

```

protected override void OnDragOver(DragEventArgs e)
{
    base.OnDragOver(e);
    e.Effects = DragDropEffects.None;

    // If the DataObject contains string data, extract it.
    if (e.Data.GetDataPresent(DataFormats.StringFormat))
    {
        string dataString = (string)e.Data.GetData(DataFormats.StringFormat);

        // If the string can be converted into a Brush, allow copying or moving.
        BrushConverter converter = new BrushConverter();
        if (converter.IsValid(dataString))
        {
            // Set Effects to notify the drag source what effect
            // the drag-and-drop operation will have. These values are
            // used by the drag source's GiveFeedback event handler.
            // (Copy if CTRL is pressed; otherwise, move.)
            if (e.KeyStates.HasFlag(DragDropKeyStates.ControlKey))
            {
                e.Effects = DragDropEffects.Copy;
            }
            else
            {
                e.Effects = DragDropEffects.Move;
            }
        }
    }
    e.Handled = true;
}

```

```

Protected Overrides Sub OnDragOver(ByVal e As System.Windows.DragEventArgs)
    MyBase.OnDragOver(e)
    e.Effects = DragDropEffects.None

    ' If the DataObject contains string data, extract it.
    If e.Data.GetDataPresent(DataFormats.StringFormat) Then
        Dim dataString As String = e.Data.GetData(DataFormats.StringFormat)

        ' If the string can be converted into a Brush, allow copying or moving.
        Dim converter As New BrushConverter
        If converter.IsValid(dataString) Then
            ' Set Effects to notify the drag source what effect
            ' the drag-and-drop operation will have. These values are
            ' used by the drag source's GiveFeedback event handler.
            ' (Copy if CTRL is pressed; otherwise, move.)
            If e.KeyStates.HasFlag(DragDropKeyStates.ControlKey) Then
                e.Effects = DragDropEffects.Copy
            Else
                e.Effects = DragDropEffects.Move
            End If
        End If
    End If
    e.Handled = True
End Sub

```

Questo [OnDragOver](#) override esegue le attività seguenti:

- Imposta la proprietà [Effects](#) su [None](#).
- Esegue gli stessi controlli che vengono eseguiti nel [OnDrop](#) metodo per determinare se il controllo utente Circle può elaborare i dati trascinati.
- Se il controllo utente può elaborare i dati, imposta la [Effects](#) proprietà su [Copy](#) o [Move](#).

3. Premere **F5** per compilare ed eseguire l'applicazione.
4. Selezionare il testo `gre` `TextBox`in.
5. Trascinare il testo su un controllo Circle. Notare che ora il cursore cambia aspetto per indicare che il trascinamento non è consentito poiché `gre` non è un colore valido.

È possibile migliorare ulteriormente l'esperienza utente applicando un'anteprima dell'operazione di trascinamento. Per il controllo utente Circle, si eseguirà l' `OnDragEnter` override `OnDragLeave` dei metodi e. Quando i dati vengono trascinati sul controllo, lo sfondo `Fill` corrente viene salvato in una variabile segnaposto. La stringa viene quindi convertita in un pennello e applicata all' `Ellipse` oggetto che fornisce l'interfaccia utente del cerchio. Se i dati vengono trascinati fuori dal cerchio senza essere eliminati, il valore `Fill` originale viene nuovamente applicato al cerchio.

Per visualizzare in anteprima gli effetti dell'operazione di trascinamento

1. Aprire Circle.xaml.cs o Circle.xaml.vb.
2. Nella classe Circle dichiarare una variabile privata `Brush` denominata `_previousFill` e inizializzarla su `null`.

```
public partial class Circle : UserControl
{
    private Brush _previousFill = null;
```

```
Public Class Circle
    Private _previousFill As Brush = Nothing
```

3. Aggiungere la seguente `OnDragEnter` sostituzione per fornire la gestione delle classi `DragEnter` per l'evento.

```
protected override void OnDragEnter(DragEventArgs e)
{
    base.OnDragEnter(e);
    // Save the current Fill brush so that you can revert back to this value in DragLeave.
    _previousFill = circleUI.Fill;

    // If the DataObject contains string data, extract it.
    if (e.Data.GetDataPresent(DataFormats.StringFormat))
    {
        string dataString = (string)e.Data.GetData(DataFormats.StringFormat);

        // If the string can be converted into a Brush, convert it.
        BrushConverter converter = new BrushConverter();
        if (converter.IsValid(dataString))
        {
            Brush newFill = (Brush)converter.ConvertFromString(dataString.ToString());
            circleUI.Fill = newFill;
        }
    }
}
```

```

Protected Overrides Sub OnDragEnter(ByVal e As System.Windows.DragEventArgs)
    MyBase.OnDragEnter(e)
    _previousFill = circleUI.Fill

    ' If the DataObject contains string data, extract it.
    If e.Data.GetDataPresent(DataFormats.StringFormat) Then
        Dim dataString As String = e.Data.GetData(DataFormats.StringFormat)

        ' If the string can be converted into a Brush, convert it.
        Dim converter As New BrushConverter
        If converter.IsValid(dataString) Then
            Dim newFill As Brush = converter.ConvertFromString(dataString)
            circleUI.Fill = newFill
        End If
    End If
    e.Handled = True
End Sub

```

Questo [OnDragEnter](#) override esegue le attività seguenti:

- Salva la [Fill](#) proprietà dell'oggetto [Ellipse](#) nella [_previousFill](#) variabile.
- Esegue gli stessi controlli che vengono eseguiti nel [OnDrop](#) metodo per determinare se i dati possono essere convertiti in un oggetto [Brush](#)
- Se i dati vengono convertiti in un [Brush](#)oggetto valido, lo applica [Fill](#) a del [Ellipse](#).

4. Aggiungere la seguente [OnDragLeave](#) sostituzione per fornire la gestione delle classi [DragLeave](#) per l'evento.

```

protected override void OnDragLeave(DragEventArgs e)
{
    base.OnDragLeave(e);
    // Undo the preview that was applied in OnDragEnter.
    circleUI.Fill = _previousFill;
}

```

```

Protected Overrides Sub OnDragLeave(ByVal e As System.Windows.DragEventArgs)
    MyBase.OnDragLeave(e)
    ' Undo the preview that was applied in OnDragEnter.
    circleUI.Fill = _previousFill
End Sub

```

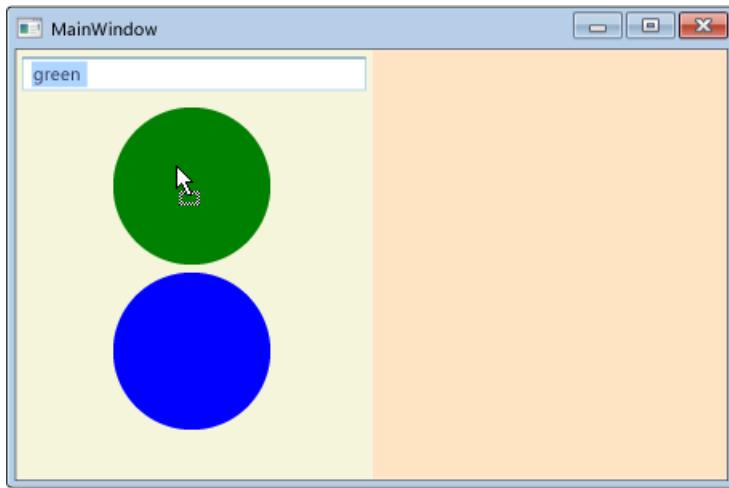
Questo [OnDragLeave](#) override esegue le attività seguenti:

- Applica l' [Brush](#) oggetto salvato [_previousFill](#) nella variabile all'oggetto [Fill](#) dell'oggetto [Ellipse](#) che fornisce l'interfaccia utente del controllo utente Circle.

5. Premere **F5** per compilare ed eseguire l'applicazione.

6. Selezionare il testo [green](#) [TextBoxin](#).

7. Trascinare il testo su un controllo Circle senza rilasciarlo. Il controllo Circle passa da blu a verde.



8. Trascinare il testo dal controllo Circle. Il controllo Circle passa da verde a blu.

Abilitare un pannello per la ricezione dei dati eliminati

In questa sezione si abilitano i pannelli che ospitano i controlli utente Circle in modo che fungano da destinazioni di rilascio per i dati del cerchio trascinato. Si implementerà il codice che consente di spostare un cerchio da un pannello a un altro o di creare una copia di un controllo Circle tenendo premuto il tasto **CTRL** mentre si trascina e si elimina un cerchio.

1. Aprire MainWindow.xaml.
2. Come illustrato nel codice XAML seguente, in ognuno dei **StackPanel** controlli aggiungere i gestori per gli **DragOver** eventi e **Drop**. Assegnare un **DragOver** nome al **panel_DragOver** gestore eventi,, e **Drop** denominare **panel_Drop** il gestore dell'evento.,

```
<StackPanel Grid.Column="0"
            Background="Beige"
            AllowDrop="True"
            DragOver="panel_DragOver"
            Drop="panel_Drop">
    <TextBox Width="Auto" Margin="2"
             Text="green"/>
    <local:Circle Margin="2" />
    <local:Circle Margin="2" />
</StackPanel>
<StackPanel Grid.Column="1"
            Background="Bisque"
            AllowDrop="True"
            DragOver="panel_DragOver"
            Drop="panel_Drop">
</StackPanel>
```

3. Aprire MainWindows.xaml.cs o MainWindow.xaml.vb.
4. Aggiungere il codice seguente per il **DragOver** gestore eventi.

```

private void panel_DragOver(object sender, DragEventArgs e)
{
    if (e.Data.GetDataPresent("Object"))
    {
        // These Effects values are used in the drag source's
        // GiveFeedback event handler to determine which cursor to display.
        if (e.KeyStates == DragDropKeyStates.ControlKey)
        {
            e.Effects = DragDropEffects.Copy;
        }
        else
        {
            e.Effects = DragDropEffects.Move;
        }
    }
}

```

```

Private Sub panel_DragOver(ByVal sender As System.Object, ByVal e As System.Windows.DragEventArgs)
If e.Data.GetDataPresent("Object") Then
    ' These Effects values are used in the drag source's
    ' GiveFeedback event handler to determine which cursor to display.
    If e.KeyStates = DragDropKeyStates.ControlKey Then
        e.Effects = DragDropEffects.Copy
    Else
        e.Effects = DragDropEffects.Move
    End If
End If
End Sub

```

Questo [DragOver](#) gestore eventi esegue le attività seguenti:

- Verifica che i dati trascinati contengano i dati "Object" inclusi nel [DataObject](#) pacchetto dal controllo utente Circle e passati nella chiamata a. [DoDragDrop](#)
- Se i dati dell'oggetto sono presenti, controlla se il tasto **CTRL** è premuto.
- Se viene premuto il tasto **CTRL**, imposta la [Effects](#) proprietà su [Copy](#). In caso contrario, [Effects](#) impostare la [Move](#)proprietà su.

5. Aggiungere il codice seguente per il [Drop](#) gestore eventi.

```
private void panel_Drop(object sender, DragEventArgs e)
{
    // If an element in the panel has already handled the drop,
    // the panel should not also handle it.
    if (e.Handled == false)
    {
        Panel _panel = (Panel)sender;
        UIElement _element = (UIElement)e.Data.GetData("Object");

        if (_panel != null && _element != null)
        {
            // Get the panel that the element currently belongs to,
            // then remove it from that panel and add it the Children of
            // the panel that its been dropped on.
            Panel _parent = (Panel)VisualTreeHelper.GetParent(_element);

            if (_parent != null)
            {
                if (e.KeyStates == DragDropKeyStates.ControlKey &&
                    e.AllowedEffects.HasFlag(DragDropEffects.Copy))
                {
                    Circle _circle = new Circle((Circle)_element);
                    _panel.Children.Add(_circle);
                    // set the value to return to the DoDragDrop call
                    e.Effects = DragDropEffects.Copy;
                }
                else if (e.AllowedEffects.HasFlag(DragDropEffects.Move))
                {
                    _parent.Children.Remove(_element);
                    _panel.Children.Add(_element);
                    // set the value to return to the DoDragDrop call
                    e.Effects = DragDropEffects.Move;
                }
            }
        }
    }
}
```

```

Private Sub panel_Drop(ByVal sender As System.Object, ByVal e As System.Windows.DragEventArgs)
    ' If an element in the panel has already handled the drop,
    ' the panel should not also handle it.
    If e.Handled = False Then
        Dim _panel As Panel = sender
        Dim _element As UIElement = e.Data.GetData("Object")

        If _panel IsNot Nothing And _element IsNot Nothing Then
            ' Get the panel that the element currently belongs to,
            ' then remove it from that panel and add it the Children of
            ' the panel that its been dropped on.

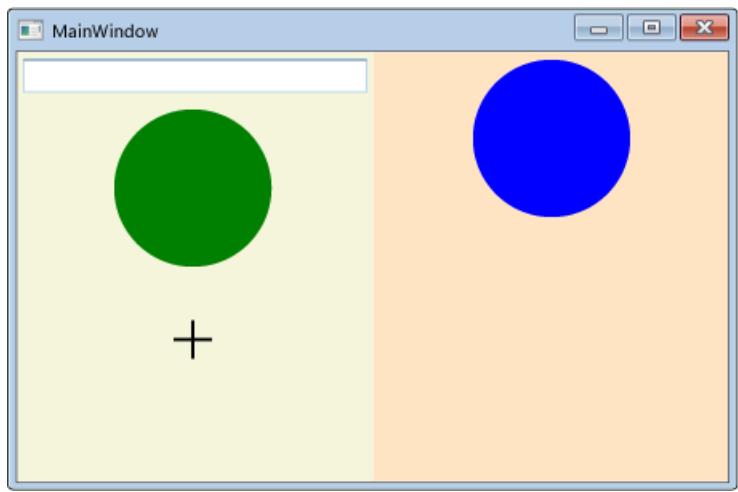
            Dim _parent As Panel = VisualTreeHelper.GetParent(_element)
            If _parent IsNot Nothing Then
                If e.KeyStates = DragDropKeyStates.ControlKey And _
                    e.AllowedEffects.HasFlag(DragDropEffects.Copy) Then
                    Dim _circle As New Circle(_element)
                    _panel.Children.Add(_circle)
                    ' set the value to return to the DoDragDrop call
                    e.Effects = DragDropEffects.Copy
                ElseIf e.AllowedEffects.HasFlag(DragDropEffects.Move) Then
                    _parent.Children.Remove(_element)
                    _panel.Children.Add(_element)
                    ' set the value to return to the DoDragDrop call
                    e.Effects = DragDropEffects.Move
                End If
            End If
        End If
    End If
End Sub

```

Questo **Drop** gestore eventi esegue le attività seguenti:

- Verifica se l' **Drop** evento è già stato gestito. Ad esempio, se un cerchio viene rilasciato in un altro cerchio che gestisce **Drop** l'evento, non si vuole che il pannello che contiene il cerchio lo gestisca anche.
- Se l' **Drop** evento non viene gestito, controlla se il tasto **CTRL** è premuto.
- Se si preme il tasto **CTRL** quando si **Drop** verifica l'evento, crea una copia del controllo **Circle** e **StackPanella Children** aggiunge alla raccolta di.
- Se il tasto **CTRL** non è premuto, sposta il cerchio dalla **Children** raccolta del relativo pannello **Children** padre alla raccolta del pannello su cui è stata rilasciata.
- Imposta la **Effects** proprietà per notificare **DoDragDrop** al metodo se è stata eseguita un'operazione di spostamento o copia.

6. Premere **F5** per compilare ed eseguire l'applicazione.
7. Selezionare il testo **green TextBoxda**.
8. Trascinare il testo su un controllo **Circle** e rilasciarlo.
9. Trascinare un controllo **Circle** dal pannello di sinistra al pannello di destra e rilasciarlo. Il cerchio viene rimosso dalla **Children** raccolta del pannello sinistro e aggiunto alla raccolta **Children** del pannello di destra.
10. Trascinare un controllo **Circle** dal pannello in cui si trova nell'altro pannello e rilasciarlo mentre si preme il tasto **CTRL**. Il cerchio viene copiato e la copia viene aggiunta alla **Children** raccolta del pannello di destinazione.



Vedere anche

- [Cenni preliminari sul trascinamento della selezione](#)

Procedure relative al trascinamento della selezione

23/10/2019 • 2 minutes to read • [Edit Online](#)

Gli esempi seguenti illustrano come eseguire attività comuni usando il Windows Presentation Foundation (WPF) framework di trascinamento e rilascio.

In questa sezione

[Aprire un file rilasciato in un controllo RichTextBox](#)

[Creare un oggetto dati](#)

[Determinare se un formato di dati è presente in un oggetto dati](#)

[Elencare i formati di dati in un oggetto dati](#)

[Recuperare dati in un formato dati particolare](#)

[Archiviare più formati di dati in un oggetto dati](#)

Vedere anche

- [Cenni preliminari sul trascinamento della selezione](#)

Procedura: Aprire un file rilasciato in un controllo RichTextBox

23/10/2019 • 4 minutes to read • [Edit Online](#)

In Windows Presentation Foundation (WPF) i controlli [TextBox](#), [RichTextBox](#) e [FlowDocument](#) hanno tutti la funzionalità di trascinamento della selezione incorporata. La funzionalità incorporata consente il trascinamento della selezione del testo all'interno e tra i controlli. Tuttavia, non consente di aprire un file eliminando il file nel controllo. Questi controlli contrassegnano anche gli eventi di trascinamento della selezione come gestiti. Di conseguenza, per impostazione predefinita, non è possibile aggiungere gestori di eventi personalizzati per fornire funzionalità per aprire i file eliminati.

Per aggiungere la gestione aggiuntiva per gli eventi di trascinamento della selezione in questi [AddHandler\(RoutedEvent, Delegate, Boolean\)](#) controlli, usare il metodo per aggiungere i gestori eventi per gli eventi di trascinamento della selezione. Impostare il `handledEventsToo` parametro su `true` per fare in modo che il gestore specificato venga richiamato per un evento indirizzato che è già stato contrassegnato come gestito da un altro elemento lungo la route dell'evento.

TIP

È possibile sostituire la funzionalità di trascinamento della selezione predefinita di [TextBox](#), [RichTextBox](#) e [FlowDocument](#) gestendo le versioni di anteprima degli eventi di trascinamento della selezione e contrassegnando gli eventi di anteprima come gestiti. Tuttavia, questa operazione Disabilita la funzionalità di trascinamento della selezione incorporata e non è consigliata.

Esempio

Nell'esempio seguente viene illustrato come aggiungere gestori per gli [DragOver](#) eventi e [Drop](#) in un oggetto [RichTextBox](#). Questo esempio usa il [AddHandler\(RoutedEvent, Delegate, Boolean\)](#) metodo e imposta il `handledEventsToo` parametro su `true` in modo che i [RichTextBox](#) gestori eventi vengano richiamati anche se questi eventi vengono contrassegnati come gestiti. Il codice nei gestori eventi aggiunge funzionalità per aprire un file di testo rilasciato in [RichTextBox](#).

Per testare questo esempio, trascinare un file di testo o un file RTF (Rich Text Format) da Esplora risorse a [RichTextBox](#). Il file verrà aperto in [RichTextBox](#). Se si preme il tasto MAIUSC prima dell'eliminazione del file, il file verrà aperto come testo normale.

```
<RichTextBox x:Name="richTextBox1"
    AllowDrop="True" />
```

```

public MainWindow()
{
    InitializeComponent();

    // Add using System.Windows.Controls;
    richTextBox1.AddHandler(RichTextBox.DragOverEvent, new DragEventHandler(RichTextBox_DragOver), true);
    richTextBox1.AddHandler(RichTextBox.DropEvent, new DragEventHandler(RichTextBox_Drop), true);
}

private void RichTextBox_DragOver(object sender, DragEventArgs e)
{
    if (e.Data.GetDataPresent(DataFormats.FileDrop))
    {
        e.Effects = DragDropEffects.All;
    }
    else
    {
        e.Effects = DragDropEffects.None;
    }
    e.Handled = false;
}

private void RichTextBox_Drop(object sender, DragEventArgs e)
{
    if (e.Data.GetDataPresent(DataFormats.FileDrop))
    {
        string[] docPath = (string[])e.Data.GetData(DataFormats.FileDrop);

        // By default, open as Rich Text (RTF).
        var dataFormat = DataFormats.Rtf;

        // If the Shift key is pressed, open as plain text.
        if (e.KeyStates == DragDropKeyStates.ShiftKey)
        {
            dataFormat = DataFormats.Text;
        }

        System.Windows.Documents.TextRange range;
        System.IO.FileStream fStream;
        if (System.IO.File.Exists(docPath[0]))
        {
            try
            {
                // Open the document in the RichTextBox.
                range = new System.Windows.Documents.TextRange(richTextBox1.Document.ContentStart,
richTextBox1.Document.ContentEnd);
                fStream = new System.IO.FileStream(docPath[0], System.IO.FileMode.OpenOrCreate);
                range.Load(fStream, dataFormat);
                fStream.Close();
            }
            catch (System.Exception)
            {
                MessageBox.Show("File could not be opened. Make sure the file is a text file.");
            }
        }
    }
}
}

```

```

Public Sub New()
    InitializeComponent()

    richTextBox1.AddHandler(RichTextBox.DragOverEvent, New DragEventHandler(AddressOf RichTextBox_DragOver),
    True)
    richTextBox1.AddHandler(RichTextBox.DropEvent, New DragEventHandler(AddressOf RichTextBox_Drop), True)

End Sub

Private Sub RichTextBox_DragOver(sender As Object, e As DragEventArgs)
    If e.Data.GetDataPresent(DataFormats.FileDrop) Then
        e.Effects = DragDropEffects.All
    Else
        e.Effects = DragDropEffects.None
    End If
    e.Handled = False
End Sub

Private Sub RichTextBox_Drop(sender As Object, e As DragEventArgs)
    If e.Data.GetDataPresent(DataFormats.FileDrop) Then
        Dim docPath As String() = TryCast(e.Data.GetData(DataFormats.FileDrop), String())

        ' By default, open as Rich Text (RTF).
        Dim dataFormat = DataFormats.Rtf

        ' If the Shift key is pressed, open as plain text.
        If e.KeyStates = DragDropKeyStates.ShiftKey Then
            dataFormat = DataFormats.Text
        End If

        Dim range As TextRange
        Dim fStream As IO.FileStream
        If IO.File.Exists(docPath(0)) Then
            Try
                ' Open the document in the RichTextBox.
                range = New TextRange(richTextBox1.Document.ContentStart, richTextBox1.Document.ContentEnd)
                fStream = New IO.FileStream(docPath(0), IO FileMode.OpenOrCreate)
                range.Load(fStream, dataFormat)
                fStream.Close()
            Catch generatedExceptionName As System.Exception
                MessageBox.Show("File could not be opened. Make sure the file is a text file.")
            End Try
        End If
    End Sub

```

Procedura: Creare un oggetto dati

23/10/2019 • 4 minutes to read • [Edit Online](#)

Gli esempi seguenti illustrano vari modi per creare un oggetto dati usando i costruttori forniti dalla classe **DataObject**.

Esempio

Descrizione

Esempio di codice seguente crea un nuovo oggetto di dati e viene utilizzato uno dei costruttori di overload ([DataObject\(Object\)](#)) per inizializzare l'oggetto dati con una stringa. In questo caso, un formato appropriato dei dati viene determinato automaticamente in base al tipo di dati archiviati e la conversione automatica dei dati archiviati è consentita per impostazione predefinita.

Codice

```
string stringData = "Some string data to store...";  
DataObject dataObject = new DataObject(stringData);
```

```
Dim stringData As String = "Some string data to store..."  
Dim dataObject As New DataObject(stringData)
```

Descrizione

Esempio di codice seguente è una versione ridotta di codice riportato sopra.

Codice

```
DataObject dataObject = new DataObject("Some string data to store...");
```

```
Dim dataObject As New DataObject("Some string data to store...")
```

Esempio

Descrizione

Esempio di codice seguente crea un nuovo oggetto di dati e viene utilizzato uno dei costruttori di overload ([DataObject\(String, Object\)](#)) per inizializzare l'oggetto dati con una stringa e un formato dati specificato. In questo caso viene specificato il formato dei dati da una stringa; il [DataFormats](#) classe fornisce un set di stringhe di tipi predefiniti. Per impostazione predefinita è consentita la conversione automatica dei dati archiviati.

Codice

```
string stringData = "Some string data to store...";  
string dataFormat = DataFormats.UnicodeText;  
DataObject dataObject = new DataObject(dataFormat, stringData);
```

```
Dim stringData As String = "Some string data to store..."  
Dim dataFormat As String = DataFormats.UnicodeText  
Dim dataObject As New DataObject(dataFormat, stringData)
```

Descrizione

Esempio di codice seguente è una versione ridotta di codice riportato sopra.

Codice

```
DataObject dataObject = new DataObject(DataFormats.UnicodeText, "Some string data to store...");
```

```
Dim dataObject As New DataObject(DataFormats.UnicodeText, "Some string data to store...")
```

Esempio

Descrizione

Esempio di codice seguente crea un nuovo oggetto di dati e viene utilizzato uno dei costruttori di overload ([DataObject](#)) per inizializzare l'oggetto dati con una stringa e un formato dati specificato. In questo caso viene specificato il formato dei dati da un [Type](#) parametro. Per impostazione predefinita è consentita la conversione automatica dei dati archiviati.

Codice

```
string stringData = "Some string data to store...";  
Type dataFormat = stringData.GetType();  
DataObject dataObject = new DataObject(dataFormat, stringData);
```

```
Dim stringData As String = "Some string data to store..."  
Dim dataFormat As Type = stringData.GetType()  
Dim dataObject As New DataObject(dataFormat, stringData)
```

Descrizione

Esempio di codice seguente è una versione ridotta di codice riportato sopra.

Codice

```
DataObject dataObject = new DataObject("".GetType(), "Some string data to store...");
```

```
Dim dataObject As New DataObject("".GetType(), "Some string data to store...")
```

Esempio

Descrizione

Esempio di codice seguente crea un nuovo oggetto di dati e viene utilizzato uno dei costruttori di overload ([DataObject\(String, Object, Boolean\)](#)) per inizializzare l'oggetto dati con una stringa e un formato dati specificato. In questo caso viene specificato il formato dei dati da una stringa; il [DataFormats](#) classe fornisce un set di stringhe di tipi predefiniti. Questo overload del costruttore specifico consente al chiamante di specificare se è consentita la conversione automatica.

Codice

```
string stringData = "Some string data to store...";  
string dataFormat = DataFormats.Text;  
bool autoConvert = false;  
DataObject dataObject = new DataObject(dataFormat, stringData, autoConvert);
```

```
Dim stringData As String = "Some string data to store..."  
Dim dataFormat As String = DataFormats.Text  
Dim autoConvert As Boolean = False  
Dim dataObject As New DataObject(dataFormat, stringData, autoConvert)
```

Descrizione

Esempio di codice seguente è una versione ridotta di codice riportato sopra.

Codice

```
DataObject dataObject = new DataObject(DataFormats.Text, "Some string data to store...", false);
```

```
Dim dataObject As New DataObject(DataFormats.Text, "Some string data to store...", False)
```

Vedere anche

- [IDataObject](#)

Procedura: Determinare se un formato dati è presente in un oggetto dati

23/10/2019 • 4 minutes to read • [Edit Online](#)

Gli esempi seguenti illustrano come usare i vari `GetDataPresent` overload del metodo di query se un particolare formato dati è presente in un oggetto dati.

Esempio

Descrizione

Il codice di esempio seguente usa il `GetDataPresent(String)` overload per eseguire query per la presenza di un formato dati particolare dalla stringa del descrittore.

Codice

```
DataObject dataObject = new DataObject("Some string data to store...");  
  
// Query for the presence of Text data in the data object, by a data format descriptor string.  
// In this overload of GetDataPresent, the method will return true both for native data formats  
// and when the data can automatically be converted to the specified format.  
  
// In this case, string data is present natively, so GetDataPresent returns "true".  
string textData = null;  
if (dataObject.GetDataPresent(DataFormats.StringFormat))  
{  
    textData = dataObject.GetData(DataFormats.StringFormat) as string;  
}  
  
// In this case, the Text data in the data object can be autoconverted to  
// Unicode text, so GetDataPresent returns "true".  
byte[] unicodeData = null;  
if (dataObject.GetDataPresent(DataFormats.UnicodeText))  
{  
    unicodeData = dataObject.GetData(DataFormats.UnicodeText) as byte[];  
}
```

```
Dim dataObject As New DataObject("Some string data to store...")  
  
' Query for the presence of Text data in the data object, by a data format descriptor string.  
' In this overload of GetDataPresent, the method will return true both for native data formats  
' and when the data can automatically be converted to the specified format.  
  
' In this case, string data is present natively, so GetDataPresent returns "true".  
Dim textData As String = Nothing  
If dataObject.GetDataPresent(DataFormats.StringFormat) Then  
    textData = TryCast(dataObject.GetData(DataFormats.StringFormat), String)  
End If  
  
' In this case, the Text data in the data object can be autoconverted to  
' Unicode text, so GetDataPresent returns "true".  
Dim unicodeData() As Byte = Nothing  
If dataObject.GetDataPresent(DataFormats.UnicodeText) Then  
    unicodeData = TryCast(dataObject.GetData(DataFormats.UnicodeText), Byte())  
End If
```

Esempio

Descrizione

Il codice di esempio seguente usa il [GetDataPresent\(Type\)](#) overload per eseguire query per la presenza di un formato di dati determinato dal tipo.

Codice

```
DataObject dataObject = new DataObject("Some string data to store...");  
  
// Query for the presence of String data in the data object, by type. In this overload  
// of GetDataPresent, the method will return true both for native data formats  
// and when the data can automatically be converted to the specified format.  
  
// In this case, the Text data present in the data object can be autoconverted  
// to type string (also represented by DataFormats.String), so GetDataPresent returns "true".  
string stringData = null;  
if (dataObject.GetDataPresent(typeof(string)))  
{  
    stringData = dataObject.GetData(DataFormats.Text) as string;  
}
```

```
Dim dataObject As New DataObject("Some string data to store...")  
  
' Query for the presence of String data in the data object, by type. In this overload  
' of GetDataPresent, the method will return true both for native data formats  
' and when the data can automatically be converted to the specified format.  
  
' In this case, the Text data present in the data object can be autoconverted  
' to type string (also represented by DataFormats.String), so GetDataPresent returns "true".  
Dim stringData As String = Nothing  
If dataObject.GetDataPresent(GetType(String)) Then  
    stringData = TryCast(dataObject.GetData(DataFormats.Text), String)  
End If
```

Esempio

Descrizione

Il codice di esempio seguente usa il [GetDataPresent\(String, Boolean\)](#) overload per eseguire query sui dati dalla stringa del descrittore e specifica come trattare i formati di dati di conversione automatica.

Codice

```

DataObject dataObject = new DataObject("Some string data to store...");

// Query for the presence of Text data in the data object, by data format descriptor string,
// and specifying whether auto-convertible data formats are acceptable.

// In this case, Text data is present natively, so GetDataPresent returns "true".
string textData = null;
if (dataObject.GetDataPresent(DataFormats.Text, false /* Auto-convert? */))
{
    textData = dataObject.GetData(DataFormats.Text) as string;
}

// In this case, the Text data in the data object can be autoconverted to
// Unicode text, but it is not available natively, so GetDataPresent returns "false".
byte[] unicodeData = null;
if (dataObject.GetDataPresent(DataFormats.UnicodeText, false /* Auto-convert? */))
{
    unicodeData = dataObject.GetData(DataFormats.UnicodeText) as byte[];
}

// In this case, the Text data in the data object can be autoconverted to
// Unicode text, so GetDataPresent returns "true".
if (dataObject.GetDataPresent(DataFormats.UnicodeText, true /* Auto-convert? */))
{
    unicodeData = dataObject.GetData(DataFormats.UnicodeText) as byte[];
}

```

```

Dim dataObject As New DataObject("Some string data to store...")

' Query for the presence of Text data in the data object, by data format descriptor string,
' and specifying whether auto-convertible data formats are acceptable.

' In this case, Text data is present natively, so GetDataPresent returns "true".
Dim textData As String = Nothing
If dataObject.GetDataPresent(DataFormats.Text, False) Then ' Auto-convert?
    textData = TryCast(dataObject.GetData(DataFormats.Text), String)
End If

' In this case, the Text data in the data object can be autoconverted to
' Unicode text, but it is not available natively, so GetDataPresent returns "false".
Dim unicodeData() As Byte = Nothing
If dataObject.GetDataPresent(DataFormats.UnicodeText, False) Then ' Auto-convert?
    unicodeData = TryCast(dataObject.GetData(DataFormats.UnicodeText), Byte())
End If

' In this case, the Text data in the data object can be autoconverted to
' Unicode text, so GetDataPresent returns "true".
If dataObject.GetDataPresent(DataFormats.UnicodeText, True) Then ' Auto-convert?
    unicodeData = TryCast(dataObject.GetData(DataFormats.UnicodeText), Byte())
End If

```

Vedere anche

- [IDataObject](#)

Procedura: Elencare i formati dati in un oggetto dati

23/10/2019 • 3 minutes to read • [Edit Online](#)

Gli esempi seguenti illustrano come usare il [GetFormats](#) overload del metodo get di una matrice di stringhe che indicano di ogni formato di dati che è disponibile in un oggetto dati.

Esempio

Descrizione

Il codice di esempio seguente usa il [GetFormats](#) overload per ottenere una matrice di stringhe che indicano tutti i formati di dati disponibili in un oggetto dati (nativo e conversione automatica).

Codice

```
DataObject dataObject = new DataObject("Some string data to store...");  
  
// Get an array of strings, each string denoting a data format  
// that is available in the data object. This overload of GetDataFormats  
// returns all available data formats, native and auto-convertible.  
string[] dataFormats = dataObject.GetFormats();  
  
// Get the number of data formats present in the data object, including both  
// auto-convertible and native data formats.  
int numberOfDataFormats = dataFormats.Length;  
  
// To enumerate the resulting array of data formats, and take some action when  
// a particular data format is found, use a code structure similar to the following.  
foreach (string dataFormat in dataFormats)  
{  
    if (dataFormat == DataFormats.Text)  
    {  
        // Take some action if/when data in the Text data format is found.  
        break;  
    }  
    else if(dataFormat == DataFormats.StringFormat)  
    {  
        // Take some action if/when data in the string data format is found.  
        break;  
    }  
}
```

```

Dim dataObject As New DataObject("Some string data to store...")

' Get an array of strings, each string denoting a data format
' that is available in the data object. This overload of GetDataFormats
' returns all available data formats, native and auto-convertible.
Dim dataFormats() As String = dataObject.GetFormats()

' Get the number of data formats present in the data object, including both
' auto-convertible and native data formats.
Dim numberOfDataFormats As Integer = dataFormats.Length

' To enumerate the resulting array of data formats, and take some action when
' a particular data format is found, use a code structure similar to the following.
For Each dataFormat As String In dataFormats
    If dataFormat = System.Windows.DataFormats.Text Then
        ' Take some action if/when data in the Text data format is found.
        Exit For
    ElseIf dataFormat = System.Windows.DataFormats.StringFormat Then
        ' Take some action if/when data in the string data format is found.
        Exit For
    End If
Next dataFormat

```

Esempio

Descrizione

Il codice di esempio seguente usa il [GetFormats](#) overload per ottenere una matrice di stringhe che indicano solo i formati di dati disponibili in un oggetto dati (auto-convertibili formati dati vengono filtrati).

Codice

```

DataObject dataObject = new DataObject("Some string data to store...");

// Get an array of strings, each string denoting a data format
// that is available in the data object. This overload of GetDataFormats
// accepts a Boolean parameter indicating whether to include auto-convertible
// data formats, or only return native data formats.
string[] dataFormats = dataObject.GetFormats(false /* Include auto-convertible? */);

// Get the number of native data formats present in the data object.
int numberOfDataFormats = dataFormats.Length;

// To enumerate the resulting array of data formats, and take some action when
// a particular data format is found, use a code structure similar to the following.
foreach (string dataFormat in dataFormats)
{
    if (dataFormat == DataFormats.Text)
    {
        // Take some action if/when data in the Text data format is found.
        break;
    }
}

```

```
Dim dataObject As New DataObject("Some string data to store...")

' Get an array of strings, each string denoting a data format
' that is available in the data object. This overload of GetDataFormats
' accepts a Boolean parameter indicating whether to include auto-convertible
' data formats, or only return native data formats.
Dim dataFormats() As String = dataObject.GetFormats(False) ' Include auto-convertible?

' Get the number of native data formats present in the data object.
Dim numberOfDataFormats As Integer = dataFormats.Length

' To enumerate the resulting array of data formats, and take some action when
' a particular data format is found, use a code structure similar to the following.
For Each dataFormat As String In dataFormats
    If dataFormat = System.Windows.DataFormats.Text Then
        ' Take some action if/when data in the Text data format is found.
        Exit For
    End If
Next dataFormat
```

Vedere anche

- [IDataObject](#)
- [Cenni preliminari sul trascinamento della selezione](#)

Procedura: Recuperare dati in un formato dati particolare

23/10/2019 • 2 minutes to read • [Edit Online](#)

Negli esempi seguenti mostrano come recuperare i dati da un oggetto dati in un formato specificato.

Esempio

Descrizione

Il codice di esempio seguente usa il [GetDataPresent\(String\)](#) overload per prima cosa controllare se i dati specificati di formato è disponibile (in modo nativo o tramite conversione automatica); se il formato specificato è disponibile, l'esempio recupera i dati utilizzando il [GetData\(String\)](#) (metodo).

Codice

```
DataObject dataObject = new DataObject("Some string data to store...");  
  
string desiredFormat = DataFormats.UnicodeText;  
byte[] data = null;  
  
// Use the GetDataPresent method to check for the presence of a desired data format.  
// This particular overload of GetDataPresent looks for both native and auto-convertible  
// data formats.  
if (dataObject.GetDataPresent(desiredFormat))  
{  
    // If the desired data format is present, use one of the GetData methods to retrieve the  
    // data from the data object.  
    data = dataObject.GetData(desiredFormat) as byte[];  
}
```

```
Dim dataObject As New DataObject("Some string data to store...")  
  
Dim desiredFormat As String = DataFormats.UnicodeText  
Dim data() As Byte = Nothing  
  
' Use the GetDataPresent method to check for the presence of a desired data format.  
' This particular overload of GetDataPresent looks for both native and auto-convertible  
' data formats.  
If dataObject.GetDataPresent(desiredFormat) Then  
    ' If the desired data format is present, use one of the GetData methods to retrieve the  
    ' data from the data object.  
    data = TryCast(dataObject.GetData(desiredFormat), Byte())  
End If
```

Esempio

Descrizione

Il codice di esempio seguente usa il [GetDataPresent\(String, Boolean\)](#) overload prima di tutto verificare se un formato dati specificato è disponibile in modo nativo (formati convertibili automaticamente i dati vengono filtrati); se il formato specificato è disponibile, l'esempio recupera i dati usando il [GetData\(String\)](#) metodo.

Codice

```
DataObject dataObject = new DataObject("Some string data to store...");  
  
string desiredFormat = DataFormats.UnicodeText;  
bool noAutoConvert = false;  
byte[] data = null;  
  
// Use the GetDataPresent method to check for the presence of a desired data format.  
// The autoconvert parameter is set to false to filter out auto-convertible data formats,  
// returning true only if the specified data format is available natively.  
if (dataObject.GetDataPresent(desiredFormat, noAutoConvert))  
{  
    // If the desired data format is present, use one of the GetData methods to retrieve the  
    // data from the data object.  
    data = dataObject.GetData(desiredFormat) as byte[];  
}
```

```
Dim dataObject As New DataObject("Some string data to store...")  
  
Dim desiredFormat As String = DataFormats.UnicodeText  
Dim noAutoConvert As Boolean = False  
Dim data() As Byte = Nothing  
  
' Use the GetDataPresent method to check for the presence of a desired data format.  
' The autoconvert parameter is set to false to filter out auto-convertible data formats,  
' returning true only if the specified data format is available natively.  
If dataObject.GetDataPresent(desiredFormat, noAutoConvert) Then  
    ' If the desired data format is present, use one of the GetData methods to retrieve the  
    ' data from the data object.  
    data = TryCast(dataObject.GetData(desiredFormat), Byte())  
End If
```

Vedere anche

- [IDataObject](#)
- [Cenni preliminari sul trascinamento della selezione](#)

Procedura: Archiviare più formati dati in un oggetto dati

23/10/2019 • 2 minutes to read • [Edit Online](#)

Nell'esempio seguente viene illustrato come utilizzare il [SetData\(String, Object\)](#) metodo per aggiungere dati a un oggetto dati in più formati.

Esempio

Descrizione

Codice

```
DataObject dataObject = new DataObject();
string sourceData = "Some string data to store...";

// Encode the source string into Unicode byte arrays.
byte[] unicodeText = Encoding.Unicode.GetBytes(sourceData); // UTF-16
byte[] utf8Text = Encoding.UTF8.GetBytes(sourceData);
byte[] utf32Text = Encoding.UTF32.GetBytes(sourceData);

// The DataFormats class does not provide data format fields for denoting
// UTF-32 and UTF-8, which are seldom used in practice; the following strings
// will be used to identify these "custom" data formats.
string utf32DateFormat = "UTF-32";
string utf8DateFormat = "UTF-8";

// Store the text in the data object, letting the data object choose
// the data format (which will be DataFormats.Text in this case).
dataObject.SetData(sourceData);
// Store the Unicode text in the data object. Text data can be automatically
// converted to Unicode (UTF-16 / UCS-2) format on extraction from the data object;
// Therefore, explicitly converting the source text to Unicode is generally unnecessary, and
// is done here as an exercise only.
dataObject.SetData(DataFormats.UnicodeText, unicodeText);
// Store the UTF-8 text in the data object...
dataObject.SetData(utf8DateFormat, utf8Text);
// Store the UTF-32 text in the data object...
dataObject.SetData(utf32DateFormat, utf32Text);
```

```
Dim dataObject As New DataObject()
Dim sourceData As String = "Some string data to store..."

' Encode the source string into Unicode byte arrays.
Dim unicodeText() As Byte = Encoding.Unicode.GetBytes(sourceData) ' UTF-16
Dim utf8Text() As Byte = Encoding.UTF8.GetBytes(sourceData)
Dim utf32Text() As Byte = Encoding.UTF32.GetBytes(sourceData)

' The DataFormats class does not provide data format fields for denoting
' UTF-32 and UTF-8, which are seldom used in practice; the following strings
' will be used to identify these "custom" data formats.
Dim utf32DataFormat As String = "UTF-32"
Dim utf8DataFormat As String = "UTF-8"

' Store the text in the data object, letting the data object choose
' the data format (which will be DataFormats.Text in this case).
dataObject.SetData(sourceData)

' Store the Unicode text in the data object. Text data can be automatically
' converted to Unicode (UTF-16 / UCS-2) format on extraction from the data object;
' Therefore, explicitly converting the source text to Unicode is generally unnecessary, and
' is done here as an exercise only.
dataObject.SetData(DataFormats.UnicodeText, unicodeText)
' Store the UTF-8 text in the data object...
dataObject.SetData(utf8DataFormat, utf8Text)
' Store the UTF-32 text in the data object...
dataObject.SetData(utf32DataFormat, utf32Text)
```

Vedere anche

- [IDataObject](#)
- [Cenni preliminari sul trascinamento della selezione](#)

Risorse (WPF)

03/02/2020 • 2 minutes to read • [Edit Online](#)

Una risorsa è un oggetto che è possibile riusare in posizioni diverse all'interno dell'applicazione. WPF supporta diversi tipi di risorse. Queste risorse sono principalmente due tipi di risorse: risorse XAML e file di dati delle risorse. Esempi di risorse XAML includono pennelli e stili. I file di dati delle risorse sono file di dati non eseguibili necessari per un'applicazione.

Contenuto della sezione

[Risorse XAML](#)

[File di dati e di risorse dell'applicazione WPF](#)

[URI di tipo pack in WPF](#)

Riferimento

[ResourceDictionary](#)

[Estensione di markup StaticResource](#)

[Estensione del markup DynamicResource](#)

[Direttiva x:Key](#)

Sezioni correlate

[XAML in WPF](#)

2 minutes to read

Risorse e codice

10/01/2020 • 9 minutes to read • [Edit Online](#)

Questo argomento illustra le modalità di accesso alle risorse di Windows Presentation Foundation (WPF) o di creazione delle risorse stesse tramite codice anziché tramite la sintassi Extensible Application Markup Language (XAML). Per altre informazioni sull'uso delle risorse in generale e sulle risorse dal punto di vista della sintassi XAML, vedere [Risorse XAML](#).

Accesso alle risorse dal codice

Le chiavi che identificano le risorse, se definite tramite XAML, possono essere usate anche per recuperare risorse specifiche qualora si richieda la risorsa nel codice. Il modo più semplice per recuperare una risorsa dal codice consiste nel chiamare il [FindResource](#) o il metodo [TryFindResource](#) da oggetti a livello di Framework nell'applicazione. La differenza di comportamento tra i due metodi sta nell'azione eseguita quando la chiave richiesta non viene trovata. [FindResource](#) genera un'eccezione; [TryFindResource](#) non genererà un'eccezione ma restituirà `null`. Ogni metodo accetta la chiave di risorsa come parametro di input e restituisce un oggetto debolmente tipizzato. In genere una chiave di risorsa è una stringa, ma esistono utilizzi occasionali di tipo non stringa. Per informazioni dettagliate, vedere la sezione [Uso di oggetti come chiavi](#). Di regola, il cast dell'oggetto restituito viene eseguito nel tipo richiesto dalla proprietà impostata al momento della richiesta della risorsa. La logica di ricerca per la risoluzione della risorsa codice è la stessa del codice XAML del riferimento di risorsa dinamica. La ricerca delle risorse ha inizio dall'elemento chiamante e continua con gli elementi padre successivi dell'albero logico. Prosegue quindi nelle risorse, nei temi e, se necessario, nelle risorse di sistema dell'applicazione. Una richiesta di codice per una risorsa terrà in considerazione le modifiche di runtime eventualmente apportate ai dizionari risorse successivamente al caricamento di un determinato dizionario risorse da XAML, oltre che le modifiche apportate alle risorse di sistema in tempo reale.

Di seguito è riportato un breve esempio di codice in cui viene trovata una risorsa in base alla chiave e viene utilizzato il valore restituito per impostare una proprietà, implementata come gestore dell'evento [Click](#).

```
void SetBGBYResource(object sender, RoutedEventArgs e)
{
    Button b = sender as Button;
    b.Background = (Brush)this.FindResource("RainbowBrush");
}
```

```
Private Sub SetBGBYResource(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Dim b As Button = TryCast(sender, Button)
    b.Background = CType(Me.FindResource("RainbowBrush"), Brush)
End Sub
```

Un metodo alternativo per l'assegnazione di un riferimento alla risorsa è [SetResourceReference](#). Questo metodo accetta due parametri: la chiave della risorsa e l'identificatore di una particolare proprietà di dipendenza presente nell'istanza dell'elemento a cui deve essere assegnato il valore della risorsa. A livello funzionale, questo metodo è identico al precedente e presenta il vantaggio di non richiedere il cast dei valori restituiti.

Un altro modo per accedere alle risorse a livello di codice consiste nell'accedere al contenuto della proprietà [Resources](#) come dizionario. L'accesso al dizionario incluso in questa proprietà consente anche di aggiungere nuove risorse alle raccolte esistenti, di verificare se un determinato nome di chiave è già usato nella raccolta e di eseguire altre operazioni sul dizionario o sulla raccolta. Se si scrive un'applicazione WPF interamente nel codice, è anche possibile creare l'intera raccolta nel codice, assegnarvi le chiavi e quindi assegnare la raccolta completata

alla proprietà [Resources](#) di un elemento stabilito. Questa procedura sarà descritta nella sezione successiva.

È possibile indicizzare in una determinata raccolta di [Resources](#), usando una chiave specifica come indice, ma è necessario tenere presente che l'accesso alla risorsa in questo modo non segue le normali regole di runtime della risoluzione delle risorse. Viene eseguito l'accesso solo a quella particolare risorsa. La ricerca della risorsa non attraversa l'ambito fino alla radice o all'applicazione se non è stato trovato alcun oggetto valido in corrispondenza della chiave richiesta. Questo approccio, tuttavia, in alcuni casi può presentare vantaggi in termini di prestazioni, proprio perché l'ambito di ricerca della chiave è più limitato. Per informazioni dettagliate sull'uso diretto del dizionario risorse, vedere la classe [ResourceDictionary](#).

Creazione delle risorse tramite codice

Se si vuole creare un'intera applicazione WPF in codice, può essere necessario creare in codice anche eventuali risorse di tale applicazione. A tale scopo, creare una nuova istanza di [ResourceDictionary](#), quindi aggiungere tutte le risorse al dizionario usando le chiamate successive a [ResourceDictionary.Add](#). Usare quindi il [ResourceDictionary](#) creato per impostare la proprietà [Resources](#) su un elemento presente in un ambito di pagina o il [Application.Resources](#). È anche possibile gestire il [ResourceDictionary](#) come oggetto autonomo senza aggiungerlo a un elemento. In questo caso, tuttavia, è necessario accedere alle risorse al suo interno tramite la chiave dell'elemento, come se si trattasse di un dizionario generico. Un [ResourceDictionary](#) che non è associato a un elemento [Resources](#) proprietà non esiste come parte dell'albero degli elementi e non ha alcun ambito in una sequenza di ricerca che può essere usata da [FindResource](#) e dai metodi correlati.

Uso di oggetti come chiavi

La chiave di una risorsa sarà impostata come stringa nella maggior parte degli utilizzi di quella risorsa. Diverse funzionalità di WPF, tuttavia, non usano un tipo stringa per specificare le chiavi, quel parametro sarà un oggetto. La possibilità di disporre di un oggetto come chiave della risorsa viene usata dal supporto degli stili e dei temi di WPF. Gli stili nei temi che diventano lo stile predefinito per un controllo altrimenti senza stile sono ognuno con chiave dal [Type](#) del controllo a cui devono essere applicati. L'uso di un tipo come chiave offre un meccanismo di ricerca affidabile che funziona sulle istanze predefinite di ogni tipo di controllo, pertanto il tipo può essere rilevato mediante reflection e usato per applicare uno stile alle classi derivate, anche se il tipo derivato non disporrebbe altrimenti di uno stile predefinito. È possibile specificare una chiave di [Type](#) per una risorsa definita in XAML usando l'[estensione di markup x:Type](#). Esistono estensioni analoghe per altri utilizzi di chiavi di tipo non stringa che supportano funzionalità di WPF, come l'[estensione di markup ComponentResourceKey](#).

Vedere anche

- [Risorse XAML](#)
- [Applicazione di stili e modelli](#)

Dizionari risorse uniti

08/01/2020 • 10 minutes to read • [Edit Online](#)

Le risorse di Windows Presentation Foundation (WPF) supportano una funzionalità di dizionari risorse uniti. Questa funzionalità consente di definire la parte delle risorse di un'applicazione WPF al di fuori dell'applicazione XAML compilata. Le risorse possono quindi essere condivise nelle applicazioni, oltre che isolate più facilmente per la localizzazione.

Introduzione di un dizionario risorse unito

Nel markup si usa la sintassi seguente per introdurre un dizionario risorse unito in una pagina:

```
<Page.Resources>
  <ResourceDictionary>
    <ResourceDictionary.MergedDictionaries>
      <ResourceDictionary Source="myresourcedictionary.xaml"/>
      <ResourceDictionary Source="myresourcedictionary2.xaml"/>
    </ResourceDictionary.MergedDictionaries>
  </ResourceDictionary>
</Page.Resources>
```

Si noti che l'elemento `ResourceDictionary` non dispone di una `direttiva x:Key`, che in genere è necessaria per tutti gli elementi di una raccolta di risorse. Un altro `ResourceDictionary` riferimento all'interno della raccolta di `MergedDictionaries` è un caso speciale, riservato per questo scenario di dizionario risorse Unito. Il `ResourceDictionary` che introduce un dizionario risorse Unito non può avere una `direttiva x:Key`. In genere, ogni `ResourceDictionary` all'interno della raccolta di `MergedDictionaries` specifica un attributo di `Source`. Il valore di `Source` deve essere un URI (Uniform Resource Identifier) che viene risolto nel percorso del file di risorse da unire. La destinazione di tale URI deve essere un altro XAML file, con `ResourceDictionary` come elemento radice.

NOTE

È possibile definire le risorse all'interno di un `ResourceDictionary` specificato come dizionario Unito, in alternativa alla specifica di `Source` in aggiunta a qualsiasi risorsa inclusa dall'origine specificata. ma questo non è uno scenario comune. Lo scenario principale per i dizionari uniti prevede l'unione delle risorse da percorsi di file esterni. Se si desidera specificare le risorse all'interno del markup per una pagina, è in genere necessario definirle nel `ResourceDictionary` principale e non nei dizionari uniti.

Comportamento dei dizionari uniti

Le risorse in un dizionario unito fanno parte dell'ambito della ricerca di risorse, che segue immediatamente l'ambito del dizionario risorse principale in cui vengono unite. Anche se una chiave di risorsa deve essere univoca in ogni singolo dizionario, una chiave può esistere più volte in un set di dizionari uniti. In questo caso, la risorsa restituita proverrà dall'ultimo dizionario trovato in sequenza nella raccolta di `MergedDictionaries`. Se la raccolta di `MergedDictionaries` è stata definita in XAML, l'ordine dei dizionari uniti nella raccolta è l'ordine degli elementi come specificato nel markup. Se una chiave viene definita nel dizionario primario e anche in un dizionario unito, la risorsa restituita proviene dal dizionario primario. Queste regole di ambito si applicano ugualmente sia ai riferimenti a risorse statiche che ai riferimenti a risorse dinamiche.

Dizionari uniti e codice

I dizionari uniti possono essere aggiunti a un dizionario `Resources` tramite il codice. Il `ResourceDictionary`

predefinito inizialmente vuoto presente per qualsiasi proprietà `Resources` dispone anche di una proprietà di raccolta `MergedDictionaries` predefinita inizialmente vuota. Per aggiungere un dizionario unito tramite codice, ottenere un riferimento al `ResourceDictionary` primario desiderato, ottenere il valore della proprietà `MergedDictionaries` e chiamare `Add` sul `Collection` generico contenuto in `MergedDictionaries`. L'oggetto aggiunto deve essere un nuovo `ResourceDictionary`. Nel codice non si imposta la proprietà `Source`. Al contrario, è necessario ottenere un oggetto `ResourceDictionary` creandone uno o caricando uno di essi. Un modo per caricare un `ResourceDictionary` esistente per chiamare `XamlReader.Load` su un flusso di file XAML esistente con una radice `ResourceDictionary`, quindi eseguire il cast del valore `XamlReader.Load` restituito a `ResourceDictionary`.

URI di dizionari risorse uniti

Esistono diverse tecniche per includere un dizionario risorse Unito, che è indicato dal formato URI (Uniform Resource Identifier) che verrà usato. In generale, queste tecniche si possono suddividere in due categorie: risorse compilate come parte del progetto e risorse non compilate come parte del progetto.

Per le risorse compilate come parte del progetto, è possibile usare un percorso relativo che fa riferimento alla posizione della risorsa. Il percorso relativo viene valutato durante la compilazione. La risorsa deve essere definita nell'ambito del progetto come azione di compilazione Risorsa. Se si include un file di risorse XAML nel progetto come risorsa, non è necessario copiare il file di risorse nella directory di output, perché la risorsa è già inclusa nell'applicazione compilata. È anche possibile usare un'azione di compilazione Contenuto, ma è necessario copiare i file nella directory di output e distribuire i file di risorse nella stessa relazione percorso con l'eseguibile.

NOTE

Non usare l'azione di compilazione Risorsa incorporata. L'azione di compilazione è supportata per le applicazioni WPF, ma la risoluzione dei `Source` non incorpora `ResourceManager` pertanto non può separare la singola risorsa dal flusso. È comunque possibile usare la risorsa incorporata per altri scopi purché sia stata usata anche `ResourceManager` per accedere alle risorse.

Una tecnica correlata consiste nell'usare un URI di tipo pack di un file XAML e nel farvi riferimento come origine. Un URI di tipo pack abilita i riferimenti ai componenti degli assembly a cui si fa riferimento e altre tecniche. Per altre informazioni sugli URI di tipo pack, vedere [File di dati e di risorse dell'applicazione WPF](#).

Per le risorse non compilate come parte del progetto, l'URI viene valutato in fase di esecuzione. È possibile usare un trasporto URI comune, ad esempio `file:` o `http:` per fare riferimento al file di risorse. Lo svantaggio nell'uso dell'approccio delle risorse non compilate è che l'accesso `file:` richiede altri passaggi per la distribuzione e l'accesso `http:` implica la zona di sicurezza Internet.

Riutilizzo dei dizionari uniti

È possibile riutilizzare o condividere dizionari risorse uniti tra le applicazioni, perché è possibile fare riferimento al dizionario risorse da unire tramite qualsiasi URI (Uniform Resource Identifier) valido. La scelta del modo per procedere dipenderà dalla strategia di distribuzione dell'applicazione e dal modello di applicazione seguito. La suddetta strategia basata sugli URI di tipo pack consente di originare a livello comune una risorsa unita in più progetti durante lo sviluppo condividendo un riferimento ad assembly. In questo scenario le risorse vengono ancora distribuite dal client e almeno una delle applicazioni deve distribuire l'assembly di riferimento. È anche possibile fare riferimento alle risorse unite tramite un URI distribuito che usa il protocollo `http`.

La scrittura di dizionari uniti come file dell'applicazione locali o nello spazio di archiviazione condiviso locale è un altro possibile scenario di distribuzione dell'applicazione/dizionari uniti.

Localizzazione

Se le risorse che è necessario localizzare vengono isolate nei dizionari uniti nei dizionari primari e conservate come XAML libero, questi file possono essere localizzati separatamente. Questa tecnica è un'alternativa leggera alla localizzazione degli assembly di risorse satellite. Per informazioni dettagliate, vedere [Panoramica della globalizzazione e localizzazione WPF](#).

Vedere anche

- [ResourceDictionary](#)
- [Risorse XAML](#)
- [Risorse e codice](#)
- [File di dati e di risorse dell'applicazione WPF](#)

Procedure relative alle risorse

04/11/2019 • 2 minutes to read • [Edit Online](#)

Negli argomenti di questa sezione viene descritto come utilizzare le risorse di Windows Presentation Foundation (WPF).

Contenuto della sezione

[Definire e fare riferimento a una risorsa](#)

[Usare le risorse delle applicazioni](#)

[Utilizzare la classe SystemFonts](#)

[Usare chiavi di caratteri del sistema](#)

[Utilizzare SystemParameters](#)

[Usare chiavi dei parametri di sistema](#)

Reference

[Resources](#)

[SystemColors](#)

[SystemParameters](#)

[SystemFonts](#)

Sezioni correlate

[Risorse XAML](#)

Procedura: definire e fare riferimento a una risorsa

04/11/2019 • 3 minutes to read • [Edit Online](#)

Questo esempio illustra come definire una risorsa e farvi riferimento usando un attributo in Extensible Application Markup Language (XAML).

Esempio

Nell'esempio seguente vengono definiti due tipi di risorse: una risorsa `SolidColorBrush` e diverse risorse `Style`. La risorsa `SolidColorBrush` `MyBrush` viene utilizzata per fornire il valore di diverse proprietà che accettano un valore di tipo `Brush`. Le risorse `Style` `PageBackground`, `TitleText` e `Label` ciascuna destinazione di un particolare tipo di controllo. Gli stili consentono di impostare una varietà di proprietà diverse sui controlli di destinazione, quando tale risorsa viene utilizzata come riferimento dalla chiave di risorsa e viene utilizzata per impostare la proprietà `Style` di diversi elementi di controllo specifici definiti in XAML.

Si noti che una delle proprietà all'interno dei setter dello stile `Label` fa riferimento anche alla risorsa `MyBrush` definita in precedenza. Si tratta di una tecnica comune, ma è importante ricordare che le risorse vengono analizzate e immesse in un dizionario risorse nell'ordine in cui vengono fornite. Le risorse vengono inoltre richieste dall'ordine trovato all'interno del dizionario se si usa l'[estensione di markup StaticResource](#) per farvi riferimento da un'altra risorsa. Assicurarsi che tutte le risorse a cui si fa riferimento siano definite in precedenza all'interno della raccolta di risorse rispetto a quella in cui viene richiesta la risorsa. Se necessario, è possibile aggirare l'ordine di creazione rigoroso dei riferimenti alle risorse utilizzando un'[estensione di markup DynamicResource](#) per fare riferimento alla risorsa in fase di esecuzione, ma è necessario tenere presente che questa tecnica `DynamicResource` presenta prestazioni conseguenze. Per informazioni dettagliate, vedere [risorse XAML](#).

```

<Page Name="root"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Page.Resources>
        <SolidColorBrush x:Key="MyBrush" Color="Gold"/>
        <Style TargetType="Border" x:Key="PageBackground">
            <Setter Property="Background" Value="Blue"/>
        </Style>
        <Style TargetType="TextBlock" x:Key="TitleText">
            <Setter Property="Background" Value="Blue"/>
            <Setter Property="DockPanel.Dock" Value="Top"/>
            <Setter Property="FontSize" Value="18"/>
            <Setter Property="Foreground" Value="#4E87D4"/>
            <Setter Property="FontFamily" Value="Trebuchet MS"/>
            <Setter Property="Margin" Value="0,40,10,10"/>
        </Style>
        <Style TargetType="TextBlock" x:Key="Label">
            <Setter Property="DockPanel.Dock" Value="Right"/>
            <Setter Property="FontSize" Value="8"/>
            <Setter Property="Foreground" Value="{StaticResource MyBrush}"/>
            <Setter Property="FontFamily" Value="Arial"/>
            <Setter Property="FontWeight" Value="Bold"/>
            <Setter Property="Margin" Value="0,3,10,0"/>
        </Style>
    </Page.Resources>
    <StackPanel>
        <Border Style="{StaticResource PageBackground}">
            <DockPanel>
                <TextBlock Style="{StaticResource TitleText}">Title</TextBlock>
                <TextBlock Style="{StaticResource Label}">Label</TextBlock>
                <TextBlock DockPanel.Dock="Top" HorizontalAlignment="Left" FontSize="36" Foreground="{StaticResource MyBrush}" Text="Text" Margin="20" />
                <Button DockPanel.Dock="Top" HorizontalAlignment="Left" Height="30" Background="{StaticResource MyBrush}" Margin="40">Button</Button>
                <Ellipse DockPanel.Dock="Top" HorizontalAlignment="Left" Width="100" Height="100" Fill="{StaticResource MyBrush}" Margin="40" />
            </DockPanel>
        </Border>
    </StackPanel>
</Page>

```

Vedere anche

- [Risorse XAML](#)
- [Applicazione di stili e modelli](#)

Procedura: Usare le risorse delle applicazioni

04/11/2019 • 2 minutes to read • [Edit Online](#)

In questo esempio viene illustrato come usare le risorse delle applicazioni.

Esempio

L'esempio seguente mostra un file di definizione dell'applicazione (ADF). Il file di definizione dell'applicazione definisce una sezione di risorse (un valore per la proprietà **Resources**). Le risorse definite a livello di applicazione sono accessibili da tutte le altre pagine che fanno parte dell'applicazione. In questo caso, la risorsa è uno stile dichiarato. Poiché uno stile completo che include un modello di controllo può essere lungo, questo esempio omette il modello di controllo definito all'interno del setter della proprietà **ContentTemplate** dello stile.

```
<Application.Resources>
    <Style TargetType="Button" x:Key="GelButton" >
        <Setter Property="Margin" Value="1,2,1,2"/>
        <Setter Property="HorizontalAlignment" Value="Left"/>
        <Setter Property="Template">
            <Setter.Value>
                ...
            </Setter.Value>
        </Setter>
    </Style>
</Application.Resources>
```

Nell'esempio seguente viene illustrata una pagina XAML che fa riferimento alla risorsa a livello di applicazione definita dall'esempio precedente. Si fa riferimento alla risorsa usando un' **estensione di markup StaticResource** che specifica la chiave di risorsa univoca per la risorsa richiesta. Nella pagina corrente non è stata rilevata alcuna risorsa con chiave "GelButton", pertanto l'ambito di ricerca per la risorsa richiesta continua oltre la pagina corrente e nelle risorse definite a livello di applicazione.

```
<StackPanel
    Name="root"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    >
    <Button Height="50" Width="250" Style="{StaticResource GelButton}" Content="Button 1" />
    <Button Height="50" Width="250" Style="{StaticResource GelButton}" Content="Button 2" />
</StackPanel>
```

Vedere anche

- [Risorse XAML](#)
- [Cenni preliminari sulla gestione di applicazioni](#)
- [Procedure relative alle proprietà](#)

Procedura: Usare la classe SystemFonts

08/01/2020 • 3 minutes to read • [Edit Online](#)

Questo esempio illustra come usare le risorse statiche della classe [SystemFonts](#) per applicare uno stile a un pulsante o personalizzarlo.

Esempio

Le risorse di sistema espongono diversi valori determinati dal sistema come risorse e proprietà per consentire la creazione di oggetti visivi coerenti con le impostazioni del sistema. [SystemFonts](#) è una classe che contiene sia i valori dei tipi di carattere di sistema come proprietà statiche che le proprietà che fanno riferimento a chiavi di risorsa che possono essere usate per accedere dinamicamente a tali valori in fase di esecuzione. Ad esempio, [CaptionFontFamily](#) è un valore [SystemFonts](#) e [CaptionFontFamilyKey](#) è una chiave di risorsa corrispondente.

In XAML, è possibile usare i membri di [SystemFonts](#) come proprietà statiche o riferimenti a risorse dinamiche (con il valore della proprietà statica come chiave). Usare un riferimento alla risorsa dinamica per aggiornare automaticamente la metrica del tipo di carattere durante l'esecuzione dell'applicazione; in caso contrario, usare un riferimento a un valore statico.

NOTE

Nelle chiavi di risorsa il suffisso "Key" viene aggiunto al nome della proprietà.

Nell'esempio seguente viene illustrato come accedere e utilizzare le proprietà di [SystemFonts](#) come valori statici per applicare uno stile a un pulsante o personalizzarlo. Questo esempio di markup assegna i valori [SystemFonts](#) a un pulsante.

```
<Button Margin="10, 10, 5, 5" Grid.Column="0" Grid.Row="3"
    FontSize="{x:Static SystemFonts.IconFontSize}"
    FontWeight="{x:Static SystemFonts.MessageFontWeight}"
    FontFamily="{x:Static SystemFonts.CaptionFontFamily}">
    SystemFonts
</Button>
```

Per usare i valori di [SystemFonts](#) nel codice, non è necessario usare un valore statico o un riferimento a una risorsa dinamica. Usare invece le proprietà non chiave della classe [SystemFonts](#). Anche se le proprietà non chiave sono apparentemente definite come proprietà statiche, il comportamento in fase di esecuzione di WPF come ospitato dal sistema rivaluterà le proprietà in tempo reale e configurerà correttamente le modifiche guidate dall'utente ai valori di sistema. L'esempio seguente illustra come specificare le impostazioni del tipo di carattere di un pulsante.

```
Button btncsharp = new Button();
btncsharp.Content = "SystemFonts";
btncsharp.Background = SystemColors.ControlDarkDarkBrush;
btncsharp.FontSize = SystemFonts.IconFontSize;
btncsharp.FontWeight = SystemFonts.MessageFontWeight;
btncsharp.FontFamily = SystemFonts.CaptionFontFamily;
cv1.Children.Add(btncsharp);
```

```
Dim btn As New Button()
btn.Content = "SystemFonts"
btn.Background = SystemColors.ControlDarkDarkBrush
btn.FontSize = SystemFonts.IconFontSize
btn.FontWeight = SystemFonts.MessageFontWeight
btn.FontFamily = SystemFonts.CaptionFontFamily
cv1.Children.Add(btn)
```

Vedere anche

- [SystemFonts](#)
- [Disegnare un'area con un pennello di sistema](#)
- [Utilizzare SystemParameters](#)
- [Usare chiavi di caratteri del sistema](#)
- [Procedure relative alle proprietà](#)
- [Estensione del markup x:Static](#)
- [Risorse XAML](#)
- [Estensione del markup DynamicResource](#)

Procedura: Usare le chiavi dei tipi di carattere del sistema

23/10/2019 • 2 minutes to read • [Edit Online](#)

Le risorse di sistema espongono diverse metriche di sistema come risorse per consentire agli sviluppatori di creare oggetti visivi coerenti con le impostazioni di sistema. [SystemFonts](#) è una classe che contiene sia i valori dei tipi di carattere del sistema che le risorse del tipo di carattere di [CaptionFontFamily](#) sistema [CaptionFontFamilyKey](#) che vengono associati ai valori, ad esempio e.

Le metriche dei tipi di carattere del sistema possono essere usate come risorse statiche o dinamiche. Usare una risorsa dinamica per aggiornare automaticamente la metrica del tipo di carattere durante l'esecuzione dell'applicazione; in caso contrario, usare una risorsa statica.

NOTE

Per le risorse dinamiche è stata aggiunta la parola *chiave Key* al nome della proprietà.

L'esempio seguente illustra come accedere alle risorse dinamiche dei tipi di carattere del sistema e usarle per applicare uno stile a un pulsante o personalizzarlo. In XAML questo esempio viene creato uno stile pulsante che [SystemFonts](#) assegna valori a un pulsante.

Esempio

```
<Style x:Key="SimpleFont" TargetType="{x:Type Button}">
    <Setter Property = "FontSize" Value= "{DynamicResource {x:Static SystemFonts.IconFontSizeKey}}"/>
    <Setter Property = "FontWeight" Value= "{DynamicResource {x:Static SystemFonts.MessageFontWeightKey}}"/>
    <Setter Property = "FontFamily" Value= "{DynamicResource {x:Static SystemFonts.CaptionFontFamilyKey}}"/>
</Style>
```

Vedere anche

- [Disegnare un'area con un pennello di sistema](#)
- [Utilizzare SystemParameters](#)
- [Utilizzare la classe SystemFonts](#)

Procedura: Usare la classe SystemParameters

23/10/2019 • 3 minutes to read • [Edit Online](#)

Questo esempio viene illustrato come accedere e usare le proprietà di [SystemParameters](#) per applicare uno stile a un pulsante o personalizzarlo.

Esempio

Le risorse di sistema espongono diverse impostazioni basate sul sistema come risorse per consentire la creazione di oggetti visivi coerenti con le impostazioni del sistema. [SystemParameters](#) è una classe contenente proprietà valore parametro di sistema e le chiavi di risorsa associate ai valori. Ad esempio, [FullPrimaryScreenHeight](#) è un [SystemParameters](#) il valore di proprietà e [FullPrimaryScreenHeightKey](#) è la chiave di risorsa corrispondente.

Nelle XAML, è possibile usare i membri di [SystemParameters](#) come utilizzo di una proprietà statica o di riferimenti di risorse dinamiche (con il valore della proprietà statica come chiave). Usare un riferimento alla risorsa dinamica per aggiornare automaticamente il valore basato sul sistema durante l'esecuzione dell'applicazione; in caso contrario, usare un riferimento statico. Le chiavi di risorsa hanno il suffisso `Key` accodato al nome della proprietà.

Nell'esempio seguente viene illustrato come accedere e usare i valori statici di [SystemParameters](#) per applicare uno stile a un pulsante o personalizzarlo. Questo esempio di markup viene ridimensionato un pulsante applicando [SystemParameters](#) valori a un pulsante.

```
<Button FontSize="8" Margin="10, 10, 5, 5" Grid.Column="0" Grid.Row="5"
    HorizontalAlignment="Left"
    Height="{x:Static SystemParameters.CaptionHeight}"
    Width="{x:Static SystemParameters.IconGridWidth}">
    SystemParameters
</Button>
```

Per usare i valori di [SystemParameters](#) nel codice, non è necessario usare riferimenti statici o riferimenti a risorse dinamiche. Usare invece i valori del [SystemParameters](#) classe. Anche se le proprietà non chiave sono apparentemente definite come proprietà statiche, il comportamento di runtime WPF ospitato dal sistema verranno rivaluterà le proprietà in tempo reale e risponderà correttamente eseguita dall'utente. le modifiche ai valori di sistema. Nell'esempio seguente viene illustrato come impostare la larghezza e altezza di un pulsante usando [SystemParameters](#) valori.

```
Button btnsharp = new Button();
btnsharp.Content = "SystemParameters";
btnsharp.FontSize = 8;
btnsharp.Background = SystemColors.ControlDarkDarkBrush;
btnsharp.Height = SystemParameters.CaptionHeight;
btnsharp.Width = SystemParameters.IconGridWidth;
cv2.Children.Add(btnsharp);
```

```
Dim btn As New Button()
btn.Content = "SystemParameters"
btn.FontSize = 8
btn.Background = SystemColors.ControlDarkDarkBrush
btn.Height = SystemParameters.CaptionHeight
btn.Width = SystemParameters.IconGridWidth
cv2.Children.Add(btn)
```

Vedere anche

- [SystemParameters](#)
- [Disegnare un'area con un pennello di sistema](#)
- [Utilizzare la classe SystemFonts](#)
- [Usare chiavi dei parametri di sistema](#)
- [Procedure relative alle proprietà](#)

Procedura: Usare le chiavi dei parametri di sistema

23/10/2019 • 2 minutes to read • [Edit Online](#)

Le risorse di sistema espongono diverse metriche di sistema come risorse per consentire agli sviluppatori di creare oggetti visivi coerenti con le impostazioni di sistema. [SystemParameters](#) è una classe che contiene sia i valori dei parametri di sistema che le chiavi di risorsa associate ai valori, [FullPrimaryScreenHeight](#) ad [FullPrimaryScreenHeightKey](#) esempio e. Le metriche dei parametri di sistema possono essere usate come risorse statiche o dinamiche. Usare una risorsa dinamica per aggiornare automaticamente le metriche dei parametri durante l'esecuzione dell'applicazione; in caso contrario, usare una risorsa statica.

NOTE

Per le risorse dinamiche è stata aggiunta la parola *chiave Key* al nome della proprietà.

L'esempio seguente illustra come accedere alle risorse dinamiche dei parametri di sistema e usarle per applicare uno stile a un pulsante o personalizzarlo. Questo XAML esempio consente di ridimensionare un pulsante [SystemParameters](#) assegnando valori alla larghezza e all'altezza del pulsante.

Esempio

```
<Style x:Key="SimpleParam" TargetType="{x:Type Button}">
    <Setter Property = "Height" Value= "{DynamicResource {x:Static SystemParameters.CaptionHeightKey}}"/>
    <Setter Property = "Width" Value= "{DynamicResource {x:Static SystemParameters.IconGridWidthKey}}"/>
</Style>
```

Vedere anche

- [Disegnare un'area con un pennello di sistema](#)
- [Utilizzare la classe SystemFonts](#)
- [Utilizzare SystemParameters](#)

Documenti

23/10/2019 • 2 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) fornisce un insieme versatile di componenti che consentono agli sviluppatori di compilare applicazioni con funzionalità avanzate del documento e migliorano l'esperienza di lettura. Oltre a caratteristiche avanzate in termini di funzionalità e qualità, Windows Presentation Foundation (WPF) assicura servizi di gestione semplificati per la creazione di pacchetti, la sicurezza e l'archiviazione dei documenti.

In questa sezione

[Documenti in WPF](#)

[Serializzazione e archiviazione di documenti](#)

[Annotazioni](#)

[Contenuto del flusso](#)

[Tipografia](#)

[Stampa e gestione di sistemi di stampa](#)

Vedere anche

- [DocumentViewer](#)
- [FlowDocument](#)
- [System.Windows.Xps](#)
- [isXPS.exe \(strumento di conformità isXPS\)](#)

Documenti in WPF

03/02/2020 • 19 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) offre un'ampia gamma di funzionalità dei documenti che consentono la creazione di contenuto ad alta fedeltà progettato per essere più facilmente accessibile e leggibile rispetto alle generazioni precedenti di Windows. Oltre a caratteristiche avanzate in termini di funzionalità e qualità, WPF assicura servizi integrati per la visualizzazione, la creazione di pacchetti e la sicurezza dei documenti. Questo argomento costituisce un'introduzione ai tipi di documenti e alla creazione di pacchetti di WPF.

Tipi di documenti

In WPF i documenti sono suddivisi in due categorie generali in base all'uso previsto: documenti statici e documenti dinamici.

I documenti corretti sono destinati ad applicazioni che richiedono una presentazione "What You See is what you get" (WYSIWYG), indipendentemente dall'hardware di visualizzazione o della stampante usato. Tra gli usi tipici dei documenti statici rientrano le attività di desktop publishing, elaborazione di testi e layout di form, in cui la corrispondenza alla progettazione della pagina originale è essenziale. Come parte del layout, in un documento statico viene mantenuta la disposizione precisa degli elementi di contenuto indipendentemente dal dispositivo di visualizzazione o di stampa in uso. Ad esempio, la pagina di un documento statico visualizzata con un'impostazione di 96 dpi non varia quando viene stampata con una stampante laser a 600 dpi o con un fotocompositore a 4800 dpi. Il layout di pagina rimane invariato in ogni caso, benché la qualità del documento dipenda dalle capacità del dispositivo usato.

D'altro canto, i documenti dinamici sono progettati per ottimizzare la visualizzazione e la leggibilità e sono particolarmente adatti quando la facilità di lettura è il principale requisito d'uso per il documento. Anziché essere impostati su un layout predefinito, questi documenti consentono di adattare e ridisporre il contenuto in modo dinamico in base alle variabili in fase di esecuzione, ad esempio, le dimensioni della finestra, la risoluzione del dispositivo e le preferenze facoltative dell'utente. Una pagina Web è un esempio semplice di documento dinamico in cui il contenuto della pagina viene formattato dinamicamente per adattarsi alla finestra corrente. I documenti dinamici ottimizzano l'esperienza di visualizzazione e di lettura per l'utente, in base all'ambiente di runtime. Ad esempio, lo stesso documento dinamico verrà riformattato in modo dinamico per una leggibilità ottimale sia su dispositivi di visualizzazione da 19 pollici ad alta risoluzione che su piccoli schermi PDA da 2x3 pollici. Inoltre, i documenti dinamici dispongono di numerose funzionalità incorporate tra cui ricerca, modalità di visualizzazione che ottimizzano la leggibilità e possibilità di modificare le dimensioni e l'aspetto dei tipi di carattere. Vedere [Cenni preliminari sui documenti dinamici](#) per immagini, esempi e informazioni approfondite sui documenti dinamici.

Controlli dei documenti e layout del testo

Il .NET Framework fornisce un set di controlli predefiniti che semplificano l'uso di documenti fissi, documenti dinamici e testo generale all'interno dell'applicazione. La visualizzazione del contenuto del documento fisso è supportata tramite il controllo [DocumentViewer](#). La visualizzazione del contenuto dei documenti dinamici è supportata da tre controlli diversi: [FlowDocumentReader](#), [FlowDocumentPageViewere](#) [FlowDocumentScrollView](#) che vengono mappati a diversi scenari utente (vedere le sezioni seguenti). Altri controlli di WPF forniscono un layout semplificato per supportare l'uso di testo generico (vedere la sezione [Testo nell'interfaccia utente](#) più avanti).

Controllo dei documenti statici: [DocumentViewer](#)

Il controllo [DocumentViewer](#) è progettato per visualizzare [FixedDocument](#) contenuto. Il controllo [DocumentViewer](#) fornisce un'interfaccia utente intuitiva che offre supporto incorporato per operazioni comuni, tra cui l'output di stampa, la copia negli Appunti, lo zoom e le funzionalità di ricerca del testo. Il controllo consente di accedere a pagine di contenuto attraverso un meccanismo di scorrimento intuitivo. Come tutti i controlli WPF, [DocumentViewer](#) supporta il restyling completo o parziale, che consente di integrare visivamente il controllo in qualsiasi applicazione o ambiente.

[DocumentViewer](#) è progettato per visualizzare il contenuto in modalità di sola lettura. la modifica o la modifica del contenuto non è disponibile e non è supportata.

Controlli dei documenti dinamici

NOTE

Per informazioni più dettagliate sulle funzionalità dei documenti dinamici e su come crearle, vedere [Cenni preliminari sui documenti dinamici](#).

La visualizzazione del contenuto dei documenti dinamici è supportata da tre controlli: [FlowDocumentReader](#), [FlowDocumentPageViewer](#) e [FlowDocumentScrollView](#).

FlowDocumentReader

[FlowDocumentReader](#) include funzionalità che consentono all'utente di scegliere in modo dinamico tra le varie modalità di visualizzazione, inclusa una modalità di visualizzazione a pagina singola (pagina alla volta), una modalità di visualizzazione a due pagine alla volta (formato lettura libro) e una modalità di visualizzazione a scorrimento continuo (senza fine). Per ulteriori informazioni su queste modalità di visualizzazione, vedere [FlowDocumentReaderViewingMode](#). Se non è necessario passare dinamicamente da una modalità di visualizzazione all'altra, [FlowDocumentPageViewer](#) e [FlowDocumentScrollView](#) forniscono visualizzatori di contenuto del flusso più semplici che sono corretti in una particolare modalità di visualizzazione.

FlowDocumentPageViewer e FlowDocumentScrollView

[FlowDocumentPageViewer](#) Visualizza il contenuto nella modalità di visualizzazione pagina alla volta, mentre [FlowDocumentScrollView](#) Visualizza il contenuto in modalità di scorrimento continuo. Sia [FlowDocumentPageViewer](#) che [FlowDocumentScrollView](#) sono corretti a una particolare modalità di visualizzazione. Confrontare con [FlowDocumentReader](#), che include funzionalità che consentono all'utente di scegliere dinamicamente tra le varie modalità di visualizzazione (fornite dall'enumerazione [FlowDocumentReaderViewingMode](#)), al costo di un maggior utilizzo di risorse rispetto a [FlowDocumentPageViewer](#) o [FlowDocumentScrollView](#).

Per impostazione predefinita, viene sempre visualizzata una barra di scorrimento verticale e in caso di necessità diventa visibile una barra di scorrimento orizzontale. Il Interfaccia utente predefinito per [FlowDocumentScrollView](#) non include una barra degli strumenti; Tuttavia, la proprietà [IsToolBarVisible](#) può essere utilizzata per abilitare una barra degli strumenti incorporata.

Testo nell'interfaccia utente

Oltre all'aggiunta di testo ai documenti, è ovviamente possibile usare il testo nell'interfaccia utente delle applicazioni, ad esempio nei form. WPF include più controlli per la creazione di testo sullo schermo. Ogni controllo è destinato a uno scenario diverso e dispone di un proprio elenco di funzionalità e limitazioni. In generale, l'elemento [TextBlock](#) deve essere utilizzato quando è necessario il supporto di testo limitato, ad esempio una breve frase in una interfaccia utente. [Label](#) può essere utilizzato quando è richiesto un supporto di testo minimo. Per altre informazioni, vedere [Cenni preliminari sul controllo TextBlock](#).

Creazione di pacchetti di documenti

Le API [System.IO.Packaging](#) offrono un modo efficiente per organizzare i dati delle applicazioni, il contenuto dei documenti e le risorse correlate in un unico contenitore semplice da accedere, portatile e facile da

distribuire. Un file ZIP è un esempio di [Package](#) tipo in grado di contenere più oggetti come singola unità. Le API per la creazione di pacchetti forniscono un'implementazione di [ZipPackage](#) predefinita progettata usando uno standard Open Packaging Conventions con l'architettura di file XML e ZIP. Le API WPF Packaging semplificano la creazione di pacchetti e l'archiviazione e l'accesso agli oggetti in essi contenuti. Un oggetto archiviato in un [Package](#) viene definito come [PackagePart](#) ("parte"). I pacchetti possono anche includere certificati digitali firmati utilizzabili per identificare l'autore di una parte e per confermare che il contenuto di un pacchetto non è stato modificato. I pacchetti includono inoltre una funzionalità [PackageRelationship](#) che consente di aggiungere informazioni aggiuntive a un pacchetto o associate a parti specifiche senza modificare effettivamente il contenuto delle parti esistenti. I servizi pacchetti supportano anche Microsoft Windows Rights Management (RM).

L'architettura dei pacchetti di WPF costituisce la base per numerose tecnologie chiave:

- Documenti XPS conformi a XPS (XML Paper Specification).
- Documenti in formato XML aperto di Microsoft Office "12" (con estensione docx).
- Formati di archiviazione personalizzati per la progettazione delle applicazioni.

In base alle API per la creazione di pacchetti, un [XpsDocument](#) è progettato appositamente per archiviare WPF documenti di contenuto fissi. Un [XpsDocument](#) è un documento autonomo che può essere aperto in un visualizzatore, visualizzato in un controllo [DocumentViewer](#), instradato a uno spool di stampa o inviato direttamente a una stampante compatibile con XPS.

Le sezioni seguenti forniscono informazioni aggiuntive sulle API [Package](#) e [XpsDocument](#) fornite con WPF.

Componenti dei pacchetti

Le API per la creazione di pacchetti di WPF consentono l'organizzazione dei dati delle applicazioni e dei documenti in un'unica unità portatile. Uno dei tipi di pacchetti più comuni è il file ZIP, il tipo predefinito fornito con WPF. [Package](#) è una classe astratta da cui viene implementato [ZipPackage](#) mediante un'architettura di file XML e ZIP standard aperta. Il metodo [Open](#) utilizza [ZipPackage](#) per creare e utilizzare file ZIP per impostazione predefinita. Un pacchetto può contenere tre tipi di elementi di base:

PackagePart	Contenuto delle applicazioni, dati, documenti e file di risorse.
PackageDigitalSignature	[Certificato x.509] per l'identificazione, l'autenticazione e la convalida.
PackageRelationship	Informazioni aggiunte relative al pacchetto o a una parte specifica.

[PackagePart](#)

Un [PackagePart](#) ("parte") è una classe astratta che fa riferimento a un oggetto archiviato in un [Package](#). In un file ZIP le parti del pacchetto corrispondono ai singoli file archiviati nel file ZIP. [ZipPackagePart](#) fornisce l'implementazione predefinita per gli oggetti serializzabili archiviati in un [ZipPackage](#). Analogamente a un file system, le parti contenute nel pacchetto vengono archiviate secondo un'organizzazione di directory o cartelle di tipo gerarchico. Utilizzando le API WPF Packaging, le applicazioni possono scrivere, archiviare e leggere più oggetti [PackagePart](#) utilizzando un singolo contenitore di file ZIP.

[PackageDigitalSignature](#)

Per la sicurezza, un [PackageDigitalSignature](#) ("firma digitale") può essere associato a parti all'interno di un pacchetto. Un [PackageDigitalSignature](#) incorpora un [509] che fornisce due funzionalità:

1. Identifica e autentica l'autore della parte.

2. Conferma che la parte non è stata modificata.

La firma digitale non impedisce che una parte venga modificata, ma un controllo di convalida della firma digitale avrà esito negativo se la parte è stata in qualche modo modificata. L'applicazione potrà quindi intraprendere l'azione appropriata, ad esempio bloccare l'apertura della parte o notificare all'utente che la parte è stata modificata e non è sicura.

PackageRelationship

Un [PackageRelationship](#) ("Relationship") fornisce un meccanismo per associare informazioni aggiuntive al pacchetto o a una parte all'interno del pacchetto. Una relazione è una funzionalità a livello di pacchetto in grado di associare informazioni aggiuntive a una parte senza modificarne il contenuto effettivo. L'inserimento di nuovi dati direttamente nel contenuto in genere non è un'operazione pratica:

- Il tipo effettivo della parte e il relativo schema di contenuto non sono noti.
- Anche se è noto, lo schema di contenuto potrebbe non offrire un metodo per l'aggiunta di nuove informazioni.
- La parte potrebbe essere crittografata o firmata digitalmente, condizioni che impediscono qualsiasi modifica.

Le relazioni dei pacchetti offrono un metodo individuabile per aggiungere e associare altre informazioni a singole parti o all'intero pacchetto. Le relazioni dei pacchetti vengono usate per due funzioni principali:

1. Definizione delle relazioni di dipendenza tra le parti.
2. Definizione delle relazioni delle informazioni che aggiungono note o altri dati correlati alla parte.

Un [PackageRelationship](#) fornisce un metodo rapido e individuabile per definire le dipendenze e aggiungere altre informazioni associate a una parte del pacchetto o al pacchetto nel suo complesso.

Relazioni di dipendenza

Le relazioni di dipendenza vengono usate per descrivere le dipendenze tra le parti. Ad esempio, un pacchetto può contenere una parte HTML che include uno o più tag immagine . I tag immagine fanno riferimento a immagini posizionate come altre parti interne o esterne al pacchetto (ad esempio accessibili tramite Internet). La creazione di un [PackageRelationship](#) associato al file HTML rende l'individuazione e l'accesso alle risorse dipendenti in modo rapido e semplice. Un'applicazione browser o del visualizzatore può accedere direttamente alle relazioni tra le parti e avviare immediatamente l'assemblaggio delle risorse dipendenti senza conoscere lo schema, né analizzare il documento.

Relazioni delle informazioni

Analogamente a una nota o a un'annotazione, un [PackageRelationship](#) può essere utilizzato anche per archiviare altri tipi di informazioni da associare a una parte senza dover modificare effettivamente il contenuto della parte.

Documenti XPS

Il documento XPS (XML Paper Specification) è un pacchetto che contiene uno o più documenti fissi insieme a tutte le risorse e le informazioni necessarie per il rendering. XPS è anche il formato nativo di file spooler di stampa di Windows Vista. Un [XpsDocument](#) viene archiviato nel set di dati ZIP standard e può includere una combinazione di componenti XML e binari, ad esempio file di immagine e di tipi di carattere. Gli oggetti [PackageRelationship](#) vengono usati per definire le dipendenze tra il contenuto e le risorse necessarie per eseguire il rendering completo del documento. La progettazione di [XpsDocument](#) offre un'unica soluzione di documento ad alta fedeltà che supporta più utilizzi:

- Lettura, scrittura e archiviazione del contenuto di documenti statici e risorse come un file singolo, portatile e facile da distribuire.

- Visualizzazione di documenti con l'applicazione XPS Viewer.
- Output dei documenti nel formato di output dello spooler di stampa nativo di Windows Vista.
- Routing di documenti direttamente a una stampante compatibile con XPS.

Vedere anche

- [FixedDocument](#)
- [FlowDocument](#)
- [XpsDocument](#)
- [ZipPackage](#)
- [ZipPackagePart](#)
- [PackageRelationship](#)
- [DocumentViewer](#)
- [Testo](#)
- [Cenni preliminari sui documenti dinamici](#)
- [Panoramica della stampa](#)
- [Serializzazione e archiviazione di documenti](#)

Serializzazione e archiviazione di documenti

10/02/2020 • 12 minutes to read • [Edit Online](#)

Microsoft .NET Framework fornisce un ambiente potente per la creazione e la visualizzazione di documenti di alta qualità. Funzionalità avanzate che supportano sia documenti fissi che documenti di flusso, controlli di visualizzazione avanzati, combinati con potenti funzionalità grafiche 2D e 3D, .NET Framework applicazioni a un nuovo livello di qualità e esperienza utente. La possibilità di gestire in modo flessibile una rappresentazione in memoria di un documento è una funzionalità chiave di .NET Framework e la possibilità di salvare e caricare in modo efficiente i documenti da un archivio dati è la necessità di quasi tutte le applicazioni. Il processo di conversione di un documento da una rappresentazione in memoria interna a un archivio dati esterno è detto serializzazione. Il processo inverso di lettura di un archivio dati e di creazione dell'istanza in memoria originale è detto deserializzazione.

Informazioni sulla serializzazione di documenti

In teoria, il processo di serializzazione e deserializzazione di un documento in memoria è trasparente per l'applicazione. L'applicazione chiama un metodo di scrittura del serializzatore per salvare il documento, mentre un metodo di lettura del deserializzatore accede all'archivio dati e ricrea l'istanza originale in memoria. Il formato specifico in cui vengono archiviati i dati non costituisce in genere un problema per l'applicazione, purché il processo di serializzazione e deserializzazione ricreia il documento nel formato originale.

Nelle applicazioni sono spesso disponibili varie opzioni di serializzazione che consentono all'utente di salvare i documenti in supporti o formati diversi. Ad esempio, è possibile che in un'applicazione siano disponibili opzioni "Salva con nome" per l'archiviazione di un documento su disco, in un database o come servizio Web. Allo stesso modo, serializzatori diversi potrebbero archiviare il documento in formati diversi quali HTML, RTF, XML, XPS o, in alternativa, in un formato di terze parti. Per l'applicazione, la serializzazione definisce un'interfaccia che isola i dettagli del supporto di archiviazione nell'implementazione di ogni specifico serializzatore. Oltre ai vantaggi derivanti dall'incapsulamento dei dettagli di archiviazione, le API .NET Framework [System.Windows.Documents.Serialization](#) forniscono molte altre importanti funzionalità.

Funzionalità dei serializzatori di documenti di .NET Framework 3.0

- L'accesso diretto agli oggetti documento di alto livello (albero logico e oggetti visivi) consente di archiviare in modo efficiente contenuto impaginato, elementi 2D/3D, immagini, supporti, collegamenti ipertestuali, annotazioni e altro contenuto di supporto.
- Operazioni sincrone e asincrone.
- Supporto per serializzatori plug-in con funzionalità avanzate:
 - Accesso a livello di sistema per l'uso da tutte le applicazioni .NET Framework.
 - Individuazione semplice dei plug-in dell'applicazione.
 - Facilità di distribuzione, installazione e aggiornamento per i plug-in di terze parti personalizzati.
 - Supporto dell'interfaccia utente per impostazioni e opzioni di runtime personalizzate.

Percorso di stampa XPS

Il percorso di stampa di Microsoft .NET Framework XPS fornisce anche un meccanismo estensibile per la scrittura di documenti tramite l'output di stampa. XPS funge sia da formato di file di documento sia dal formato nativo dello spooler di stampa per Windows Vista. I documenti XPS possono essere inviati direttamente alle stampanti compatibili con XPS senza dover eseguire la conversione in un formato intermedio. Vedere [Cenni preliminari sulla](#)

[stampa](#) per altre informazioni sulle opzioni e le funzionalità dell'output del percorso di stampa.

Serializzatori plug-in

Le API [System.Windows.Documents.Serialization](#) forniscono supporto sia per i serializzatori plug-in che per i serializzatori collegati installati separatamente dall'applicazione, vengono associati in fase di esecuzione e sono accessibili tramite il meccanismo di individuazione [SerializerProvider](#). I serializzatori plug-in offrono notevoli vantaggi in termini di facilità di distribuzione e uso a livello di sistema. I serializzatori collegati possono essere implementati anche per ambienti con attendibilità parziale, ad esempio le applicazioni browser XAML (XBAPs), in cui i serializzatori plug-in non sono accessibili. I serializzatori collegati, basati su un'implementazione derivata della classe [SerializerWriter](#), vengono compilati e collegati direttamente nell'applicazione. Sia i serializzatori plug-in che i serializzatori collegati funzionano tramite eventi e metodi pubblici identici che semplificano l'uso di uno o di entrambi i tipi di serializzatori nella stessa applicazione.

I serializzatori plug-in garantiscono agli sviluppatori di applicazioni estensibilità per nuovi progetti di archiviazione e formati di file, senza la necessità di scrivere direttamente codice per ogni potenziale formato in fase di compilazione. I serializzatori plug-in sono utili anche agli sviluppatori di terze parti in quanto forniscono un metodo standardizzato per distribuire, installare e aggiornare plug-in accessibili al sistema per formati di file personalizzati o proprietari.

Uso di un serializzatore plug-in

I serializzatori plug-in sono semplici da usare. La classe [SerializerProvider](#) enumera un oggetto [SerializerDescriptor](#) per ogni plug-in installato nel sistema. La proprietà [IsLoadable](#) filtra i plug-in installati in base alla configurazione corrente e verifica che il serializzatore possa essere caricato e utilizzato dall'applicazione. Il [SerializerDescriptor](#) fornisce anche altre proprietà, ad esempio [DisplayName](#) e [DefaultFileExtension](#), che l'applicazione può usare per richiedere all'utente di selezionare un serializzatore per un formato di output disponibile. Un serializzatore plug-in predefinito per XPS viene fornito con .NET Framework ed è sempre enumerato. Quando l'utente seleziona un formato di output, viene usato il metodo [CreateSerializerWriter](#) per creare un [SerializerWriter](#) per il formato specifico. È quindi possibile chiamare il metodo [SerializerWriter.Write](#) per restituire il flusso del documento nell'archivio dati.

Nell'esempio seguente viene illustrata un'applicazione che utilizza il metodo [SerializerProvider](#) in una proprietà "PlugInFileFilter". PlugInFileFilter enumera i plug-in installati e compila una stringa di filtro con le opzioni del file disponibili per un [SaveFileDialog](#).

```

// ----- PlugInFileFilter -----
/// <summary>
/// Gets a filter string for installed plug-in serializers.</summary>
/// <remark>
/// PlugInFileFilter is used to set the SaveFileDialog or
/// OpenFileDialog "Filter" property when saving or opening files
/// using plug-in serializers.</remark>
private string PlugInFileFilter
{
    get
    {
        // Create a SerializerProvider for accessing plug-in serializers.
        SerializerProvider serializerProvider = new SerializerProvider();
        string filter = "";

        // For each loadable serializer, add its display
        // name and extension to the filter string.
        foreach (SerializerDescriptor serializerDescriptor in
            serializerProvider.InstalledSerializers)
        {
            if (serializerDescriptor.IsLoadable)
            {
                // After the first, separate entries with a "|".
                if (filter.Length > 0)    filter += "|";

                // Add an entry with the plug-in name and extension.
                filter += serializerDescriptor.DisplayName + " (" +
                    serializerDescriptor.DefaultFileExtension + ")|*" +
                    serializerDescriptor.DefaultFileExtension;
            }
        }

        // Return the filter string of installed plug-in serializers.
        return filter;
    }
}

```

Dopo che un nome di file di output è stato selezionato dall'utente, nell'esempio seguente viene illustrato l'utilizzo del metodo [CreateSerializerWriter](#) per archiviare un documento specificato in un formato specificato.

```

// Create a SerializerProvider for accessing plug-in serializers.
SerializerProvider serializerProvider = new SerializerProvider();

// Locate the serializer that matches the fileName extension.
SerializerDescriptor selectedPlugIn = null;
foreach ( SerializerDescriptor serializerDescriptor in
            serializerProvider.InstalledSerializers )
{
    if ( serializerDescriptor.IsLoadable &&
        fileName.EndsWith(serializerDescriptor.DefaultFileExtension) )
    { // The plug-in serializer and fileName extensions match.
        selectedPlugIn = serializerDescriptor;
        break; // foreach
    }
}

// If a match for a plug-in serializer was found,
// use it to output and store the document.
if (selectedPlugIn != null)
{
    Stream package = File.Create(fileName);
    SerializerWriter serializerWriter =
        serializerProvider.CreateSerializerWriter(selectedPlugIn,
                                                package);
    IDocumentPaginatorSource idoc =
        flowDocument as IDocumentPaginatorSource;
    serializerWriter.Write(idoc.DocumentPaginator, null);
    package.Close();
    return true;
}

```

Installazione di serializzatori plug-in

La classe [SerializerProvider](#) fornisce l'interfaccia dell'applicazione di livello superiore per l'individuazione e l'accesso del serializzatore plug-in. [SerializerProvider](#) individua e fornisce all'applicazione un elenco dei serializzatori installati e accessibili nel sistema. Le specifiche dei serializzatori installati vengono definite tramite le impostazioni del Registro di sistema. I serializzatori plug-in possono essere aggiunti al registro di sistema tramite il metodo [RegisterSerializer](#). Se .NET Framework non è ancora installato, lo script di installazione del plug-in può impostare direttamente i valori del registro di sistema. Il metodo [UnregisterSerializer](#) può essere utilizzato per rimuovere un plug-in installato in precedenza oppure è possibile reimpostare le impostazioni del registro di sistema in modo analogo tramite uno script di disinstallazione.

Creazione di un serializzatore plug-in

Sia i serializzatori plug-in che i serializzatori collegati usano gli stessi eventi e metodi pubblici esposti e possono essere progettati in modo simile per funzionare in modalità sincrona o asincrona. Per creare un serializzatore plug-in, sono in genere necessari tre passaggi di base:

1. Implementare ed eseguire il debug del serializzatore innanzitutto come serializzatore collegato. La creazione iniziale di un serializzatore compilato e collegato direttamente in un'applicazione di prova consente l'accesso completo ai punti di interruzione e ad altri servizi di debug utili per il test.
2. Quando il serializzatore viene testato completamente, viene aggiunta un'interfaccia [ISerializerFactory](#) per creare un plug-in. L'interfaccia [ISerializerFactory](#) consente l'accesso completo a tutti gli oggetti .NET Framework che includono l'albero logico, gli oggetti [UIElement](#), [IDocumentPaginatorSource](#) gli elementi [Visual](#). Inoltre [ISerializerFactory](#) fornisce gli stessi metodi ed eventi sincroni e asincroni utilizzati dai serializzatori collegati. Poiché l'output dei documenti di grandi dimensioni può richiedere tempo, le operazioni asincrone sono consigliate per garantire l'interazione dell'utente e fornire un'opzione "Annulla" se si verifica un problema relativo all'archivio dati.
3. Dopo la creazione del serializzatore plug-in, viene implementato uno script di installazione per la

distribuzione, l'installazione e la disinstallazione del plug-in (vedere la sezione precedente "[Installazione di serializzatori plug-in](#)").

Vedere anche

- [System.Windows.Documents.Serialization](#)
- [XpsDocumentWriter](#)
- [XpsDocument](#)
- [Documenti in WPF](#)
- [Panoramica della stampa](#)
- [XML Paper Specification](#)

Annotazioni

23/10/2019 • 2 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) fornisce controlli che supportano l'annotazione del contenuto dei documenti di visualizzazione del documento.

In questa sezione

[Cenni preliminari sulle annotazioni](#)

[Schema annotazioni](#)

Riferimenti

[Annotation](#)

[AnnotationService](#)

[DocumentViewer](#)

Sezioni correlate

[Documenti in WPF](#)

[Cenni preliminari sui documenti dinamici](#)

Cenni preliminari sulle annotazioni

07/01/2020 • 8 minutes to read • [Edit Online](#)

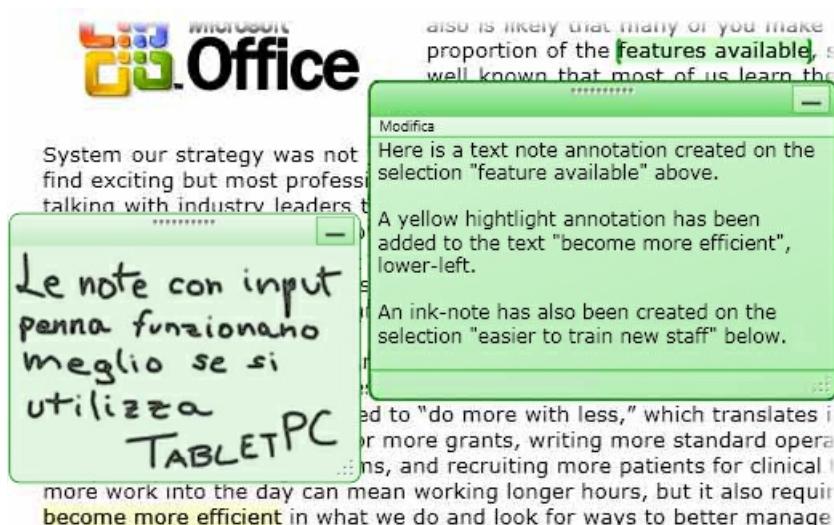
La scrittura di note o commenti su documenti cartacei è un'attività comune che diamo quasi per scontata. Queste note o commenti sono "annotazioni" aggiunte a un documento per contrassegnare informazioni o evidenziare elementi di interesse a cui fare riferimento in un secondo momento. Sebbene la scrittura di note su documenti stampati sia un'operazione semplice e comune, la capacità di aggiungere commenti personali ai documenti elettronici, se disponibile, è in genere molto limitata.

In questo argomento vengono esaminati diversi tipi comuni di annotazioni, in particolare note e evidenziazioni e viene illustrato il modo in cui il framework delle annotazioni Microsoft semplifica questi tipi di annotazioni nelle applicazioni tramite il Windows Presentation Foundation (WPF) controlli di visualizzazione del documento. I controlli di visualizzazione dei documenti WPF che supportano le annotazioni includono [FlowDocumentReader](#) e [FlowDocumentScrollView](#), nonché controlli derivati da [DocumentViewerBase](#) come [DocumentViewer](#) e [FlowDocumentPageViewer](#).

Memo

Normalmente una nota adesiva contiene informazioni scritte su un pezzetto di carta colorata che viene successivamente "attaccato" su un documento. Le note adesive digitali forniscono funzionalità simili per i documenti elettronici, ma con la flessibilità aggiuntiva per includere molti altri tipi di contenuto, ad esempio il testo tipizzato, le note scritte a mano (ad esempio, i tratti di "input penna" di Tablet PC) o i collegamenti Web.

La figura seguente mostra alcuni esempi di annotazioni con evidenziatore, annotazioni di testo con Memo e annotazioni a penna di Memo.



L'esempio seguente illustra il metodo che è possibile usare per abilitare il supporto delle annotazioni nell'applicazione.

```

// ----- StartAnnotations -----
/// <summary>
///   Enables annotations and displays all that are viewable.</summary>
private void StartAnnotations()
{
    // If there is no AnnotationService yet, create one.
    if (_annotService == null)
        // docViewer is a document viewing control named in Window1.xaml.
        _annotService = new AnnotationService(docViewer);

    // If the AnnotationService is currently enabled, disable it.
    if (_annotService.IsEnabled == true)
        _annotService.Disable();

    // Open a stream to the file for storing annotations.
    _annotStream = new FileStream(
        _annotStorePath, FileMode.OpenOrCreate, FileAccess.ReadWrite);

    // Create an AnnotationStore using the file stream.
    _annotStore = new XmlStreamStore(_annotStream);

    // Enable the AnnotationService using the new store.
    _annotService.Enable(_annotStore);
} // end:StartAnnotations()

```

```

' ----- StartAnnotations -----
''' <summary>
'''   Enables annotations and displays all that are viewable.</summary>
Private Sub StartAnnotations()
    ' If there is no AnnotationService yet, create one.
    If _annotService Is Nothing Then
        ' docViewer is a document viewing control named in Window1.xaml.
        _annotService = New AnnotationService(docViewer)
    End If

    ' If the AnnotationService is currently enabled, disable it.
    If _annotService.IsEnabled = True Then
        _annotService.Disable()
    End If

    ' Open a stream to the file for storing annotations.
    _annotStream = New FileStream(_annotStorePath, FileMode.OpenOrCreate, FileAccess.ReadWrite)

    ' Create an AnnotationStore using the file stream.
    _annotStore = New XmlStreamStore(_annotStream)

    ' Enable the AnnotationService using the new store.
    _annotService.Enable(_annotStore)
End Sub

```

Evidenziazioni

Quando si eseguono annotazioni su un documento cartaceo, per attirare l'attenzione su elementi di interesse si usano metodi creativi, ad esempio sottolineando, evidenziando, cerchiando parole in una frase o tracciando segni o notazioni sul margine. Le annotazioni evidenziate in Microsoft Annotations Framework forniscono una funzionalità simile per contrassegnare le informazioni visualizzate nei controlli di visualizzazione dei documenti WPF.

La figura seguente mostra un esempio di annotazione con evidenziatore.

Evidenziato Il framework di annotazione Microsoft semplifica l'aggiunta di funzionalità di annotazione a quasi tutte le applicazioni.

Gli utenti in genere creano annotazioni selezionando prima di tutto un testo o un elemento di interesse, quindi facendo clic con il pulsante destro del mouse per visualizzare un [ContextMenu](#) di opzioni di annotazione. Nell'esempio seguente viene illustrata la Extensible Application Markup Language (XAML) è possibile utilizzare per dichiarare una [ContextMenu](#) con i comandi indirizzati a cui gli utenti possono accedere per creare e gestire le annotazioni.

```
<DocumentViewer.ContextMenu>
  <ContextMenu>
    <MenuItem Command="ApplicationCommands.Copy" />
    <Separator />
    <!-- Add a Highlight annotation to a user selection. -->
    <MenuItem Command="ann:AnnotationService.CreateHighlightCommand"
      Header="Add Highlight" />
    <!-- Add a Text Note annotation to a user selection. -->
    <MenuItem Command="ann:AnnotationService.CreateTextStickyNoteCommand"
      Header="Add Text Note" />
    <!-- Add an Ink Note annotation to a user selection. -->
    <MenuItem Command="ann:AnnotationService.CreateInkStickyNoteCommand"
      Header="Add Ink Note" />
    <Separator />
    <!-- Remove Highlights from a user selection. -->
    <MenuItem Command="ann:AnnotationService.ClearHighlightsCommand"
      Header="Remove Highlights" />
    <!-- Remove Text Notes and Ink Notes from a user selection. -->
    <MenuItem Command="ann:AnnotationService.DeleteStickyNotesCommand"
      Header="Remove Notes" />
    <!-- Remove Highlights, Text Notes, Ink Notes from a selection. -->
    <MenuItem Command="ann:AnnotationService.DeleteAnnotationsCommand"
      Header="Remove Highlights &amp; Notes" />
  </ContextMenu>
</DocumentViewer.ContextMenu>
```

Ancoraggio dei dati

Il framework delle annotazioni associa le annotazioni ai dati selezionati dall'utente, non solo a una posizione nella visualizzazione. Pertanto, se la visualizzazione del documento cambia, ad esempio quando l'utente scorre o ridimensiona la finestra di visualizzazione, l'annotazione rimane nella selezione dati alla quale è associata. Ad esempio, l'immagine seguente mostra un'annotazione effettuata dall'utente su una selezione di testo. Quando la visualizzazione del documento cambia (scorre, viene ridimensionata o si sposta), l'annotazione con evidenziatore si sposta insieme alla selezione dati originale.

Evidenziato

Il framework di annotazione Microsoft semplifica l'aggiunta di funzionalità di annotazione a quasi tutte le applicazioni.

Con adattamento del flusso

Il framework di annotazione Microsoft semplifica l'aggiunta di funzionalità di annotazione a quasi tutte le applicazioni.

Abbinamento delle annotazioni agli oggetti con annotazioni

È possibile abbinare le annotazioni agli oggetti con annotazioni corrispondenti. Si consideri, ad esempio, una semplice applicazione per la lettura di documenti con un riquadro dei commenti. Il riquadro dei commenti potrebbe essere una casella di riepilogo in cui viene visualizzato il testo di un elenco di annotazioni ancorate a un documento. Se l'utente seleziona un elemento della casella di riepilogo, nell'applicazione viene visualizzato il paragrafo del documento a cui è ancorato l'oggetto di annotazione corrispondente.

L'esempio seguente dimostra come implementare il gestore eventi di una casella di riepilogo di questo tipo che funge da riquadro dei commenti.

```

void annotationsListBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
{

    Annotation comment = (sender as ListBox).SelectedItem as Annotation;
    if (comment != null)
    {
        // IAnchorInfo info;
        // service is an AnnotationService object
        // comment is an Annotation object
        info = AnnotationHelper.GetAnchorInfo(this.service, comment);
        TextAnchor resolvedAnchor = info.ResolvedAnchor as TextAnchor;
        TextPointer textPointer = (TextPointer)resolvedAnchor.BoundingStart;
        textPointer.Paragraph.BringIntoView();
    }
}

```

```

Private Sub annotationsListBox_SelectionChanged(ByVal sender As Object, ByVal e As SelectionChangedEventArgs)

    Dim comment As Annotation = TryCast((TryCast(sender, ListBox)).SelectedItem, Annotation)
    If comment IsNot Nothing Then
        ' service is an AnnotationService object
        ' comment is an Annotation object
        info = AnnotationHelper.GetAnchorInfo(Me.service, comment)
        Dim resolvedAnchor As TextAnchor = TryCast(info.ResolvedAnchor, TextAnchor)
        Dim textPointer As TextPointer = CType(resolvedAnchor.BoundingStart, TextPointer)
        textPointer.Paragraph.BringIntoView()
    End If
End Sub

```

Un altro scenario di esempio riguarda le applicazioni che consentono lo scambio di annotazioni e note permanenti tra i lettori di documenti tramite posta elettronica. Questa funzionalità consente a tali applicazioni di far spostare il lettore alla pagina contenente l'annotazione che viene scambiata.

Vedere anche

- [DocumentViewerBase](#)
- [DocumentViewer](#)
- [FlowDocumentPageViewer](#)
- [FlowDocumentScrollView](#)
- [FlowDocumentReader](#)
- [IAnchorInfo](#)
- [Schema annotazioni](#)
- [Panoramica sull'oggetto ContextMenu](#)
- [Panoramica sull'esecuzione di comandi](#)
- [Cenni preliminari sui documenti dinamici](#)
- [How to: Add a Command to a MenuItem \(Procedura: Aggiungere un comando a un MenuItem\)](#)

Schema annotazioni

23/10/2019 • 10 minutes to read • [Edit Online](#)

Questo argomento descrive la definizione di XML Schema (XSD, XML Schema Definition) usata da Microsoft Annotations Framework per salvare e recuperare i dati di annotazione dell'utente.

Il framework delle annotazioni serializza i dati di annotazione da una rappresentazione interna a un formato XML. Il formato XML utilizzato per questa conversione è descritto dallo schema XSD del framework delle annotazioni. Lo schema definisce il formato XML indipendente dall'implementazione che può essere usato per scambiare dati di annotazione tra le applicazioni.

Il framework delle annotazioni XML Schema definizione è costituito da due sottoschemi

- Schema principale XML delle annotazioni (schema principale).
- Schema di base XML delle annotazioni (schema di base).

Lo schema principale definisce la struttura XML primaria di un [Annotation](#) oggetto. La maggior parte degli elementi XML definiti nello schema principale corrisponde ai tipi nello [System.Windows.Annotations](#) spazio dei nomi. Lo schema principale espone tre punti di estensione in cui le applicazioni possono aggiungere i propri dati XML. Questi punti di estensione includono [Authors](#), [ContentLocatorPart](#) "Content". Gli elementi di contenuto vengono specificati sotto forma di [XmlElement](#) elenco.

Lo schema di base descritto in questo argomento definisce le estensioni per [Authors](#) i [ContentLocatorPart](#) tipi di contenuto, e inclusi nella versione iniziale Windows Presentation Foundation (WPF).

Schema principale XML delle annotazioni

Lo schema principale XML delle annotazioni definisce la struttura XML utilizzata per archiviare [Annotation](#) gli oggetti.

```
<xsd:schema elementFormDefault="qualified" attributeFormDefault="unqualified"
    blockDefault="#all"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://schemas.microsoft.com/windows/annotations/2003/11/core"
    xmlns:anc="http://schemas.microsoft.com/windows/annotations/2003/11/core">

    <!-- The Annotations element groups a number of annotations. -->
    <xsd:element name="Annotations" type="anc:AnnotationsType" />

    <xsd:complexType name="AnnotationsType">
        <xsd:sequence>
            <xsd:element name="Annotation" minOccurs="0" maxOccurs="unbounded"
                type="anc:AnnotationType" />
        </xsd:sequence>
    </xsd:complexType>

    <!-- AnnotationType defines the structure of the Annotation element. -->
    <xsd:complexType name="AnnotationType">
        <xsd:sequence>

            <!-- List of 0 or more authors. -->
            <xsd:element name="Authors" minOccurs="0" maxOccurs="1"
                type="anc:AuthorListType" />

            <!-- List of 0 or more anchors. -->
            <xsd:element name="Anchors" minOccurs="0" maxOccurs="1"
                type="anc:ResourceListType" />

        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

```

    type="anc:ResourceListType" />

    <!-- List of 0 or more cargos. -->
    <xsd:element name="Cargos" minOccurs="0" maxOccurs="1"
        type="anc:ResourceListType" />

</xsd:sequence>

<!-- Unique annotation ID. -->
<xsd:attribute name="Id" type="xsd:string" use="required" />

<!-- Annotation "Type" is used to map the annotation to an annotation
     component that takes care of the visual representation of the
     annotation. WPF V1 recognizes three annotation types:
http://schemas.microsoft.com/windows/annotations/2003/11/base:Highlight
http://schemas.microsoft.com/windows/annotations/2003/11/base:TextStickyNote
http://schemas.microsoft.com/windows/annotations/2003/11/base:InkStickyNote
-->
<xsd:attribute name="Type" type="xsd:QName" use="required" />

<!-- Time when the annotation was last modified. -->
<xsd:attribute name="LastModificationTime" use="optional"
    type="xsd:dateTime" />

<!-- Time when the annotation was created. -->
<xsd:attribute name="CreationTime" use="optional"
    type="xsd:dateTime" />
</xsd:complexType>

<!-- "Authors" consists of 0 or more elements that represent an author. -->
<xsd:complexType name="AuthorListType">
    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="anc:Author" />
    </xsd:sequence>
</xsd:complexType>

<!-- The core schema allows any author type. Supported author types
     in version 1 (V1) are described in the base schema. -->
<xsd:element name="Author" abstract="true" block="extension restriction"/>

<!-- Both annotation anchor and annotation cargo are represented by the
     ResourceListType which contains 0 or more "Resource" elements. -->
<xsd:complexType name="ResourceListType">
    <xsd:sequence>
        <xsd:element name="Resource" minOccurs="0" maxOccurs="unbounded"
            type="anc:ResourceType" />
    </xsd:sequence>
</xsd:complexType>

<!-- Resource groups identification, location
     and/or content of some information. -->
<xsd:complexType name="ResourceType">
    <xsd:choice minOccurs="0" maxOccurs="unbounded" >
        <xsd:choice>
            <xsd:element name="ContentLocator" type="anc:ContentLocatorType" />
            <xsd:element name="ContentLocatorGroup" type="anc:ContentLocatorGroupType" />
        </xsd:choice>
        <xsd:element ref="anc:Content"/>
    </xsd:choice>

    <!-- Unique resource identifier. -->
    <xsd:attribute name="Id" type="xsd:string" use="required" />

    <!-- Optional resource name. -->
    <xsd:attribute name="Name" type="xsd:string" use="optional" />
</xsd:complexType>

<!-- ContentLocatorGroup contains a set of ContentLocators -->
<xsd:complexType name="ContentLocatorGroupType">
    <xsd:sequence>
        <xsd:element name="ContentLocators" type="anc:ResourceListType" />
    </xsd:sequence>
</xsd:complexType>

```

```

<xsa:sequence>
  <xsd:element name="ContentLocator" minOccurs="1" maxOccurs="unbounded"
    type="anc:ContentLocatorType" />
</xsa:sequence>
</xsd:complexType>

<!-- A ContentLocator describes the location or the identification
     of particular data within some context. The ContentLocator consists
     of one or more ContentLocatorParts. Each ContentLocatorPart needs to
     be successively applied to the context to arrive at the data. What
     "applying", "context", and "data" mean is application dependent.
-->
<xsd:complexType name="ContentLocatorType">
  <xsd:sequence minOccurs="1" maxOccurs="unbounded">
    <xsd:element ref="anc:ContentLocatorPart" />
  </xsd:sequence>
</xsd:complexType>

<!-- A ContentLocatorPart is a set of "Item" elements. Each "Item" element
     has "Name" and "Value" attributes that define a name/value pair.
     ContentLocatorPart is an abstract type that must be restricted for each
     concrete ContentLocatorPart definition. This restriction should define
     allowed names and values for the concrete ContentLocatorPart type. That
     way the application can define its own way of locating information. The
     ContentLocatorPartTypes that are allowed in version 1 (V1) of WPF are
     defined in the Annotations Base Schema.
-->
<xsd:element name="ContentLocatorPart" type="anc:ContentLocatorPartType"
  abstract="true" />

<xsd:complexType name="ContentLocatorPartType" abstract="true"
  block="restriction">
  <xsd:sequence minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="Item" type="anc:ItemType" />
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ItemType" abstract="true" >
  <xsd:attribute name="Name" type='xsd:string' use="required" />
  <xsd:attribute name="Value" type='xsd:string' use="optional" />
</xsd:complexType>

<!-- Content describes the underlying content of a resource. This is an
     abstract type that should be redefined for each concrete content type
     through restriction. Allowed content types in WPF version 1 are
     defined in the Annotations Base Schema.
-->
<xsd:element name="Content" abstract="true" block="extension restriction"/>

</xsd:schema>

```

Schema di base XML delle annotazioni

Lo schema di base definisce la struttura XML per i tre elementi astratti definiti nello schema principale [Authors](#), [ContentLocatorPart](#) ovvero, [Content](#).

```

<xsd:schema elementFormDefault="qualified" attributeFormDefault="unqualified"
  blockDefault="#all"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://schemas.microsoft.com/windows/annotations/2003/11/base"
  xmlns:anb="http://schemas.microsoft.com/windows/annotations/2003/11/base"
  xmlns:anc="http://schemas.microsoft.com/windows/annotations/2003/11/core">

  <xsd:import schemaLocation="AnnotationCoreV1.xsd"
    namespace="http://schemas.microsoft.com/windows/annotations/2003/11/core"/>

```

```

<!-- ***** Author ---->
<!-- Simple DisplayName Author -->
<xsd:complexType name="StringAuthorType">
  <xsd:simpleContent>
    <xsd:extension base='xsd:string' />
  </xsd:simpleContent>
</xsd:complexType>
<xsd:element name="StringAuthor" type="anb:StringAuthorType"
  substitutionGroup="anc:Author"/>

<!-- ***** LocatorParts ***** -->

<!-- Helper types -->

<!-- CountItemNameType - helper type to define count item -->
<xsd:simpleType name="CountItemNameType">
  <xsd:restriction base='xsd:string'>
    <xsd:pattern value="Count" />
  </xsd:restriction>
</xsd:simpleType>

<!-- NumberType - helper type to define segment count item -->
<xsd:simpleType name="NumberType">
  <xsd:restriction base='xsd:string'>
    <xsd:pattern value="\d*" />
  </xsd:restriction>
</xsd:simpleType>

<!-- SegmentNameType: helper type to define possible segment name types -->
<xsd:simpleType name="SegmentItemNameType">
  <xsd:restriction base='xsd:string'>
    <xsd:pattern value="Segment\d*" />
  </xsd:restriction>
</xsd:simpleType>

<!-- Flow Locator Part -->

<!-- FlowSegmentValueItemType: helper type to define flow segment values -->
<xsd:simpleType name="FlowSegmentItemValueType">
  <xsd:restriction base='xsd:string'>
    <xsd:pattern value=" \d*,\d*" />
  </xsd:restriction>
</xsd:simpleType>

<!-- FlowItemType -->
<xsd:complexType name="FlowItemType" abstract = "true">
  <xsd:complexContent>
    <xsd:restriction base="anc:ItemType">
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>

<!-- FlowSegmentItemType -->
<xsd:complexType name="FlowSegmentItemType">
  <xsd:complexContent>
    <xsd:restriction base="anb:FlowItemType">
      <xsd:attribute name="Name" use="required"
        type="anb:SegmentItemNameType"/>
      <xsd:attribute name="Value" use="required"
        type="anb:FlowSegmentItemValueType"/>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>

<!-- FlowCountItemType -->
<xsd:complexType name="FlowCountItemType">
  <xsd:complexContent>
    <xsd:restriction base="anb:FlowItemType">
      <xsd:attribute name="Name" type="anb:CountItemNameType" use="required"/>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>

```

```

        <xsd:attribute name="Value" type="anb:NumberType" use="required"/>
    </xsd:restriction>
</xsd:complexContent>
</xsd:complexType>

<!-- CharacterRangeType is an extension of ContentLocatorPartType that locates
*   part of the content within a FlowDocument. CharacterRangeType contains one
*   "Item" element with name "Count" and value the number(N) of "SegmentXX"
*   elements that this ContentLocatorPart has. It also contains N "Item"
*   elements with name "SegmentXX" where XX is a number from 0 to N-1. The
*   value of each "SegmentXX" element is a string in the form "offset, length"
*   which locates one sequence of symbols in the FlowDocument. Example:

*       <anb:CharacterRange>
*           <anc:Item Name="Count" Value="2" />
*           <anc:Item Name="Segment0" Value="5,10" />
*           <anc:Item Name="Segment1" Value="25,2" />
*       </anb:CharacterRange>
-->
<xsd:complexType name="CharacterRangeType">
<xsd:complexContent>
<xsd:extension base="anc:ContentLocatorPartType">
    <xsd:sequence minOccurs="1" maxOccurs="unbounded">
        <xsd:element name="Item" type="anb:FlowItemType" />
    </xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<!-- CharacterRange element substitutes ContentLocatorPart element -->
<xsd:element name="CharacterRange" type="anb:CharacterRangeType"
    substitutionGroup="anc:ContentLocatorPart"/>

<!-- Fixed LocatorPart -->

<!-- Helper type - FixedItemType -->
<xsd:complexType name="FixedItemType" abstract = "true">
<xsd:complexContent>
<xsd:restriction base="anc:ItemType">
</xsd:restriction>
</xsd:complexContent>
</xsd:complexType>

<!-- Helper type - FixedCountItemType: ContentLocatorPart items count -->
<xsd:complexType name="FixedCountItemType">
<xsd:complexContent>
<xsd:restriction base="anb:FixedItemType">
    <xsd:attribute name="Name" type="anb:CountItemNameType" use="required"/>
    <xsd:attribute name="Value" type="anb:NumberType" use="required"/>
</xsd:restriction>
</xsd:complexContent>
</xsd:complexType>

<!-- Helper type -FixedSegmentValue: Defines possible fixed segment values -->
<xsd:simpleType name="FixedSegmentItemValueType">
    <xsd:restriction base='xsd:string'>
        <xsd:pattern value="\d*,\d*,\d*,\d*" />
    </xsd:restriction>
</xsd:simpleType>

<!-- Helper type - FixedSegmentItemType -->
<xsd:complexType name="FixedSegmentItemType">
<xsd:complexContent>
<xsd:restriction base="anb:FixedItemType">
    <xsd:attribute name="Name" use="required"
        type="anb:SegmentItemNameType"/>
    <xsd:attribute name="Value" use="required"
        type="anb:FixedSegmentItemValueType" />
</xsd:restriction>

```

```

</xsd:complexContent>
</xsd:complexType>

<!-- FixedTextRangeType is an extension of ContentLocatorPartType that locates
*   content within a FixedDocument. It contains one "Item" element with name
*   "Count" and value the number (N) of "Item" elements with name "SegmentXX"
*   that this ContentLocatorPart has. FixedTextRange locator part also
*   contains N "Item" elements with one attribute Name="SegmentXX" where XX is
*   a number from 0 to N-1 and one attribute "Value" in the form "X1, Y1, X2,
*   Y2". Here X1,Y1 are the coordinates of the start symbol in this segment,
*   X2,Y2 are the coordinates of the end symbol in this segment. Example:
*-->

*      <anb:FixedTextRange>
*          <anc:Item Name="Count" Value="2" />
*          <anc:Item Name="Segment0" Value="10,5,20,5" />
*          <anc:Item Name="Segment1" Value="25,15, 25,20" />
*      </anb:FixedTextRange>
-->
<xsd:complexType name="FixedTextRangeType">
    <xsd:complexContent>
        <xsd:extension base="anc:ContentLocatorPartType">
            <xsd:sequence minOccurs="1" maxOccurs="unbounded">
                <xsd:element name="Item" type="anb:FixedItemType" />
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<!-- FixedTextRange element substitutes ContentLocatorPart element -->
<xsd:element name="FixedTextRange" type="anb:FixedTextRangeType"
    substitutionGroup="anc:ContentLocatorPart"/>

<!-- DataId -->

<!-- ValueItemNameType: helper type to define value item -->
<xsd:simpleType name="ValueItemNameType">
    <xsd:restriction base='xsd:string'>
        <xsd:pattern value="Value" />
    </xsd:restriction>
</xsd:simpleType>

<!-- StringValueItemType -->
<xsd:complexType name="StringValueItemType">
    <xsd:complexContent>
        <xsd:restriction base="anc:ItemType">
            <xsd:attribute name="Name" type="anb:ValueItemNameType" use="required"/>
            <xsd:attribute name="Value" type="xsd:string" use="required"/>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="StringValueLocatorPartType">
    <xsd:complexContent>
        <xsd:extension base="anc:ContentLocatorPartType">
            <xsd:sequence minOccurs="1" maxOccurs="1">
                <xsd:element name="Item" type="anb:ValueItemType" />
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<!-- DataId element substitutes ContentLocatorPart and is used to locate a
*   subtree in the logical tree. Including DataId locator part in a
*   ContentLocator helps to narrow down the search for a particular content.
*   Example of DataId ContentLocatorPart:
*-->

*      <anb:DataId>
*          <anc:Item Name="Value" Value="FlowDocument" />
*      </anb:DataId>

```

```

-->

<xsd:element name="DataId" type="anb: StringValueLocatorPartType "
    substitutionGroup="anc:ContentLocatorPart"/>

<!-- PageNumber -->

<!-- NumberValueItemType -->
<xsd:complexType name="NumberValueItemType">
    <xsd:complexContent>
        <xsd:restriction base="anc:ItemType">
            <xsd:attribute name="Name" type="anb:ValueItemNameType" use="required"/>
            <xsd:attribute name="Value" type="anb:NumberType" use="required"/>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="NumberValueLocatorPartType">
    <xsd:complexContent>
        <xsd:extension base="anc:ContentLocatorPartType">
            <xsd:sequence minOccurs="1" maxOccurs="1">
                <xsd:element name="Item" type="anb:ValueItemType" />
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<!-- PageNumber element substitutes ContentLocatorPart and is used to locate a
*   page in a FixedDocument. PageNumber ContentLocatorPart is used in
*   conjunction with the FixedTextRange ContentLocatorPart and it shows on with
*   page are the coordinates defined in the FixedTextRange.
*   Example of a PageNumber ContentLocatorPart:
*
*       <anb:PageNumber>
*           <anc:Item Name="Value" Value="1" />
*       </anb:PageNumber>
-->
<xsd:element name="PageNumber" type="anb:NumberValueLocatorPartType"
    substitutionGroup="anc:ContentLocatorPart"/>

<!-- ***** Content ***** -->
<!-- Highlight colors - defines highlight color for annotations of type
*   Highlight or normal and active anchor colors for annotations of type
*   TextStickyNote and InkStickyNote.
-->
<xsd:complexType name="ColorsContentType">
    <xsd:attribute name="Background" type='xsd:string' use="required" />
    <xsd:attribute name="ActiveBackground" type='xsd:string' use="optional" />
</xsd:complexType>

<xsd:element name="Colors" type="anb:ColorsContentType"
    substitutionGroup="anc:Content"/>

<!-- RTB Text -contains XAML representing StickyNote Reach Text Box text.
*   Used in annotations of type TextStickyNote. -->
<xsd:complexType name="TextContentType">
    <!-- See XAML schema for RTB content -->
</xsd:complexType>

<xsd:element name="Text" type="anb:TextContentType"
    substitutionGroup="anc:Content"/>

<!-- Ink - contains XAML representing Sticky Note ink.
*   Used in annotations of type InkStickyNote. -->
<xsd:complexType name="InkContentType">
    <!-- See XAML schema for Ink content -->
</xsd:complexType>

<xsd:element name="Ink" type="anb:InkContentType">

```

```

<xsd:element name="Ink" type="anb:InkContentType"
    substitutionGroup="anc:Content"/>

<!-- SN Metadata - defines StickyNote attributes as position width, height,
*   etc. Used in annotations of type TextStickyNote and InkStickyNote. -->
<xsd:complexType name="MetadataContentType">
    <xsd:attribute name="Left" type='xsd:decimal' use="optional" />
    <xsd:attribute name="Top" type='xsd:decimal' use="optional" />
    <xsd:attribute name="Width" type='xsd:decimal' use="optional" />
    <xsd:attribute name="Height" type='xsd:decimal' use="optional" />
    <xsd:attribute name="XOffset" type='xsd:decimal' use="optional" />
    <xsd:attribute name="YOffset" type='xsd:decimal' use="optional" />
    <xsd:attribute name="ZOrder" type='xsd:decimal' use="optional" />
</xsd:complexType>

<xsd:element name="Metadata" type="anb:MetadataContentType"
    substitutionGroup="anc:Content"/>

</xsd:schema>

```

XML di esempio creato da Annotations XmlStreamStore

Il codice XML seguente mostra l'output di un'annotazione **XmlStreamStore** e l'organizzazione di un file di esempio che contiene tre annotazioni, un'evidenziazione, una nota appiccicosa del testo e un input penna.

```

<?xml version="1.0" encoding="utf-8"?>
<anc:Annotations
    xmlns:anc="http://schemas.microsoft.com/windows/annotations/2003/11/core"
    xmlns:anb="http://schemas.microsoft.com/windows/annotations/2003/11/base">

    <anc:Annotation Id="d308ea9b-36eb-4cc4-94d0-97634f10f7a2"
        CreationTime="2006-09-13T18:28:51.4465702-07:00"
        LastModificationTime="2006-09-13T18:28:51.4465702-07:00"
        Type="anb:Highlight">
        <anc:Anchors>
            <anc:Resource Id="4f53661b-7328-4673-8e3f-c53f08b9cd94">
                <anc:ContentLocator>
                    <anb:DataId>
                        <anc:Item Name="Value" Value="FlowDocument" />
                    </anb:DataId>
                    <anb:CharacterRange>
                        <anc:Item Name="Segment0" Value="600,609" />
                        <anc:Item Name="Count" Value="1" />
                    </anb:CharacterRange>
                </anc:ContentLocator>
            </anc:Resource>
        </anc:Anchors>
    </anc:Annotation>

    <anc:Annotation Id="d7a8d271-387e-4144-9f8b-bc3c97816e5f"
        CreationTime="2006-09-13T18:28:56.7903202-07:00"
        LastModificationTime="2006-09-13T18:28:56.8996952-07:00"
        Type="anb:TextStickyNote">
        <anc:Authors>
            <anb:StringAuthor>Denise Smith</anb:StringAuthor>
        </anc:Authors>

        <anc:Anchors>
            <anc:Resource Id="dab2560e-6ebd-4ad0-80f9-483356a3be0b">
                <anc:ContentLocator>
                    <anb:DataId>
                        <anc:Item Name="Value" Value="FlowDocument" />
                    </anb:DataId>
                    <anb:CharacterRange>
                        <anc:Item Name="Segment0" Value="787,801" />
                        <anc:Item Name="Count" Value="1" />
                    </anb:CharacterRange>
                </anc:ContentLocator>
            </anc:Resource>
        </anc:Anchors>
    </anc:Annotation>

```

```

        </anb:CharacterRange>
        </anc:ContentLocator>
    </anc:Resource>
</anc:Anchors>

<anc:Cargos>
    <anc:Resource Id="ea4dbabd-b400-4cf9-8908-5716b410f9e4" Name="Meta Data">
        <anb:MetaData anb:ZOrder="0" />
    </anc:Resource>
</anc:Cargos>
</anc:Annotation>

<anc:Annotation Id="66803c69-b0d7-4cc3-bdff-cacc1955e806">
    CreationTime="2006-09-13T18:29:03.6653202-07:00"
    LastModificationTime="2006-09-13T18:29:03.7121952-07:00"
    Type="anb:InkStickyNote">
    <anc:Authors>
        <anb:StringAuthor>Mike Nash</anb:StringAuthor>
    </anc:Authors>

    <anc:Anchors>
        <anc:Resource Id="52251c53-8eeb-4fd7-b8f3-94e78dfc25fa">
            <anc:ContentLocator>
                <anb:DataId>
                    <anc:Item Name="Value" Value="FlowDocument" />
                </anb:DataId>
                <anb:CharacterRange>
                    <anc:Item Name="Segment0" Value="880,884" />
                    <anc:Item Name="Count" Value="1" />
                </anb:CharacterRange>
            </anc:ContentLocator>
        </anc:Resource>
    </anc:Anchors>

    <anc:Cargos>
        <anc:Resource Id="11e50b97-8d91-4ff9-82c3-16607b2b552b" Name="Meta Data">
            <anb:MetaData anb:ZOrder="1" />
        </anc:Resource>
    </anc:Cargos>
</anc:Annotation>

</anc:Annotations>

```

Vedere anche

- [System.Windows.Annotations](#)
- [System.Windows.Annotations.Storage](#)
- [Annotation](#)
- [AnnotationStore](#)
- [XmlStreamStore](#)
- [Cenni preliminari sulle annotazioni](#)

Contenuto del flusso

23/10/2019 • 2 minutes to read • [Edit Online](#)

Elementi di contenuto dinamico forniscono i blocchi predefiniti per la creazione del contenuto del flusso appropriato per l'hosting in un [FlowDocument](#).

In questa sezione

[Cenni preliminari sui documenti dinamici](#)

[Panoramica sul modello di contenuto TextElement](#)

[Cenni preliminari sull'elemento Table](#)

[Procedure relative alle proprietà](#)

Riferimenti

[FlowDocument](#)

[Block](#)

[List](#)

[Paragraph](#)

[Section](#)

[Table](#)

[Figure](#)

[Floater](#)

[Hyperlink](#)

[Inline](#)

[Run](#)

[Span](#)

[ListItem](#)

Sezioni correlate

[Documenti in WPF](#)

Cenni preliminari sui documenti dinamici

23/10/2019 • 41 minutes to read • [Edit Online](#)

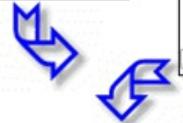
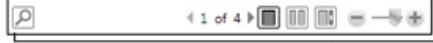
I documenti dinamici sono progettati per ottimizzare la visualizzazione e la leggibilità. Anziché essere impostati su un layout predefinito, questi documenti consentono di adattare e ridisporre il contenuto in modo dinamico in base alle variabili in fase di esecuzione, ad esempio, le dimensioni della finestra, la risoluzione del dispositivo e le preferenze facoltative dell'utente. Questi documenti offrono anche funzionalità avanzate del documento, quali paginazione e colonne. Questo argomento offre una panoramica dei documenti dinamici e di come crearli.

Che cos'è un documento dinamico

I documenti dinamici sono progettati per "adattare il flusso del contenuto" in base alle dimensioni della finestra, alla risoluzione del dispositivo e ad altre variabili di ambiente. Inoltre, i documenti dinamici dispongono di numerose funzionalità incorporate tra cui ricerca, modalità di visualizzazione che ottimizzano la leggibilità e possibilità di modificare le dimensioni e l'aspetto dei tipi di carattere. I documenti dinamici sono particolarmente adatti quando la facilità di lettura è il principale requisito d'uso per il documento. I documenti statici, al contrario, sono progettati per avere una presentazione statica e risultano utili quando la fedeltà del contenuto di origine è fondamentale. Per ulteriori informazioni sui diversi tipi di documenti, vedere [documenti in WPF](#).

La figura seguente mostra un documento dinamico di esempio visualizzato in molte finestre di dimensioni diverse. Quando l'area di visualizzazione cambia, il contenuto viene ridisposto per un uso ottimale dello spazio disponibile.

Lore ipsum dolor sit amet, consectetur adipiscing elit. Nulla ligula tortor, dapibus et, facilisis a, aliquet et, diam. Cras convallis. Fusce at urna. Quisque interdum, turpis sed dictum fringilla, magna arcu gravida libero, elementum tincidunt dolor mauris sed erat. Curabitur egestas aliquet nibh. Etiam quis sem in lorem auctor consequat. Suspendisse vitae lorem. Proin eleifend elementum ligula. Donec mattis consectetur erat. Donec fermentum consectetur neque. Suspendisse tempus faucibus magna. Pellentesque sodales velit eget nibh. Phasellus velit magna, malesuada vitae, rhoncus eget, dignissim ut, metus. Praesent pulvinar suscipit diam.



Lore ipsum dolor sit amet, consectetur adipiscing elit. Nulla ligula tortor, dapibus et, facilisis a, aliquet et, diam. Cras convallis. Fusce at urna. Quisque interdum, turpis sed dictum fringilla, magna arcu gravida libero, elementum tincidunt dolor mauris sed erat. Curabitur egestas aliquet nibh. Etiam quis sem in lorem auctor consequat. Suspendisse vitae lorem. Proin eleifend elementum ligula. Donec mattis consectetur erat. Donec fermentum neque. Suspendisse tempus fauces magna. Pellentesque sodales velit eget nibh. Phasellus velit magna, malesuada vitae, rhoncus eget, dignissim ut, metus. Praesent pulvinar suscipit diam.



Morbi ut tellus at tellus semper consectetuer. Nullam fringilla nonummy justo. Proin rutrum, purus id adipiscing malesuada, enim velit accumsan nisi, pellentesque rhoncus nibh nisi ut magna. Nunc massa nulla, volutpat sit amet, pulvinar ac, scelerisque ac, magna. Nulla facilisi. Integer pharetra felis vitae risus. Nunc aliquet lacus pretium urna varius hendrerit. Duis odio nisi, venenatis at, sollicitudin eu, viverra sed, nibh. Nullam consequat. Duis felis felis, rutrum viverra, auctor eget, iaculis ut, mi. Integer sagittis pharetra ipsum. Donec lacinia scelerisque nisi. Nullam aliquet. In et sapien. Fusce blandit odio et mi.

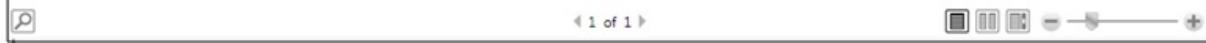


Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla ligula tortor, dapibus et, facilisis a, aliquet et, diam. Cras convallis. Fusce at urna. Quisque interdum, turpis sed dictum fringilla, magna arcu gravida libero, elementum tincidunt dolor mauris sed erat. Curabitur egestas aliquet nibh. Etiam quis sem in lorem auctor consequat. Suspendisse vitae lorem. Proin eleifend elementum ligula. Donec mattis consectetur erat. Donec fermentum consectetur neque. Suspendisse tempus faucibus magna. Pellentesque sodales velit eget nibh. Phasellus velit magna, malesuada vitae, rhoncus eget, dignissim ut, metus. Praesent pulvinar suscipit diam.



Morbi ut tellus at tellus semper consectetuer. Nullam fringilla nonummy justo. Proin rutrum, purus id adipiscing malesuada, enim velit accumsan nisi, pellentesque rhoncus nibh nisi ut magna. Nunc massa nulla, volutpat sit amet, pulvinar ac, scelerisque ac, magna. Nulla facilisi. Integer pharetra felis vitae risus. Nunc aliquet lacus pretium urna varius hendrerit. Duis odio nisl, venenatis at, sollicitudin eu, viverra sed, nibh. Nullam consequat. Duis felis felis, rutrum viverra, auctor eget, iaculis ut, mi. Integer sagittis pharetra ipsum. Donec lacinia scelerisque nisl. Nullam aliquet. In et sapien. Fusce blandit odio et mi.

Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Suspendisse laoreet condimentum purus. Etiam vel enim. Suspendisse at dolor. Pellentesque gravida. Aenean convallis. Vivamus diam. Aenean lorem libero, suscipit vel, tempor eu, fermentum aliquam, metus. Quisque volutpat urna at augue. Nam placerat. In viverra congue tellus. Proin auctor. Fusce nisl lorem, dapibus vitae, porta sed, malesuada a, lacus. Phasellus mollis elementum enim. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam vitae urna vel velit volutpat tempor. Quisque ac dolor. Suspendisse potenti. Nunc nec tortor. Curabitur pharetra pellentesque diam.



Come illustrato nell'immagine precedente, il contenuto dinamico può includere molti componenti, tra cui paragrafi, elenchi, immagini e altro. Questi componenti corrispondono agli elementi nel markup e agli oggetti nel codice procedurale. Queste classi verranno esaminate in dettaglio più avanti nella sezione [relativa alle classi correlate al flusso](#) di questa panoramica. Per il momento, di seguito è riportato un semplice esempio di codice che crea un documento dinamico costituito da un paragrafo con un testo in grassetto e un elenco.

```
<!-- This simple flow document includes a paragraph with some
bold text in it and a list. -->
<FlowDocumentReader xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
<FlowDocument>
<Paragraph>
<Bold>Some bold text in the paragraph.</Bold>
Some text that is not bold.
</Paragraph>

<List>
<ListItem>
<Paragraph>ListItem 1</Paragraph>
</ListItem>
<ListItem>
<Paragraph>ListItem 2</Paragraph>
</ListItem>
<ListItem>
<Paragraph>ListItem 3</Paragraph>
</ListItem>
</List>

</FlowDocument>
</FlowDocumentReader>
```

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;

namespace SDKSample
{
    public partial class SimpleFlowExample : Page
    {
        public SimpleFlowExample()
        {

            Paragraph myParagraph = new Paragraph();

            // Add some Bold text to the paragraph
            myParagraph.Inlines.Add(new Bold(new Run("Some bold text in the paragraph.")));

            // Add some plain text to the paragraph
            myParagraph.Inlines.Add(new Run(" Some text that is not bold."));

            // Create a List and populate with three list items.
            List myList = new List();

            // First create paragraphs to go into the list item.
            Paragraph paragraphListItem1 = new Paragraph(new Run("ListItem 1"));
            Paragraph paragraphListItem2 = new Paragraph(new Run("ListItem 2"));
            Paragraph paragraphListItem3 = new Paragraph(new Run("ListItem 3"));

            // Add ListItems with paragraphs in them.
            myList.ListItems.Add(new ListItem(paragraphListItem1));
            myList.ListItems.Add(new ListItem(paragraphListItem2));
            myList.ListItems.Add(new ListItem(paragraphListItem3));

            // Create a FlowDocument with the paragraph and list.
            FlowDocument myFlowDocument = new FlowDocument();
            myFlowDocument.Blocks.Add(myParagraph);
            myFlowDocument.Blocks.Add(myList);

            // Add the FlowDocument to a FlowDocumentReader Control
            FlowDocumentReader myFlowDocumentReader = new FlowDocumentReader();
            myFlowDocumentReader.Document = myFlowDocument;

            this.Content = myFlowDocumentReader;
        }
    }
}
```

```

Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Documents

Namespace SDKSample
    Partial Public Class SimpleFlowExample
        Inherits Page
        Public Sub New()

            Dim myParagraph As New Paragraph()

            ' Add some Bold text to the paragraph
            myParagraph.Inlines.Add(New Bold(New Run("Some bold text in the paragraph.")))

            ' Add some plain text to the paragraph
            myParagraph.Inlines.Add(New Run(" Some text that is not bold."))

            ' Create a List and populate with three list items.
            Dim myList As New List()

            ' First create paragraphs to go into the list item.
            Dim paragraphListItem1 As New Paragraph(New Run("ListItem 1"))
            Dim paragraphListItem2 As New Paragraph(New Run("ListItem 2"))
            Dim paragraphListItem3 As New Paragraph(New Run("ListItem 3"))

            ' Add ListItems with paragraphs in them.
            myList.ListItems.Add(New ListItem(paragraphListItem1))
            myList.ListItems.Add(New ListItem(paragraphListItem2))
            myList.ListItems.Add(New ListItem(paragraphListItem3))

            ' Create a FlowDocument with the paragraph and list.
            Dim myFlowDocument As New FlowDocument()
            myFlowDocument.Blocks.Add(myParagraph)
            myFlowDocument.Blocks.Add(myList)

            ' Add the FlowDocument to a FlowDocumentReader Control
            Dim myFlowDocumentReader As New FlowDocumentReader()
            myFlowDocumentReader.Document = myFlowDocument

            Me.Content = myFlowDocumentReader
        End Sub
    End Class
End Namespace

```

Nella figura riportata di seguito viene illustrato il frammento di codice in questione.

![Screenshot Esempio]di FlowDocument sottoposto a rendering./media/flow-ovw-first-example.png "Flow_Ovw_First_Example")

In questo esempio il **FlowDocumentReader** controllo viene usato per ospitare il contenuto del flusso. Per altre informazioni sui controlli di hosting del contenuto dinamico, vedere [tipi di documenti dinamici](#). **Paragraph** gli elementi, **List**, **Bold** e vengono usati per controllare la formattazione del contenuto, in base al relativo ordine di markup. **ListItem** Ad esempio, l' **Bold** elemento si estende solo in parte del testo del paragrafo. di conseguenza, solo quella parte del testo è in grassetto. Se si ha esperienza con il linguaggio HTML, questo meccanismo sarà familiare.

Come evidenziato nell'illustrazione precedente, sono disponibili diverse funzionalità incorporate nei documenti dinamici:

- Ricerca: Consente all'utente di eseguire una ricerca full-text di un intero documento.
- Modalità di visualizzazione: L'utente può selezionare la modalità di visualizzazione preferita, inclusa una modalità di visualizzazione a pagina singola (pagina alla volta), una modalità di visualizzazione a due

pagine (formato lettura libro) e una modalità di visualizzazione a scorrimento continuo (senza ribasso).

Per ulteriori informazioni su queste modalità di visualizzazione,

[FlowDocumentReaderViewingMode](#) vedere.

- Controlli per la navigazione tra le pagine: Se la modalità di visualizzazione del documento utilizza pagine, i controlli di spostamento della pagina includono un pulsante per passare alla pagina successiva (la freccia verso il basso) o alla pagina precedente (freccia su), nonché agli indicatori per il numero di pagina corrente e il numero totale di pagine. È anche possibile scorrere le pagine usando i tasti di direzione della tastiera.
- Zoom: I controlli zoom consentono all'utente di aumentare o diminuire il livello di zoom facendo clic rispettivamente sui pulsanti più o meno. I controlli dello zoom includono anche un dispositivo di scorrimento per regolare il livello dello zoom. Per altre informazioni, vedere [Zoom](#).

Queste funzionalità possono essere modificate in base al controllo usato per ospitare il contenuto del flusso. I vari controlli vengono descritti nella sezione seguente.

Tipi di documenti dinamici

La visualizzazione del contenuto del documento dinamico e la modalità di visualizzazione corrispondente dipendono dall'oggetto usato per ospitare il contenuto del flusso. Sono disponibili quattro controlli che supportano la visualizzazione del contenuto del [FlowDocumentReader](#) flusso [FlowDocumentPageViewer](#): [RichTextBox](#), [FlowDocumentScrollView](#) e [FlowDocumentPageViewer](#). Questi controlli vengono descritti brevemente di seguito.

NOTE

[FlowDocument](#) è necessario per ospitare direttamente il contenuto del flusso, quindi tutti questi controlli di visualizzazione [FlowDocument](#) utilizzano un per abilitare l'hosting del contenuto del flusso.

FlowDocumentReader

[FlowDocumentReader](#) include funzionalità che consentono all'utente di scegliere in modo dinamico tra le varie modalità di visualizzazione, inclusa una modalità di visualizzazione a pagina singola (pagina per volta), una modalità di visualizzazione a due pagine alla volta (formato lettura libro) e una modalità di visualizzazione a scorrimento continuo (senza alcun risultato). Per ulteriori informazioni su queste modalità di visualizzazione, [FlowDocumentReaderViewingMode](#) vedere. Se non è necessario passare dinamicamente da una modalità [FlowDocumentPageViewer](#) di visualizzazione all'altra e [FlowDocumentScrollView](#) fornire visualizzatori di contenuto del flusso più semplici che sono corretti in una particolare modalità di visualizzazione.

FlowDocumentPageViewer e FlowDocumentScrollView

[FlowDocumentPageViewer](#) Visualizza il contenuto nella modalità di visualizzazione pagina alla volta, mentre [FlowDocumentScrollView](#) Visualizza il contenuto nella modalità di scorrimento continuo.

[FlowDocumentPageViewer](#) E [FlowDocumentScrollView](#) sono corretti a una particolare modalità di visualizzazione. Confrontare con [FlowDocumentReader](#), che include funzionalità che consentono all'utente di scegliere dinamicamente tra le varie modalità di visualizzazione (fornite [FlowDocumentReaderViewingMode](#) dall'enumerazione), a scapito di un maggior utilizzo di risorse [FlowDocumentPageViewer](#) rispetto [FlowDocumentScrollView](#) a o.

Per impostazione predefinita, viene sempre visualizzata una barra di scorrimento verticale e in caso di necessità diventa visibile una barra di scorrimento orizzontale. L'interfaccia utente predefinita [FlowDocumentScrollView](#) per non include una barra degli strumenti. tuttavia [IsToolBarVisible](#) , la proprietà può essere utilizzata per abilitare una barra degli strumenti incorporata.

RichTextBox

Usare un [RichTextBox](#) quando si vuole consentire all'utente di modificare il contenuto del flusso. Se, ad

esempio, si desidera creare un editor che consentisse a un utente di modificare elementi quali tabelle, formattazione in corsivo e grassetto e così [RichTextBox](#) via, si utilizzerebbe un. Per ulteriori informazioni, vedere [Cenni preliminari su RichTextBox](#).

NOTE

Il contenuto del flusso [RichTextBox](#) all'interno di un non si comporta esattamente come il contenuto del flusso contenuto in altri controlli. Ad esempio, non sono presenti colonne in un [RichTextBox](#) oggetto e pertanto nessun comportamento di ridimensionamento automatico. Inoltre, le funzionalità di in genere compilate con contenuto dinamico come la ricerca, la modalità di visualizzazione, l'esplorazione delle pagine [RichTextBox](#) e lo zoom non sono disponibili all'interno di.

Creazione di contenuto dinamico

Il contenuto del flusso può essere complesso, costituito da vari elementi, tra cui testo, immagini, [UIElement](#) tabelle e persino classi derivate come i controlli. Per comprendere la modalità di creazione di contenuto dinamico complesso, i punti riportati di seguito sono fondamentali:

- **Classi correlate al flusso:** Ogni classe utilizzata nel contenuto dinamico ha uno scopo specifico. Inoltre, la relazione gerarchica tra le classi di flusso facilita la comprensione della modalità d'uso. Ad esempio, le classi derivate dalla [Block](#) classe vengono usate per contenere altri oggetti mentre le classi derivate da [Inline](#) contengono oggetti visualizzati.
- **Schema del contenuto:** Un documento dinamico può richiedere un numero considerevole di elementi annidati. Nello schema del contenuto vengono specificate le possibili relazioni padre/figlio tra elementi.

Ognuna di queste aree verrà esaminata in maniera dettagliata nella sezioni seguenti.

Classi correlate al flusso

Il diagramma seguente illustra gli oggetti usati più di frequente con il contenuto dinamico:

![Diagramma: Gerarchia]di classi di elementi del contenuto di flusso./media/flow-class-hierarchy.png
"Flow_Class_Hierarchy")

Ai fini del contenuto dinamico, esistono due categorie importanti:

1. **Classi derivate da Block:** Detto anche "elementi di contenuto del blocco" o solo "elementi Block". Gli elementi che ereditano da [Block](#) possono essere usati per raggruppare gli elementi in un elemento padre comune o per applicare attributi comuni a un gruppo.
2. **Classi derivate da inline:** Detti anche "elementi di contenuto inline" o solo "elementi inline". Gli elementi che ereditano da sono contenuti all'interno di un elemento Block o di [Inline](#) un altro elemento inline. Gli elementi [Inline](#) vengono spesso usati come contenitori diretti del contenuto di cui viene eseguito il rendering sullo schermo. Un [Paragraph](#) (elemento Block), ad esempio, può contenere [Run](#) un (elemento [Run](#) inline), ma contiene effettivamente il testo di cui viene eseguito il rendering sullo schermo.

Ogni classe di queste due categorie è descritta brevemente di seguito.

Classi derivate da Block

Paragraph

[Paragraph](#)viene in genere usato per raggruppare il contenuto in un paragrafo. L'uso più semplice e più comune di [Paragraph](#) è creare un paragrafo di testo.

```

<FlowDocument xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Paragraph>
        Some paragraph text.
    </Paragraph>
</FlowDocument>

```

```

using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;

namespace SDKSample
{
    public partial class ParagraphExample : Page
    {
        public ParagraphExample()
        {

            // Create paragraph with some text.
            Paragraph myParagraph = new Paragraph();
            myParagraph.Inlines.Add(new Run("Some paragraph text."));

            // Create a FlowDocument and add the paragraph to it.
            FlowDocument myFlowDocument = new FlowDocument();
            myFlowDocument.Blocks.Add(myParagraph);

            this.Content = myFlowDocument;
        }
    }
}

```

```

Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Documents

Namespace SDKSample
    Partial Public Class ParagraphExample
        Inherits Page
        Public Sub New()

            ' Create paragraph with some text.
            Dim myParagraph As New Paragraph()
            myParagraph.Inlines.Add(New Run("Some paragraph text.))

            ' Create a FlowDocument and add the paragraph to it.
            Dim myFlowDocument As New FlowDocument()
            myFlowDocument.Blocks.Add(myParagraph)

            Me.Content = myFlowDocument
        End Sub
    End Class
End Namespace

```

Tuttavia, è possibile anche contenere altri elementi derivati da inline come illustrato di seguito.

Sezione

[Section](#) viene usato solo per contenere altri [Block](#) elementi derivati da. Non applica alcuna formattazione predefinita agli elementi in essa contenuti. Tuttavia, i valori delle proprietà impostati su [Section](#) un oggetto si applicano ai relativi elementi figlio. Una sezione consente inoltre di effettuare iterazioni a livello di codice

mediante la relativa raccolta figlio. [Section](#) viene usato in modo analogo al <tag div> in HTML.

Nell'esempio seguente vengono definiti tre paragrafi sotto uno [Section](#). Il valore della [Background](#) proprietà della sezione è rosso, quindi anche il colore di sfondo dei paragrafi è rosso.

```
<FlowDocument xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <!-- By default, Section applies no formatting to elements contained
        within it. However, in this example, the section has a Background
        property value of "Red", therefore, the three paragraphs (the block)
        inside the section also have a red background. -->
    <Section Background="Red">
        <Paragraph>
            Paragraph 1
        </Paragraph>
        <Paragraph>
            Paragraph 2
        </Paragraph>
        <Paragraph>
            Paragraph 3
        </Paragraph>
    </Section>
</FlowDocument>
```

```
using System;
using System.Windows;
using System.Windows.Media;
using System.Windows.Controls;
using System.Windows.Documents;

namespace SDKSample
{
    public partial class SectionExample : Page
    {
        public SectionExample()
        {

            // Create three paragraphs
            Paragraph myParagraph1 = new Paragraph(new Run("Paragraph 1"));
            Paragraph myParagraph2 = new Paragraph(new Run("Paragraph 2"));
            Paragraph myParagraph3 = new Paragraph(new Run("Paragraph 3"));

            // Create a Section and add the three paragraphs to it.
            Section mySection = new Section();
            mySection.Background = Brushes.Red;

            mySection.Blocks.Add(myParagraph1);
            mySection.Blocks.Add(myParagraph2);
            mySection.Blocks.Add(myParagraph3);

            // Create a FlowDocument and add the section to it.
            FlowDocument myFlowDocument = new FlowDocument();
            myFlowDocument.Blocks.Add(mySection);

            this.Content = myFlowDocument;
        }
    }
}
```

```

Imports System.Windows
Imports System.Windows.Media
Imports System.Windows.Controls
Imports System.Windows.Documents

Namespace SDKSample
    Partial Public Class SectionExample
        Inherits Page
        Public Sub New()

            ' Create three paragraphs
            Dim myParagraph1 As New Paragraph(New Run("Paragraph 1"))
            Dim myParagraph2 As New Paragraph(New Run("Paragraph 2"))
            Dim myParagraph3 As New Paragraph(New Run("Paragraph 3"))

            ' Create a Section and add the three paragraphs to it.
            Dim mySection As New Section()
            mySection.Background = Brushes.Red

            mySection.Blocks.Add(myParagraph1)
            mySection.Blocks.Add(myParagraph2)
            mySection.Blocks.Add(myParagraph3)

            ' Create a FlowDocument and add the section to it.
            Dim myFlowDocument As New FlowDocument()
            myFlowDocument.Blocks.Add(mySection)

            Me.Content = myFlowDocument
        End Sub
    End Class
End Namespace

```

BlockUIContainer

[BlockUIContainer](#) consente [UIElement](#) agli elementi (ad esempio [Button](#)) di essere incorporati nel contenuto del flusso derivato dal blocco. [InlineUIContainer](#) (vedere di seguito) viene usato per [UIElement](#) incorporare elementi nel contenuto di flusso derivato da inline. [BlockUIContainer](#) e [InlineUIContainer](#) sono importanti perché non esiste un altro modo per usare un [UIElement](#) oggetto nel contenuto del flusso, a meno che non sia contenuto in uno di questi due elementi.

Nell'esempio seguente viene illustrato come utilizzare l' [BlockUIContainer](#) elemento per ospitare [UIElement](#) oggetti all'interno del contenuto del flusso.

```

<FlowDocument ColumnWidth="400">
    <Section Background="GhostWhite">
        <Paragraph>
            A UIElement element may be embedded directly in flow content
            by enclosing it in a BlockUIContainer element.
        </Paragraph>
        <BlockUIContainer>
            <Button>Click me!</Button>
        </BlockUIContainer>
        <Paragraph>
            The BlockUIContainer element may host no more than one top-level
            UIElement. However, other UIElements may be nested within the
            UIElement contained by an BlockUIContainer element. For example,
            a StackPanel can be used to host multiple UIElement elements within
            a BlockUIContainer element.
        </Paragraph>
        <BlockUIContainer>
            <StackPanel>
                <Label Foreground="Blue">Choose a value:</Label>
                <ComboBox>
                    <ComboBoxItem IsSelected="True">a</ComboBoxItem>
                    <ComboBoxItem>b</ComboBoxItem>
                    <ComboBoxItem>c</ComboBoxItem>
                </ComboBox>
                <Label Foreground ="Red">Choose a value:</Label>
                <StackPanel>
                    <RadioButton>x</RadioButton>
                    <RadioButton>y</RadioButton>
                    <RadioButton>z</RadioButton>
                </StackPanel>
                <Label>Enter a value:</Label>
                <TextBox>
                    A text editor embedded in flow content.
                </TextBox>
            </StackPanel>
        </BlockUIContainer>
    </Section>
</FlowDocument>

```

Nella figura seguente viene illustrato il rendering di questo esempio:

A UIElement element may be embedded directly in flow content by enclosing it in a BlockUIContainer element.

Click me!

The BlockUIContainer element may host no more than one top-level UIElement. However, other UIElements may be nested within the UIElement contained by an BlockUIContainer element. For example, a StackPanel can be used to host multiple UIElement elements within a BlockUIContainer element.

Choose a value:

a



Choose a value:

- x
- y
- z

Enter a value:

A text editor embedded in flow content.

List

List viene usato per creare un elenco puntato o numerico. Impostare la **MarkerStyle** proprietà su un

[TextMarkerStyle](#) valore di enumerazione per determinare lo stile dell'elenco. L'esempio seguente mostra come creare un elenco semplice.

```
<FlowDocument xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <List>
    <ListItem>
      <Paragraph>
        List Item 1
      </Paragraph>
    </ListItem>
    <ListItem>
      <Paragraph>
        List Item 2
      </Paragraph>
    </ListItem>
    <ListItem>
      <Paragraph>
        List Item 3
      </Paragraph>
    </ListItem>
  </List>
</FlowDocument>
```

```
using System;
using System.Windows;
using System.Windows.Media;
using System.Windows.Controls;
using System.Windows.Documents;

namespace SDKSample
{
    public partial class ListExample : Page
    {
        public ListExample()
        {

            // Create three paragraphs
            Paragraph myParagraph1 = new Paragraph(new Run("List Item 1"));
            Paragraph myParagraph2 = new Paragraph(new Run("List Item 2"));
            Paragraph myParagraph3 = new Paragraph(new Run("List Item 3"));

            // Create the ListItem elements for the List and add the
            // paragraphs to them.
            ListItem myListItem1 = new ListItem();
            myListItem1.Blocks.Add(myParagraph1);
            ListItem myListItem2 = new ListItem();
            myListItem2.Blocks.Add(myParagraph2);
            ListItem myListItem3 = new ListItem();
            myListItem3.Blocks.Add(myParagraph3);

            // Create a List and add the three ListItems to it.
            List myList = new List();

            myList.ListItems.Add(myListItem1);
            myList.ListItems.Add(myListItem2);
            myList.ListItems.Add(myListItem3);

            // Create a FlowDocument and add the section to it.
            FlowDocument myFlowDocument = new FlowDocument();
            myFlowDocument.Blocks.Add(myList);

            this.Content = myFlowDocument;
        }
    }
}
```

```

Imports System.Windows
Imports System.Windows.Media
Imports System.Windows.Controls
Imports System.Windows.Documents

Namespace SDKSample
    Partial Public Class ListExample
        Inherits Page
        Public Sub New()

            ' Create three paragraphs
            Dim myParagraph1 As New Paragraph(New Run("List Item 1"))
            Dim myParagraph2 As New Paragraph(New Run("List Item 2"))
            Dim myParagraph3 As New Paragraph(New Run("List Item 3"))

            ' Create the ListItem elements for the List and add the
            ' paragraphs to them.
            Dim myListItem1 As New ListItem()
            myListItem1.Blocks.Add(myParagraph1)
            Dim myListItem2 As New ListItem()
            myListItem2.Blocks.Add(myParagraph2)
            Dim myListItem3 As New ListItem()
            myListItem3.Blocks.Add(myParagraph3)

            ' Create a List and add the three ListItems to it.
            Dim myList As New List()

            myList.ListItems.Add(myListItem1)
            myList.ListItems.Add(myListItem2)
            myList.ListItems.Add(myListItem3)

            ' Create a FlowDocument and add the section to it.
            Dim myFlowDocument As New FlowDocument()
            myFlowDocument.Blocks.Add(myList)

            Me.Content = myFlowDocument
        End Sub
    End Class
End Namespace

```

NOTE

[List](#) è l'unico elemento Flow che utilizza [ListItemCollection](#) per gestire gli elementi figlio.

Tabella

[Table](#) viene utilizzato per creare una tabella. [Table](#) è simile all'elemento [Grid](#), ma dispone di più funzionalità e, di conseguenza, richiede un sovraccarico di risorse maggiore. Poiché [Grid](#) è un [UIElement](#), non può essere usato nel contenuto del flusso a meno che non sia contenuto [BlockUIContainer](#) in [InlineUIContainer](#) un oggetto o. Per altre informazioni su [Table](#), vedere [Cenni preliminari sulle tabelle](#).

Classi derivate da Inline

Run

[Run](#) viene utilizzato per contenere testo non formattato. È possibile [Run](#) che gli oggetti vengano usati in modo estensivo nel contenuto del flusso. Tuttavia, nel markup [Run](#) gli elementi non devono essere utilizzati in modo esplicito. [Run](#) è necessario utilizzare per la creazione o la modifica di documenti dinamici tramite codice. Nel markup seguente, ad esempio, il primo [Paragraph](#) specifica l' [Run](#) elemento in modo esplicito, mentre il secondo no. Entrambi i paragrafi generano output identici.

```
<Paragraph>
  <Run>Paragraph that explicitly uses the Run element.</Run>
</Paragraph>

<Paragraph>
  This Paragraph omits the Run element in markup. It renders
  the same as a Paragraph with Run used explicitly.
</Paragraph>
```

NOTE

A partire da .NET Framework 4, la [Text](#) proprietà [Run](#) dell'oggetto è una proprietà di dipendenza. È possibile associare la [Text](#) proprietà a un'origine dati, ad esempio. [TextBlock](#) La [Text](#) proprietà supporta completamente l'associazione unidirezionale. La [Text](#) proprietà supporta anche l'associazione bidirezionale, ad eccezione [RichTextBox](#). Per un esempio, vedere [Run.Text](#).

Span

[Span](#) Raggruppa altri elementi di contenuto inline. Nessun rendering inherente viene applicato al contenuto all'[Span](#) interno di un elemento. Tuttavia, gli elementi che ereditano [Hyperlink](#), [Bold](#), [Span](#), [Italic](#) e [Underline](#) applicano la formattazione al testo.

Di seguito è riportato un esempio [Span](#) di un oggetto utilizzato per contenere contenuto inline [Bold](#), incluso testo, elemento e [Button](#).

```
<FlowDocument xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

  <Paragraph>
    Text before the Span. <Span Background="Red">Text within the Span is
    red and <Bold>this text is inside the Span-derived element Bold.</Bold>
    A Span can contain more than text, it can contain any inline content. For
    example, it can contain a
    <InlineUIContainer>
      <Button>Button</Button>
    </InlineUIContainer>
    or other UIElement, a Floater, a Figure, etc.</Span>
  </Paragraph>

</FlowDocument>
```

Lo screenshot seguente illustra il rendering di questo esempio.

![Screenshot Esempio] di span con rendering(./media/flow-spanexample.gif "Flow_SpanExample")

InlineUIContainer

[InlineUIContainer](#) Abilita [UIElement](#) gli elementi, ad esempio un controllo [Button](#) come, da incorporare in [Inline](#) un elemento di contenuto. Questo elemento è l'equivalente inline a [BlockUIContainer](#) quello descritto in precedenza. Di seguito è riportato un esempio [InlineUIContainer](#) che usa per [Button](#) inserire un oggetto inline in un oggetto [Paragraph](#).

```

<FlowDocument xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <Paragraph>
        Text to precede the button...

        <!-- Set the BaselineAlignment property to "Bottom"
            so that the Button aligns properly with the text. -->
        <InlineUIContainer BaselineAlignment="Bottom">
            <Button>Button</Button>
        </InlineUIContainer>
        Text to follow the button...
    </Paragraph>

</FlowDocument>

```

```

using System;
using System.Windows;
using System.Windows.Media;
using System.Windows.Controls;
using System.Windows.Documents;

namespace SDKSample
{
    public partial class InlineUIContainerExample : Page
    {
        public InlineUIContainerExample()
        {
            Run run1 = new Run(" Text to precede the button... ");
            Run run2 = new Run(" Text to follow the button... ");

            // Create a new button to be hosted in the paragraph.
            Button myButton = new Button();
            myButton.Content = "Click me!";

            // Create a new InlineUIContainer to contain the Button.
            InlineUIContainer myInlineUIContainer = new InlineUIContainer();

            // Set the BaselineAlignment property to "Bottom" so that the
            // Button aligns properly with the text.
            myInlineUIContainer.BaselineAlignment = BaselineAlignment.Bottom;

            // Assign the button as the UI container's child.
            myInlineUIContainer.Child = myButton;

            // Create the paragraph and add content to it.
            Paragraph myParagraph = new Paragraph();
            myParagraph.Inlines.Add(run1);
            myParagraph.Inlines.Add(myInlineUIContainer);
            myParagraph.Inlines.Add(run2);

            // Create a FlowDocument and add the paragraph to it.
            FlowDocument myFlowDocument = new FlowDocument();
            myFlowDocument.Blocks.Add(myParagraph);

            this.Content = myFlowDocument;
        }
    }
}

```

```

Imports System.Windows
Imports System.Windows.Media
Imports System.Windows.Controls
Imports System.Windows.Documents

Namespace SDKSample
    Partial Public Class InlineUIContainerExample
        Inherits Page
        Public Sub New()
            Dim run1 As New Run(" Text to precede the button... ")
            Dim run2 As New Run(" Text to follow the button... ")

            ' Create a new button to be hosted in the paragraph.
            Dim myButton As New Button()
            myButton.Content = "Click me!"

            ' Create a new InlineUIContainer to contain the Button.
            Dim myInlineUIContainer As New InlineUIContainer()

            ' Set the BaselineAlignment property to "Bottom" so that the
            ' Button aligns properly with the text.
            myInlineUIContainer.BaselineAlignment = BaselineAlignment.Bottom

            ' Assign the button as the UI container's child.
            myInlineUIContainer.Child = myButton

            ' Create the paragraph and add content to it.
            Dim myParagraph As New Paragraph()
            myParagraph.Inlines.Add(run1)
            myParagraph.Inlines.Add(myInlineUIContainer)
            myParagraph.Inlines.Add(run2)

            ' Create a FlowDocument and add the paragraph to it.
            Dim myFlowDocument As New FlowDocument()
            myFlowDocument.Blocks.Add(myParagraph)

            Me.Content = myFlowDocument
        End Sub
    End Class
End Namespace

```

NOTE

[InlineUIContainer](#) non deve essere utilizzato in modo esplicito nel markup. Se viene omesso, verrà [InlineUIContainer](#) creato un oggetto quando il codice viene compilato.

Figure e Floater

[Figure](#) e [Floater](#) vengono usati per incorporare contenuto nei documenti dinamici con proprietà di posizionamento che possono essere personalizzate indipendentemente dal flusso di contenuto principale. [Figure](#) gli [Floater](#) elementi o vengono spesso usati per evidenziare o accettare porzioni di contenuto, per ospitare immagini di supporto o altro contenuto all'interno del flusso di contenuto principale o per inserire contenuti correlati in modo libero come gli annunci.

Nell'esempio seguente viene illustrato come incorporare [Figure](#) un oggetto in un paragrafo di testo.

```
<FlowDocument xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

  <Paragraph>
    <Figure
      Width="300" Height="100"
      Background="GhostWhite" HorizontalAnchor="PageLeft" >
      <Paragraph FontStyle="Italic" Background="Beige" Foreground="DarkGreen" >
        A Figure embeds content into flow content with placement properties
        that can be customized independently from the primary content flow
      </Paragraph>
    </Figure>
    Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy
    nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi
    enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis
    nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure.
  </Paragraph>

</FlowDocument>
```

```

using System;
using System.Windows;
using System.Windows.Media;
using System.Windows.Controls;
using System.Windows.Documents;

namespace SDKSample
{
    public partial class FigureExample : Page
    {
        public FigureExample()
        {

            // Create strings to use as content.
            string strFigure = "A Figure embeds content into flow content with" +
                " placement properties that can be customized" +
                " independently from the primary content flow";
            string strOther = "Lorem ipsum dolor sit amet, consectetur adipiscing" +
                " elit, sed diam nonummy nibh euismod tincidunt ut laoreet" +
                " dolore magna aliquam erat volutpat. Ut wisi enim ad" +
                " minim veniam, quis nostrud exerci tation ullamcorper" +
                " suscipit lobortis nisl ut aliquip ex ea commodo consequat." +
                " Duis autem vel eum iriure.";

            // Create a Figure and assign content and layout properties to it.
            Figure myFigure = new Figure();
            myFigure.Width = new FigureLength(300);
            myFigure.Height = new FigureLength(100);
            myFigure.Background = Brushes.GhostWhite;
            myFigure.HorizontalAnchor = FigureHorizontalAnchor.PageLeft;
            Paragraph myFigureParagraph = new Paragraph(new Run(strFigure));
            myFigureParagraph.FontStyle = FontStyles.Italic;
            myFigureParagraph.Background = Brushes.Beige;
            myFigureParagraph.Foreground = Brushes.DarkGreen;
            myFigure.Blocks.Add(myFigureParagraph);

            // Create the paragraph and add content to it.
            Paragraph myParagraph = new Paragraph();
            myParagraph.Inlines.Add(myFigure);
            myParagraph.Inlines.Add(new Run(strOther));

            // Create a FlowDocument and add the paragraph to it.
            FlowDocument myFlowDocument = new FlowDocument();
            myFlowDocument.Blocks.Add(myParagraph);

            this.Content = myFlowDocument;
        }
    }
}

```

```

Imports System.Windows
Imports System.Windows.Media
Imports System.Windows.Controls
Imports System.Windows.Documents

Namespace SDKSample
    Partial Public Class FigureExample
        Inherits Page
        Public Sub New()

            ' Create strings to use as content.
            Dim strFigure As String = "A Figure embeds content into flow content with" & " placement
properties that can be customized" & " independently from the primary content flow"
            Dim strOther As String = "Lorem ipsum dolor sit amet, consectetur adipiscing" & " elit, sed
diam nonummy nibh euismod tincidunt ut laoreet" & " dolore magna aliquam erat volutpat. Ut wisi enim ad" &
" minim veniam, quis nostrud exerci tation ullamcorper" & " suscipit lobortis nisl ut aliquip ex ea commodo
consequat." & " Duis autem vel eum iriure."

            ' Create a Figure and assign content and layout properties to it.
            Dim myFigure As New Figure()
            myFigure.Width = New FigureLength(300)
            myFigure.Height = New FigureLength(100)
            myFigure.Background = Brushes.GhostWhite
            myFigure.HorizontalAnchor = FigureHorizontalAnchor.PageLeft
            Dim myFigureParagraph As New Paragraph(New Run(strFigure))
            myFigureParagraph.FontStyle = FontStyles.Italic
            myFigureParagraph.Background = Brushes.Beige
            myFigureParagraph.Foreground = Brushes.DarkGreen
            myFigure.Blocks.Add(myFigureParagraph)

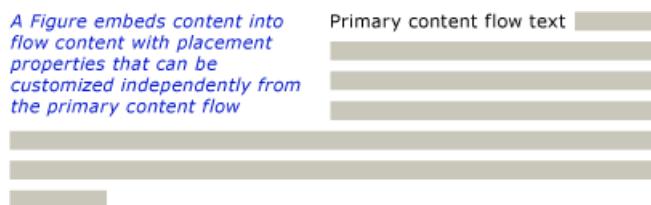
            ' Create the paragraph and add content to it.
            Dim myParagraph As New Paragraph()
            myParagraph.Inlines.Add(myFigure)
            myParagraph.Inlines.Add(New Run(strOther))

            ' Create a FlowDocument and add the paragraph to it.
            Dim myFlowDocument As New FlowDocument()
            myFlowDocument.Blocks.Add(myParagraph)

            Me.Content = myFlowDocument
        End Sub
    End Class
End Namespace

```

La figura seguente illustra come viene eseguito il rendering dell'esempio.



[Figuree Floater](#) si differenziano in diversi modi e vengono usati per scenari diversi.

Figure:

- Può essere posizionato: È possibile impostare i relativi ancoraggi orizzontali e verticali per ancorarlo rispetto alla pagina, al contenuto, alla colonna o al paragrafo. È anche possibile usare le [HorizontalOffset](#) proprietà [VerticalOffset](#) e per specificare gli offset arbitrari.
- È ridimensionabile a più di una colonna: È possibile impostare [Figure](#) l'altezza e la larghezza per i multipli della pagina, del contenuto o dell'altezza o della larghezza della colonna. Nel caso della pagina e

del contenuto, i multipli superiori a 1 non sono consentiti. Ad esempio, è possibile impostare la larghezza di un [Figure](#) oggetto su "0,5 page" o "0,25 content" o "2 Column". È anche possibile impostare l'altezza e la larghezza su valori di pixel assoluti.

- Non viene impaginato: Se il contenuto all'interno [Figure](#) di un oggetto non rientra [Figure](#) nell'oggetto, verrà eseguito il rendering di qualsiasi contenuto e il contenuto rimanente verrà perso

Floater:

- Non può essere posizionato ed esegue il rendering ovunque vi sia uno spazio a disposizione. Non è possibile impostare l'offset o l'[Floater](#) ancoraggio a.
- Non può essere ridimensionato in più di una colonna: Per impostazione predefinita [Floater](#), le dimensioni sono in una colonna. Ha una [Width](#) proprietà che può essere impostata su un valore assoluto in pixel, ma se questo valore è maggiore di una larghezza di colonna viene ignorato e il floater viene ridimensionato in una colonna. È possibile ridimensionarlo a meno di una colonna impostando la larghezza in pixel corretta, ma il ridimensionamento non è relativo alla colonna, quindi "0.5 Column" non è un'[Floater](#) espressione valida per [Width](#). [Floater](#) non ha una proprietà [Height](#) e l'altezza non può essere impostata, l'altezza dipende dal contenuto
- [Floater](#) impagina Se il contenuto nella larghezza specificata si estende a più di un'altezza di colonna, [Floater](#) si interrompe e si impagina nella colonna successiva, nella pagina successiva e così via.

[Figure](#) è il luogo ideale per inserire contenuto autonomo in cui si desidera controllare le dimensioni e il posizionamento e si è certi che il contenuto rientrà nelle dimensioni specificate. [Floater](#) è una posizione ideale per inserire un contenuto più dinamico che scorre in modo simile al contenuto della pagina principale, ma è separato.

LineBreak

[LineBreak](#) causa un'interruzione di riga nel contenuto del flusso. L'esempio seguente illustra l'uso di [LineBreak](#).

```
<FlowDocument xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Paragraph>
    Before the LineBreak in Paragraph.
    <LineBreak />
    After the LineBreak in Paragraph.
    <LineBreak/><LineBreak/>
    After two LineBreaks in Paragraph.
  </Paragraph>

  <Paragraph>
    <LineBreak/>
  </Paragraph>

  <Paragraph>
    After a Paragraph with only a LineBreak in it.
  </Paragraph>
</FlowDocument>
```

Lo screenshot seguente illustra il rendering di questo esempio.

![Screenshot Esempio]di [LineBreak](#)(./media/flow-ovw-linebreakexample.png "Flow_Ovw_LineBreakExample")

Elementi della raccolta di flusso

In molti degli esempi precedenti vengono usati e [BlockCollection](#) [InlineCollection](#) per costruire il contenuto del flusso a livello di codice. Ad esempio, per aggiungere elementi a un [Paragraph](#) oggetto, è possibile usare la sintassi:

```
myParagraph.Inlines.Add(new Run("Some text"));
```

Viene aggiunto un [Run](#) oggetto [InlineCollection](#) alla di [Paragraph](#). Equivale all'oggetto implicito [Run](#) rilevato all'interno di un [Paragraph](#) markup:

```
<Paragraph>
  Some Text
</Paragraph>
```

Come esempio di utilizzo [BlockCollection](#)di, nell'esempio seguente viene creato un nuovo [Section](#) oggetto, quindi viene utilizzato il [Section](#) metodo **Add** per aggiungere [Paragraph](#) un nuovo oggetto al contenuto.

```
Section secx = new Section();
secx.Blocks.Add(new Paragraph(new Run("A bit of text content...")));
```

```
Dim secx As New Section()
secx.Blocks.Add(New Paragraph(New Run("A bit of text content...")))
```

Oltre ad aggiungere elementi a una raccolta di flusso, è anche possibile rimuovere elementi. Nell'esempio seguente viene eliminato l' [Inline](#) ultimo elemento [Span](#)in.

```
spanx.Inlines.Remove(spanx.Inlines.LastInline);
```

```
spanx.Inlines.Remove(spanx.Inlines.LastInline)
```

Nell'esempio seguente vengono cancellati tutti i contenuti ([Inline](#) elementi) [Span](#)da.

```
spanx.Inlines.Clear();
```

```
spanx.Inlines.Clear()
```

Quando si usa il contenuto dinamico a livello di codice, è probabile che queste raccolte vengano usate diffusamente.

Il fatto che un elemento Flow [InlineCollection](#) usi un (inline) [BlockCollection](#) o (blocchi) per contenere gli elementi figlio dipende dal tipo di elementi figlio ([Block](#) o [Inline](#)) che può essere contenuto dall'elemento padre. Le regole di contenimento per gli elementi di contenuto dinamico sono riepilogate nello schema del contenuto nella sezione seguente.

NOTE

È presente un terzo tipo di raccolta usato con contenuto dinamico, [ListCollection](#), ma questa raccolta viene usata solo con un oggetto. [List](#) Sono inoltre disponibili diverse raccolte con [Table](#). Per ulteriori informazioni, vedere [Cenni preliminari sulle tabelle](#).

Schema del contenuto

Dato il gran numero di elementi diversi del contenuto dinamico, può essere complicato tenere traccia del tipo

di elementi figlio che può essere contenuto da un elemento. Il diagramma seguente riepiloga le regole di contenimento per gli elementi di flusso. Le frecce rappresentano le possibili relazioni padre/figlio.

![Diagramma: Schema]di contenimento del contenuto del flusso./media/flow-content-schema.png
"Flow_Content_Schema")

Come si può osservare dal diagramma precedente, gli elementi figlio consentiti per un elemento non vengono necessariamente determinati in base al **Block** fatto che si **Inline** tratti di un elemento o di un elemento. Ad esempio, un **Span** (un **Inline** elemento) può avere **Inline** solo elementi figlio mentre un **Figure** (anche un **Inline** elemento) può avere **Block** solo elementi figlio. Pertanto, un diagramma è utile per determinare rapidamente quale elemento può essere contenuto in un altro elemento. È ad esempio possibile usare il diagramma per determinare come costruire il contenuto del flusso di un oggetto **RichTextBox**.

1. Un **RichTextBox** deve contenere un **FlowDocument** che a sua volta deve contenere **Block** un oggetto derivato da. Di seguito viene riportato il segmento corrispondente del diagramma precedente.

![Diagramma: Regole]di contenimento RichTextBox./media/flow-ovw-schemawalkthrough1.png
"Flow_Ovw_SchemaWalkThrough1")

Fino a questo punto, l'aspetto del markup potrebbe essere simile al seguente.

```
<RichTextBox>
  <FlowDocument>
    <!-- One or more Block-derived object... -->
  </FlowDocument>
</RichTextBox>
```

2. In base al diagramma, è possibile scegliere **Block** tra diversi elementi, tra **Paragraph** cui **Section**, **Table**, **List**, e **BlockUIContainer** (vedere le classi derivate da blocchi precedenti). Supponiamo di volere un **Table**. In base al diagramma precedente, un **Table** oggetto **TableRowGroup** contiene elementi **TableRow** che contengono **TableCell** elementi contenenti un **Block** oggetto derivato da. Di seguito è riportato il segmento **Table** corrispondente per tratto dal diagramma precedente.

![Diagramma: Schema/figlio padre per la tabella./media/flow-ovw-schemawalkthrough2.png
"Flow_Ovw_SchemaWalkThrough2")

Ecco il markup corrispondente.

```
<RichTextBox>
  <FlowDocument>
    <Table>
      <TableRowGroup>
        <TableRow>
          <TableCell>
            <!-- One or more Block-derived object... -->
          </TableCell>
        </TableRow>
      </TableRowGroup>
    </Table>
  </FlowDocument>
</RichTextBox>
```

3. Anche in questo caso, **Block** uno o più elementi sono **TableCell** necessari sotto un. Per rendere più chiaro l'esempio, viene inserito del testo nella cella. Questa operazione può essere eseguita usando **Paragraph** un oggetto **Run** con un elemento. Di seguito sono riportati i segmenti corrispondenti del diagramma che indicano **Paragraph** che un oggetto **Inline** può assumere un elemento **Run** e che **Inline** un oggetto (un elemento) può solo assumere testo normale.

![Diagramma: Schema/figlio padre per il paragrafo./media/flow-ovw-schemawalkthrough3.png

"Flow_Ovw_SchemaWalkThrough3")



Di seguito viene mostrato l'intero esempio a livello di markup.

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <RichTextBox>
        <FlowDocument>

            <!-- Normally a table would have multiple rows and multiple
                cells but this code is for demonstration purposes.-->
            <Table>
                <TableRowGroup>
                    <TableRow>
                        <TableCell>
                            <Paragraph>

                                <!-- The schema does not actually require
                                    explicit use of the Run tag in markup. It
                                    is only included here for clarity. -->
                                <Run>Paragraph in a Table Cell.</Run>
                            </Paragraph>
                        </TableCell>
                    </TableRow>
                </TableRowGroup>
            </Table>

        </FlowDocument>
    </RichTextBox>
</Page>
```

Personalizzazione del testo

In genere il testo è il tipo di contenuto più diffuso in un documento dinamico. Sebbene gli oggetti illustrati in precedenza possano essere usati per controllare la maggior parte degli aspetti della modalità di rendering del testo, esistono alcuni altri metodi per la personalizzazione del testo descritti in questa sezione.

Decorazioni di testo

Le decorazioni di testo consentono di applicare gli effetti sottolineato, linea sopra, linea di base e barrato al testo, come illustrato nelle figure riportate di seguito. Questi decori vengono aggiunti utilizzando [TextDecorations](#) la proprietà esposta da un numero di oggetti, tra [Inline](#) cui [Paragraph](#), [TextBlock](#), e [TextBox](#).

Nell'esempio seguente viene illustrato come impostare la proprietà [TextDecorations](#) di un oggetto [Paragraph](#).

```
<FlowDocument ColumnWidth="200">
    <Paragraph TextDecorations="Strikethrough">
        This text will render with the strikethrough effect.
    </Paragraph>
</FlowDocument>
```

```
Paragraph parx = new Paragraph(new Run("This text will render with the strikethrough effect."));
parx.TextDecorations = TextDecorations.Strikethrough;
```

```
Dim parx As New Paragraph(New Run("This text will render with the strikethrough effect."))
parx.TextDecorations = TextDecorations.Strikethrough
```

La figura seguente illustra il rendering di questo esempio.

![Screenshot Testo con effetto]barrato predefinito./media/inline-textdec-strike.png "Inline_TextDec_Strike"

Nelle figure seguenti viene illustrato il rendering, rispettivamente, della **linea di base**, della linea di base e delle decorazioni di **sottolineatura**.

This text will render with the default overline effect.

This text will render with the default baseline effect.

![Screenshot Testo con effetto]di sottolineatura predefinito./media/inline-textdec-under.png

"Inline_TextDec_Under")

Opzioni tipografiche

La **Typography** proprietà è esposta dalla maggior parte del contenuto relativo al **TextElement** flusso, **TextBlock** tra cui **TextBox**, **FlowDocument**, e. Questa proprietà viene usata per controllare le caratteristiche o le variazioni tipografiche del testo (ad esempio maiuscoletto o maiuscolo, pedici e sottoscrizioni e così via).

Nell'esempio seguente viene illustrato come impostare l' **Typography** attributo utilizzando **Paragraph** come elemento di esempio.

```
<Paragraph
    TextAlignment="Left"
    FontSize="18"
    FontFamily="Palatino Linotype"
    Typography.NumeralStyle="OldStyle"
    Typography.Fraction="Stacked"
    Typography.Variants="Inferior"
>
<Run>
    This text has some altered typography characteristics. Note
    that use of an open type font is necessary for most typographic
    properties to be effective.
</Run>
<LineBreak/><LineBreak/>
<Run>
    0123456789 10 11 12 13
</Run>
<LineBreak/><LineBreak/>
<Run>
    1/2 2/3 3/4
</Run>
</Paragraph>
```

La figura seguente illustra il rendering di questo esempio.

![Screenshot Testo con tipografia]modificata./media/textelement-typog.png "TextElement_Typog")

La figura seguente mostra invece il rendering dello stesso esempio con le proprietà tipografiche predefinite.

![Screenshot Testo con tipografia]modificata./media/textelement-typog-default.png
"TextElement_Typog_Default")

Nell'esempio seguente viene illustrato come impostare la **Typography** proprietà a livello di codice.

```

Paragraph par = new Paragraph();

Run runText = new Run(
    "This text has some altered typography characteristics. Note" +
    "that use of an open type font is necessary for most typographic" +
    "properties to be effective.");
Run runNumerals = new Run("0123456789 10 11 12 13");
Run runFractions = new Run("1/2 2/3 3/4");

par.Inlines.Add(runText);
par.Inlines.Add(new LineBreak());
par.Inlines.Add(new LineBreak());
par.Inlines.Add(runNumerals);
par.Inlines.Add(new LineBreak());
par.Inlines.Add(new LineBreak());
par.Inlines.Add(runFractions);

par.TextAlignment = TextAlignment.Left;
par.FontSize = 18;
par.FontFamily = new FontFamily("Palatino Linotype");

par.Typography.NumeralStyle = FontNumeralStyle.OldStyle;
par.Typography.Fraction = FontFraction.Stacked;
par.Typography.Variants = FontVariants.Inferior;

```

```

Dim par As New Paragraph()

Dim runText As New Run("This text has some altered typography characteristics. Note" & "that use of an
open type font is necessary for most typographic" & "properties to be effective.")
Dim runNumerals As New Run("0123456789 10 11 12 13")
Dim runFractions As New Run("1/2 2/3 3/4")

par.Inlines.Add(runText)
par.Inlines.Add(New LineBreak())
par.Inlines.Add(New LineBreak())
par.Inlines.Add(runNumerals)
par.Inlines.Add(New LineBreak())
par.Inlines.Add(New LineBreak())
par.Inlines.Add(runFractions)

par.TextAlignment = TextAlignment.Left
par.FontSize = 18
par.FontFamily = New FontFamily("Palatino Linotype")

par.Typography.NumeralStyle = FontNumeralStyle.OldStyle
par.Typography.Fraction = FontFraction.Stacked
par.Typography.Variants = FontVariants.Inferior

```

Per ulteriori informazioni sulla tipografia, vedere [Typography in WPF](#).

Vedere anche

- [Text](#)
- [Funzionalità tipografiche di WPF](#)
- [Procedure relative alle proprietà](#)
- [Panoramica sul modello di contenuto TextElement](#)
- [Cenni preliminari sul controllo RichTextBox](#)
- [Documenti in WPF](#)
- [Cenni preliminari sull'elemento Table](#)
- [Cenni preliminari sulle annotazioni](#)

Cenni preliminari sul modello di contenuto

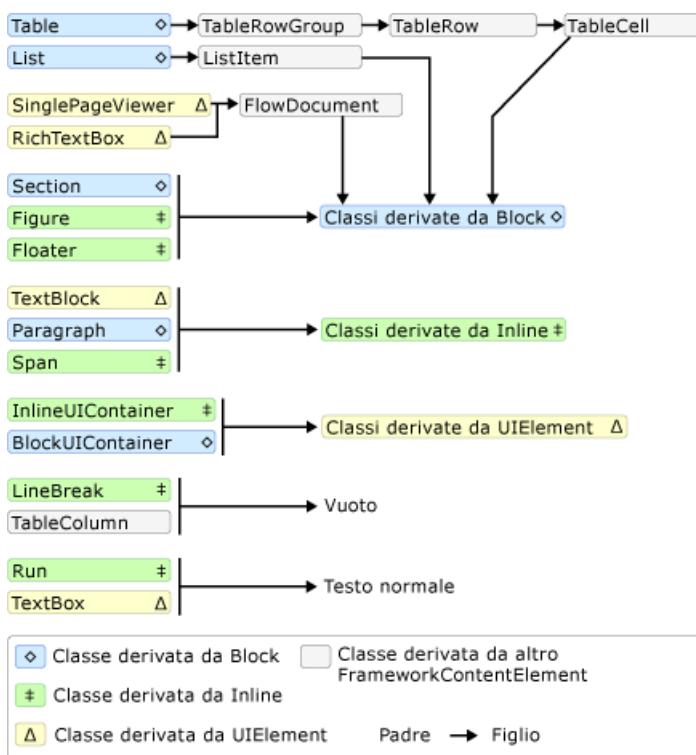
TextElement

08/11/2019 • 7 minutes to read • [Edit Online](#)

Questa panoramica sul modello di contenuto descrive il contenuto supportato per un [TextElement](#). La classe [Paragraph](#) è un tipo di [TextElement](#). Un modello di contenuto descrive gli oggetti o gli elementi che possono essere contenuti in altri oggetti o elementi. Questa panoramica riepiloga il modello di contenuto usato per gli oggetti derivati da [TextElement](#). Per altre informazioni, vedere [Cenni preliminari sui documenti dinamici](#).

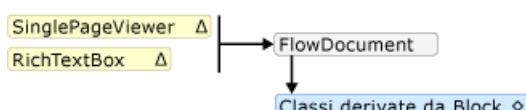
Diagramma del modello di contenuto

Il diagramma seguente riepiloga il modello di contenuto per le classi derivate da [TextElement](#) e il modo in cui le altre classi non [TextElement](#) rientrano in questo modello.



Come si può osservare dal diagramma precedente, gli elementi figlio consentiti per un elemento non vengono necessariamente determinati in base al fatto che una classe deriva dalla classe [Block](#) o da una classe [Inline](#). Ad esempio, un [Span](#) (una classe derivata da [Inline](#)) può avere solo elementi figlio [Inline](#), ma un [Figure](#) (anche una classe derivata da [Inline](#)) può avere solo [Block](#) elementi figlio. Pertanto, un diagramma è utile per determinare rapidamente quale elemento può essere contenuto in un altro elemento. È ad esempio possibile usare il diagramma per determinare come costruire il contenuto del flusso di un [RichTextBox](#).

1. Un [RichTextBox](#) deve contenere una [FlowDocument](#) che a sua volta deve contenere un oggetto derivato da [Block](#). Di seguito viene riportato il segmento corrispondente del diagramma precedente.



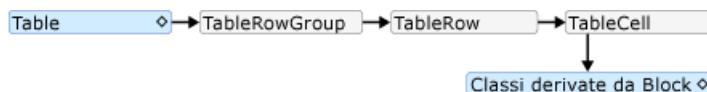
Fino a questo punto, l'aspetto del markup potrebbe essere simile al seguente.

```

<RichTextBox>
  <FlowDocument>
    <!-- One or more Block-derived object... -->
  </FlowDocument>
</RichTextBox>

```

2. In base al diagramma, è possibile scegliere tra diversi elementi **Block** inclusi **Paragraph**, **Section**, **Table**, **Liste** **BlockUIContainer** (vedere classi derivate da blocchi nel diagramma precedente). Supponiamo di volere un **Table**. In base al diagramma precedente, un **Table** contiene un **TableRowGroup** contenente elementi **TableRow** che contengono **TableCell** elementi che contengono un oggetto derivato da **Block**. Di seguito è riportato il segmento corrispondente per **Table** tratto dal diagramma precedente.



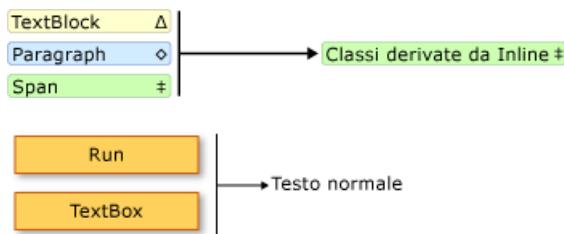
Ecco il markup corrispondente.

```

<RichTextBox>
  <FlowDocument>
    <Table>
      <TableRowGroup>
        <TableRow>
          <TableCell>
            <!-- One or more Block-derived object... -->
          </TableCell>
        </TableRow>
      </TableRowGroup>
    </Table>
  </FlowDocument>
</RichTextBox>

```

3. Anche in questo caso, uno o più elementi **Block** sono necessari sotto una **TableCell**. Per rendere più chiaro l'esempio, viene inserito del testo nella cella. Questa operazione può essere eseguita usando un **Paragraph** con un elemento **Run**. Di seguito sono riportati i segmenti corrispondenti del diagramma che mostrano che un **Paragraph** può assumere un elemento **Inline** e che un **Run** (un elemento **Inline**) può solo assumere testo normale.



Di seguito viene mostrato l'intero esempio a livello di markup.

```

<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <RichTextBox>
        <FlowDocument>

            <!-- Normally a table would have multiple rows and multiple
                 cells but this code is for demonstration purposes.-->
            <Table>
                <TableRowGroup>
                    <TableRow>
                        <TableCell>
                            <Paragraph>

                                <!-- The schema does not actually require
                                     explicit use of the Run tag in markup. It
                                     is only included here for clarity. -->
                                <Run>Paragraph in a Table Cell.</Run>
                            </Paragraph>
                        </TableCell>
                    </TableRow>
                </TableRowGroup>
            </Table>

        </FlowDocument>
    </RichTextBox>
</Page>

```

Uso del contenuto TextElement a livello di codice

Il contenuto di un [TextElement](#) è costituito da raccolte, quindi la modifica a livello di codice del contenuto degli oggetti [TextElement](#) viene eseguita con queste raccolte. Sono disponibili tre diverse raccolte utilizzate dalle classi derivate da [TextElement](#):

- [InlineCollection](#): rappresenta una raccolta di elementi di [Inline](#). [InlineCollection](#) definisce il contenuto figlio consentito degli elementi [Paragraph](#), [Span](#) e [TextBlock](#).
- [BlockCollection](#): rappresenta una raccolta di elementi di [Block](#). [BlockCollection](#) definisce il contenuto figlio consentito degli elementi [FlowDocument](#), [Section](#), [ListItem](#), [TableCell](#), [Floater](#) e [Figure](#).
- [ListCollection](#): un elemento di contenuto del flusso che rappresenta un particolare elemento di contenuto in una [List](#) ordinata o non ordinata.

È possibile modificare (aggiungere o rimuovere elementi) da queste raccolte usando le rispettive proprietà di **Inlines**, **Blockse ListItems**. Negli esempi seguenti viene illustrato come modificare il contenuto di un intervallo utilizzando la proprietà **Inlines**.

NOTE

Per modificare il contenuto di una tabella vengono usate diverse raccolte, che tuttavia non vengono illustrate in questa sezione. Per ulteriori informazioni, vedere [Cenni preliminari sulle tabelle](#).

Nell'esempio seguente viene creato un nuovo oggetto [Span](#), quindi viene utilizzato il metodo [Add](#) per aggiungere due esecuzioni di testo come elementi figlio di contenuto del [Span](#).

```

Span spanx = new Span();
spanx.Inlines.Add(new Run("A bit of text content..."));
spanx.Inlines.Add(new Run("A bit more text content..."));

```

```
Dim spanx As New Span()
spanx.Inlines.Add(New Run("A bit of text content..."))
spanx.Inlines.Add(New Run("A bit more text content..."))
```

Nell'esempio seguente viene creato un nuovo elemento [Run](#) e inserito all'inizio del [Span](#).

```
Run runx = new Run("Text to insert...");
spanx.Inlines.InsertBefore(spanx.Inlines.FirstInline, runx);
```

```
Dim runx As New Run("Text to insert...")
spanx.Inlines.InsertBefore(spanx.Inlines.FirstInline, runx)
```

Nell'esempio seguente viene eliminato l'ultimo elemento [Inline](#) nell'[Span](#).

```
spanx.Inlines.Remove(spanx.Inlines.LastInline);
```

```
spanx.Inlines.Remove(spanx.Inlines.LastInline)
```

Nell'esempio seguente vengono cancellati tutti i contenuti (elementi [Inline](#)) dall'[Span](#).

```
spanx.Inlines.Clear();
```

```
spanx.Inlines.Clear()
```

Tipi che condividono questo modello di contenuto

I tipi seguenti ereditano dalla classe [TextElement](#) e possono essere usati per visualizzare il contenuto descritto in questa panoramica.

[Bold](#), [Figure](#), [Floater](#), [Hyperlink](#), [InlineUIContainer](#), [Italic](#), [LineBreak](#), [List](#), [ListItem](#), [Paragraph](#), [Run](#), [Section](#), [Span](#), [Table](#), [Underline](#).

Si noti che questo elenco include solo i tipi non astratti distribuiti con la Windows SDK. È possibile utilizzare altri tipi che ereditano da [TextElement](#).

Tipi in grado di contenere oggetti [TextElement](#)

Vedere [modello di contenuto WPF](#).

Vedere anche

- [Modificare un oggetto FlowDocument tramite la proprietà Blocks](#)
- [Modificare elementi di contenuto dinamico tramite la proprietà Blocks](#)
- [Modificare un oggetto FlowDocument tramite la proprietà Blocks](#)
- [Modificare le colonne di una tabella tramite la proprietà Columns](#)
- [Modificare i gruppi di righe di una tabella tramite la proprietà RowGroups](#)

Cenni preliminari sull'elemento Table

23/11/2019 • 14 minutes to read • [Edit Online](#)

Table è un elemento a livello di blocco che supporta la presentazione basata su griglia del contenuto del documento dinamico. La flessibilità di questo elemento lo rende molto utile, ma al contempo ne complica la comprensione e l'uso corretto.

In questo argomento sono contenute le seguenti sezioni.

- [Nozioni di base su Table](#)
- [Differenze tra Table e Grid](#)
- [Struttura di base di Table](#)
- [Contenimento di Table](#)
- [Raggruppamenti di righe](#)
- [Precedenza del rendering dello sfondo](#)
- [Estensione su più righe o colonne](#)
- [Compilazione di un elemento Table con codice](#)
- [Argomenti correlati]

Nozioni fondamentali sulle tabelle

Differenze tra Table e Grid

Table e **Grid** condividono alcune funzionalità comuni, ma ognuna è più adatta per scenari diversi. Una **Table** è progettata per l'uso all'interno del contenuto di flusso. per altre informazioni sul contenuto dinamico, vedere [Cenni preliminari sui documenti dinamici](#). Gli elementi **Grid** sono particolarmente adatti all'interno di moduli (in generale in un qualsiasi punto all'esterno del contenuto dinamico). All'interno di un **FlowDocument**, **Table** supporta i comportamenti del contenuto di flusso come paginazione, riflusso di colonne e selezione di contenuto mentre un **Grid** non lo è. Un **Grid** d'altra parte viene utilizzato meglio all'esterno di un **FlowDocument** per molti motivi, tra cui **Grid** aggiunge elementi in base a un indice di riga e di colonna, **Table** non lo è. L'elemento **Grid** consente la sovrapposizione di contenuto figlio, consentendo l'esistenza di più di un elemento all'interno di una singola "cella". **Table** non supporta la sovrapposizione. Gli elementi figlio di un **Grid** possono essere posizionati in modo assoluto rispetto all'area dei limiti "Cell". **Table** non supporta questa funzionalità. Infine, un **Grid** richiede un minor numero di risorse e quindi una **Table** quindi considerare l'uso di una **Grid** per migliorare le prestazioni.

Struttura di base di Table

Table fornisce una presentazione basata su griglia costituita da colonne (rappresentate da elementi **TableColumn**) e da righe (rappresentate da elementi **TableRow**). gli elementi **TableColumn** non ospitano contenuto; definiscono semplicemente le colonne e le caratteristiche delle colonne. gli elementi **TableRow** devono essere ospitati in un elemento **TableRowGroup**, che definisce un raggruppamento di righe per la tabella. gli elementi **TableCell**, che contengono il contenuto effettivo che deve essere presentato dalla tabella, devono essere ospitati in un elemento **TableRow**. **TableCell** possono contenere solo elementi che derivano da **Block**. Gli elementi figlio validi per un **TableCell** includono:

- [BlockUIContainer](#)
- [List](#)

- [Paragraph](#)
- [Section](#)
- [Table](#)

NOTE

[TableCell](#) elementi non possono ospitare direttamente contenuto di testo. Per altre informazioni sulle regole di contenimento per gli elementi di contenuto del flusso come [TableCell](#), vedere [Cenni preliminari sui documenti dinamici](#).

NOTE

[Table](#) è simile all'elemento [Grid](#) ma dispone di più funzionalità e, di conseguenza, richiede un sovraccarico di risorse maggiore.

Nell'esempio seguente viene definita una semplice tabella 2 x 3 con XAML.

```

<!--
Table is a Block element, and as such must be hosted in a container
for Block elements. FlowDocument provides such a container.
-->
<FlowDocument>
  <Table>
    <!--
      This table has 3 columns, each described by a TableColumn
      element nested in a Table.Columns collection element.
    -->
    <Table.Columns>
      < TableColumn />
      < TableColumn />
      < TableColumn />
    </Table.Columns>
    <!--
      This table includes a single TableRowGroup which hosts 2 rows,
      each described by a TableRow element.
    -->
    <TableRowGroup>
      <!--
        Each of the 2 TableRow elements hosts 3 cells, described by
        TableCell elements.
      -->
      <TableRow>
        <TableCell>
          <!--
            TableCell elements may only host elements derived from Block.
            In this example, Paragraph elements serve as the ultimate content
            containers for the cells in this table.
          -->
          <Paragraph>Cell at Row 1 Column 1</Paragraph>
        </TableCell>
        <TableCell>
          <Paragraph>Cell at Row 1 Column 2</Paragraph>
        </TableCell>
        <TableCell>
          <Paragraph>Cell at Row 1 Column 3</Paragraph>
        </TableCell>
      </TableRow>
      <TableRow>
        <TableCell>
          <Paragraph>Cell at Row 2 Column 1</Paragraph>
        </TableCell>
        <TableCell>
          <Paragraph>Cell at Row 2 Column 2</Paragraph>
        </TableCell>
        <TableCell>
          <Paragraph>Cell at Row 2 Column 3</Paragraph>
        </TableCell>
      </TableRow>
    </TableRowGroup>
  </Table>
</FlowDocument>

```

La figura seguente illustra il rendering di questo esempio.

Cell at Row 1 Column 1	Cell at Row 1 Column 2	Cell at Row 1 Column 3
Cell at Row 2 Column 1	Cell at Row 2 Column 2	Cell at Row 2 Column 3

Contenimento di Table

Table deriva dall'elemento [Block](#) e rispetta le regole comuni per gli elementi [Block](#) Level. Un elemento [Table](#) può essere contenuto da uno qualsiasi degli elementi seguenti:

- [FlowDocument](#)
- [TableCell](#)
- [ListBoxItem](#)
- [ListViewItem](#)
- [Section](#)
- [Floater](#)
- [Figure](#)

Raggruppamenti di righe

L'elemento [TableRowGroup](#) consente di raggruppare in modo arbitrario le righe all'interno di una tabella. Ogni riga di una tabella deve appartenere a un raggruppamento di righe. Spesso, le righe appartenenti allo stesso gruppo condividono un intento comune ed è possibile usare uno stile comune per il gruppo. Un uso tipico dei raggruppamenti di righe consiste nel separare le righe con scopi specifici, ad esempio le righe di titolo, intestazione e più di pagina, dal contenuto principale della tabella.

Nell'esempio seguente viene usato XAML per definire una tabella con righe di intestazione e più di pagina con stile.

```

<Table>
  <Table.Resources>
    <!-- Style for header/footer rows. -->
    <Style x:Key="headerFooterRowStyle" TargetType="{x:Type TableRowGroup}">
      <Setter Property="FontWeight" Value="DemiBold"/>
      <Setter Property="FontSize" Value="16"/>
      <Setter Property="Background" Value="LightGray"/>
    </Style>

    <!-- Style for data rows. -->
    <Style x:Key="dataRowStyle" TargetType="{x:Type TableRowGroup}">
      <Setter Property="FontSize" Value="12"/>
      <Setter Property="FontStyle" Value="Italic"/>
    </Style>
  </Table.Resources>

  <Table.Columns>
    <TableColumn/> <TableColumn/> <TableColumn/> <TableColumn/>
  </Table.Columns>

  <!-- This TableRowGroup hosts a header row for the table. -->
  <TableRowGroup Style="{StaticResource headerFooterRowStyle}">
    <TableRow>
      <TableCell/>
      <TableCell><Paragraph>Gizmos</Paragraph></TableCell>
      <TableCell><Paragraph>Thingamajigs</Paragraph></TableCell>
      <TableCell><Paragraph>Doohickies</Paragraph></TableCell>
    </TableRow>
  </TableRowGroup>

  <!-- This TableRowGroup hosts the main data rows for the table. -->
  <TableRowGroup Style="{StaticResource dataRowStyle}">
    <TableRow>
      <TableCell><Paragraph Foreground="Blue">Blue</Paragraph></TableCell>
      <TableCell><Paragraph>1</Paragraph></TableCell>
      <TableCell><Paragraph>2</Paragraph></TableCell>
      <TableCell><Paragraph>3</Paragraph></TableCell>
    </TableRow>
    <TableRow>
      <TableCell><Paragraph Foreground="Red">Red</Paragraph></TableCell>
      <TableCell><Paragraph>1</Paragraph></TableCell>
      <TableCell><Paragraph>2</Paragraph></TableCell>
      <TableCell><Paragraph>3</Paragraph></TableCell>
    </TableRow>
    <TableRow>
      <TableCell><Paragraph Foreground="Green">Green</Paragraph></TableCell>
      <TableCell><Paragraph>1</Paragraph></TableCell>
      <TableCell><Paragraph>2</Paragraph></TableCell>
      <TableCell><Paragraph>3</Paragraph></TableCell>
    </TableRow>
  </TableRowGroup>

  <!-- This TableRowGroup hosts a footer row for the table. -->
  <TableRowGroup Style="{StaticResource headerFooterRowStyle}">
    <TableRow>
      <TableCell><Paragraph>Totals</Paragraph></TableCell>
      <TableCell><Paragraph>3</Paragraph></TableCell>
      <TableCell><Paragraph>6</Paragraph></TableCell>
      <TableCell>
        <Table></Table>
      </TableCell>
    </TableRow>
  </TableRowGroup>
</Table>

```

La figura seguente illustra il rendering di questo esempio.

	Gizmos	Thingamajigs	Doohickies
<i>Blue</i>	1	2	3
<i>Red</i>	1	2	3
<i>Green</i>	1	2	3
Totals	3	6	9

Precedenza del rendering dello sfondo

Il rendering di elementi Table viene eseguito nell'ordine seguente (z order dal più basso al più alto). Quest'ordine non può essere modificato. Ad esempio, per questi elementi non esiste una proprietà "z order" che è possibile usare per l'override dell'ordine stabilito.

1. [Table](#)
2. [TableColumn](#)
3. [TableRowGroup](#)
4. [TableRow](#)
5. [TableCell](#)

L'esempio seguente definisce i colori di sfondo per ogni elemento all'interno di una tabella.

```
<Table Background="Yellow">
  <Table.Columns>
    < TableColumn />
    < TableColumn Background="LightGreen" />
    < TableColumn />
  </Table.Columns>
  < TableRowGroup >
    < TableRow >
      < TableCell />< TableCell />< TableCell />
    </ TableRow >
  </ TableRowGroup >
  < TableRowGroup Background="Tan" >
    < TableRow >
      < TableCell />< TableCell />< TableCell />
    </ TableRow >
    < TableRow Background="LightBlue" >
      < TableCell />< TableCell Background="Purple" />< TableCell />
    </ TableRow >
    < TableRow >
      < TableCell />< TableCell />< TableCell />
    </ TableRow >
  </ TableRowGroup >
  < TableRowGroup >
    < TableRow >
      < TableCell />< TableCell />< TableCell />
    </ TableRow >
  </ TableRowGroup >
</Table>
```

La figura seguente illustra il rendering di questo esempio (visualizzazione dei soli colori di sfondo).



Estensione su più righe o colonne

È possibile configurare le celle della tabella in modo che si estendano su più righe o colonne utilizzando

rispettivamente gli attributi [RowSpan](#) o [ColumnSpan](#).

L'esempio seguente definisce una cella che si estende su tre colonne.

```
<Table>
  <Table.Columns>
    <TableColumn/>
    <TableColumn/>
    <TableColumn/>
  </Table.Columns>

  <TableRowGroup>
    <TableRow>
      <TableCell ColumnSpan="3" Background="Cyan">
        <Paragraph>This cell spans all three columns.</Paragraph>
      </TableCell>
    </TableRow>
    <TableRow>
      <TableCell Background="LightGray"><Paragraph>Cell 1</Paragraph></TableCell>
      <TableCell Background="LightGray"><Paragraph>Cell 2</Paragraph></TableCell>
      <TableCell Background="LightGray"><Paragraph>Cell 3</Paragraph></TableCell>
    </TableRow>
  </TableRowGroup>
</Table>
```

La figura seguente illustra il rendering di questo esempio.

This cell spans all three columns.		
Cell 1	Cell 2	Cell 3

Compilazione di un elemento Table con codice

Negli esempi seguenti viene illustrato come creare una [Table](#) a livello di codice e compilarla con contenuto. Il contenuto della tabella è ripartito in cinque righe (rappresentate da [TableRow](#) oggetti contenuti in un oggetto [RowGroups](#)) e sei colonne (rappresentate da oggetti [TableColumn](#)). Le righe vengono usate per scopi di presentazione diversi, ad esempio una riga è destinata a contenere il titolo dell'intera tabella, una riga di intestazione a descrivere le colonne di dati nella tabella e una riga di più di pagina a fornire informazioni di riepilogo. Si noti che i concetti di righe di "titolo", "intestazione" e "più di pagina" non sono inerenti alla tabella, ma fanno semplicemente riferimento a righe con caratteristiche diverse. Le celle della tabella contengono il contenuto effettivo, che può essere costituito da testo, immagini o quasi qualsiasi altro elemento interfaccia utente.

In primo luogo, viene creato un [FlowDocument](#) per ospitare la [Table](#) e viene creato un nuovo [Table](#) e viene aggiunto al contenuto del [FlowDocument](#).

```
// Create the parent FlowDocument...
flowDoc = new FlowDocument();

// Create the Table...
table1 = new Table();
// ...and add it to the FlowDocument Blocks collection.
flowDoc.Blocks.Add(table1);

// Set some global formatting properties for the table.
table1.CellSpacing = 10;
table1.Background = Brushes.White;
```

```

' Create the parent FlowDocument...
flowDoc = New FlowDocument()

' Create the Table...
table1 = New Table()

' ...and add it to the FlowDocument Blocks collection.
flowDoc.Blocks.Add(table1)

' Set some global formatting properties for the table.
table1.CellSpacing = 10
table1.Background = Brushes.White

```

Successivamente, vengono creati sei oggetti [TableColumn](#) e aggiunti alla raccolta [Columns](#) della tabella, con una formattazione applicata.

NOTE

Si noti che la raccolta [Columns](#) della tabella usa l'indicizzazione standard in base zero.

```

// Create 6 columns and add them to the table's Columns collection.
int numberOfRowsColumns = 6;
for (int x = 0; x < numberOfRowsColumns; x++)
{
    table1.Columns.Add(new TableColumn());

    // Set alternating background colors for the middle columns.
    if(x%2 == 0)
        table1.Columns[x].Background = Brushes.Beige;
    else
        table1.Columns[x].Background = Brushes.LightSteelBlue;
}

```

```

' Create 6 columns and add them to the table's Columns collection.
Dim numberOfRowsColumns = 6
Dim x
For x = 0 To numberOfRowsColumns
    table1.Columns.Add(new TableColumn())

    ' Set alternating background colors for the middle columns.
    If x Mod 2 = 0 Then
        table1.Columns(x).Background = Brushes.Beige
    Else
        table1.Columns(x).Background = Brushes.LightSteelBlue
    End If
Next x

```

Viene quindi creata una riga del titolo che viene aggiunta alla tabella con l'applicazione di alcuni elementi di formattazione. La riga del titolo può contenere una sola cella che si estende su tutte e sei le colonne della tabella.

```

// Create and add an empty TableRowGroup to hold the table's Rows.
table1.RowGroups.Add(new TableRowGroup());

// Add the first (title) row.
table1.RowGroups[0].Rows.Add(new TableRow());

// Alias the current working row for easy reference.
TableRow currentRow = table1.RowGroups[0].Rows[0];

// Global formatting for the title row.
currentRow.Background = Brushes.Silver;
currentRow.FontSize = 40;
currentRow.FontWeight = System.Windows.FontWeights.Bold;

// Add the header row with content,
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("2004 Sales Project"))));
// and set the row to span all 6 columns.
currentRow.Cells[0].ColumnSpan = 6;

```

```

' Create and add an empty TableRowGroup to hold the table's Rows.
table1.RowGroups.Add(new TableRowGroup())

' Add the first (title) row.
table1.RowGroups(0).Rows.Add(new TableRow())

' Alias the current working row for easy reference.
Dim currentRow As New TableRow()
currentRow = table1.RowGroups(0).Rows(0)

' Global formatting for the title row.
currentRow.Background = Brushes.Silver
currentRow.FontSize = 40
currentRow.FontWeight = System.Windows.FontWeights.Bold

' Add the header row with content,
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("2004 Sales Project"))))
' and set the row to span all 6 columns.
currentRow.Cells(0).ColumnSpan = 6

```

Successivamente, viene creata e aggiunta alla tabella una riga di intestazione, per la quale vengono create celle in cui viene inserito contenuto.

```

// Add the second (header) row.
table1.RowGroups[0].Rows.Add(new TableRow());
currentRow = table1.RowGroups[0].Rows[1];

// Global formatting for the header row.
currentRow.FontSize = 18;
currentRow.FontWeight = FontWeights.Bold;

// Add cells with content to the second row.
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Product"))));
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Quarter 1"))));
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Quarter 2"))));
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Quarter 3"))));
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Quarter 4"))));
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("TOTAL"))));

```

```

' Add the second (header) row.
table1.RowGroups(0).Rows.Add(new TableRow())
currentRow = table1.RowGroups(0).Rows(1)

' Global formatting for the header row.
currentRow.FontSize = 18
currentRow.FontWeight = FontWeights.Bold

' Add cells with content to the second row.
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Product"))))
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Quarter 1"))))
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Quarter 2"))))
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Quarter 3"))))
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Quarter 4"))))
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("TOTAL"))))

```

Successivamente, viene creata e aggiunta alla tabella una riga per i dati, per la quale vengono create celle in cui viene inserito contenuto. La compilazione di questa riga è simile alla compilazione della riga di intestazione, con l'applicazione di una formattazione leggermente diversa.

```

// Add the third row.
table1.RowGroups[0].Rows.Add(new TableRow());
currentRow = table1.RowGroups[0].Rows[2];

// Global formatting for the row.
currentRow.FontSize = 12;
currentRow.FontWeight = FontWeights.Normal;

// Add cells with content to the third row.
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Widgets"))));
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("$50,000"))));
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("$55,000"))));
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("$60,000"))));
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("$65,000"))));
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("$230,000"))));

// Bold the first cell.
currentRow.Cells[0].FontWeight = FontWeights.Bold;

```

```

' Add the third row.
table1.RowGroups(0).Rows.Add(new TableRow())
currentRow = table1.RowGroups(0).Rows(2)

' Global formatting for the row.
currentRow.FontSize = 12
currentRow.FontWeight = FontWeights.Normal

' Add cells with content to the third row.
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Widgets"))))
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("$50,000"))))
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("$55,000"))))
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("$60,000"))))
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("$65,000"))))
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("$230,000"))))

' Bold the first cell.
currentRow.Cells(0).FontWeight = FontWeights.Bold

```

Infine, viene creata, aggiunta e formattata una riga di piè di pagina. Analogamente alla riga del titolo, il piè di pagina contiene una sola cella che si estende su tutte e sei le colonne della tabella.

```
table1.RowGroups[0].Rows.Add(new TableRow());
currentRow = table1.RowGroups[0].Rows[3];

// Global formatting for the footer row.
currentRow.Background = Brushes.LightGray;
currentRow.FontSize = 18;
currentRow.FontWeight = System.Windows.FontWeights.Normal;

// Add the header row with content,
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Projected 2004 Revenue: $810,000"))));
// and set the row to span all 6 columns.
currentRow.Cells[0].ColumnSpan = 6;
```

```
table1.RowGroups(0).Rows.Add(new TableRow())
currentRow = table1.RowGroups(0).Rows(3)

' Global formatting for the footer row.
currentRow.Background = Brushes.LightGray
currentRow.FontSize = 18
currentRow.FontWeight = System.Windows.FontWeights.Normal

' Add the header row with content,
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Projected 2004 Revenue: $810,000"))))
' and set the row to span all 6 columns.
currentRow.Cells(0).ColumnSpan = 6
```

Vedere anche

- [Cenni preliminari sui documenti dinamici](#)
- [Definire un oggetto Table con XAML](#)
- [Documenti in WPF](#)
- [Usare elementi di contenuto dinamico](#)

Argomenti sulle procedure relative a elementi di contenuto del flusso

23/10/2019 • 2 minutes to read • [Edit Online](#)

Negli argomenti di questa sezione viene descritto come eseguire attività comuni utilizzando vari elementi di contenuto dinamico e le funzionalità correlate.

In questa sezione

[Regolare la spaziatura tra paragrafi](#)

[Compilare oggetti Table a livello di codice](#)

[Modificare l'oggetto FlowDirection del contenuto a livello di codice](#)

[Modificare la proprietà TextWrapping a livello di codice](#)

[Definire un oggetto Table con XAML](#)

[Modificare la tipografia del testo](#)

[Attivare l'enumerazione TextTrimming](#)

[Inserire un elemento in un testo a livello di codice](#)

[Modificare elementi di contenuto dinamico tramite la proprietà Blocks](#)

[Modificare gli elementi di contenuto del flusso tramite la proprietà Inlines](#)

[Modificare un oggetto FlowDocument tramite la proprietà Blocks](#)

[Modificare le colonne di una tabella tramite la proprietà Columns](#)

[Modificare i gruppi di righe di una tabella tramite la proprietà RowGroups](#)

[Usare elementi di contenuto dinamico](#)

[Usare gli attributi di separazione delle colonne di un oggetto FlowDocument](#)

Riferimenti

[FlowDocument](#)

[Block](#)

[Inline](#)

Sezioni correlate

[Documenti in WPF](#)

Procedura: Regolare la spaziatura tra i paragrafi

23/10/2019 • 2 minutes to read • [Edit Online](#)

In questo esempio viene illustrato come modificare o eliminare la spaziatura tra paragrafi nel contenuto dinamico.

Nel contenuto dinamico, lo spazio aggiuntivo che viene visualizzato tra i paragrafi è il risultato dei margini impostati per queste paragrafi. di conseguenza, la spaziatura tra paragrafi può essere controllata regolando i margini in paragrafi. Per eliminare completamente la spaziatura aggiuntiva tra due paragrafi, impostare i margini di paragrafi vengano **0**. Per ottenere la spaziatura uniforme tra i paragrafi di un intero [FlowDocument](#), applicare uno stile per impostare un valore di un margine uniforme per tutti i paragrafi di [FlowDocument](#).

È importante notare che i margini per due paragrafi adiacenti "comprimerà" per il valore più grande tra i due margini, anziché raddoppiati. Pertanto, se due paragrafi adiacenti presentano il margine di 20 e 40 pixel rispettivamente, lo spazio tra i paragrafi 40 pixel, il più elevato tra i valori per i due margini.

Esempio

L'esempio seguente usa lo stile per impostare il margine per tutti [Paragraph](#) elementi in un [FlowDocument](#) al **0**, che elimina in modo efficace la spaziatura tra paragrafi nel [FlowDocument](#).

```
<FlowDocument>
  <FlowDocument.Resources>
    <!-- This style is used to set the margins for all paragraphs in the FlowDocument to 0. -->
    <Style TargetType="{x:Type Paragraph}">
      <Setter Property="Margin" Value="0"/>
    </Style>
  </FlowDocument.Resources>

  <Paragraph>
    Spacing between paragraphs is caused by margins set on the paragraphs. Two adjacent margins
    will "collapse" to the larger of the two margin widths, rather than doubling up.
  </Paragraph>

  <Paragraph>
    To eliminate extra spacing between two paragraphs, just set the paragraph margins to 0.
  </Paragraph>
</FlowDocument>
```

Procedura: Creare una tabella a livello di codice

23/10/2019 • 6 minutes to read • [Edit Online](#)

Negli esempi seguenti viene illustrato come creare un oggetto a [Table](#) livello di codice e compilarlo con il contenuto. Il contenuto della tabella è ripartito in cinque righe (rappresentate da [TableRow](#) oggetti contenuti in un [RowGroups](#) oggetto) e sei colonne ([TableColumn](#) rappresentate da oggetti). Le righe vengono usate per scopi di presentazione diversi, ad esempio una riga è destinata a contenere il titolo dell'intera tabella, una riga di intestazione a descrivere le colonne di dati nella tabella e una riga di più di pagina a fornire informazioni di riepilogo. Si noti che i concetti di righe di "titolo", "intestazione" e "più di pagina" non sono inerenti alla tabella, ma fanno semplicemente riferimento a righe con caratteristiche diverse. Le celle della tabella contengono il contenuto effettivo, che può essere costituito da testo, immagini o quasi tutti interfaccia utente gli altri elementi.

Esempio

Viene innanzitutto creato [FlowDocument](#) un oggetto per ospitare l' [Table](#)oggetto e viene [FlowDocument](#)creato [Table](#) un nuovo oggetto che viene aggiunto al contenuto di.

```
// Create the parent FlowDocument...
flowDoc = new FlowDocument();

// Create the Table...
table1 = new Table();
// ...and add it to the FlowDocument Blocks collection.
flowDoc.Blocks.Add(table1);

// Set some global formatting properties for the table.
table1.CellSpacing = 10;
table1.Background = Brushes.White;
```

```
' Create the parent FlowDocument...
flowDoc = New FlowDocument()

' Create the Table...
table1 = New Table()

' ...and add it to the FlowDocument Blocks collection.
flowDoc.Blocks.Add(table1)

' Set some global formatting properties for the table.
table1.CellSpacing = 10
table1.Background = Brushes.White
```

Esempio

Successivamente, vengono [TableColumn](#) creati sei oggetti e aggiunti alla [Columns](#) raccolta della tabella, con una formattazione applicata.

NOTE

Si noti che la [Columns](#) raccolta della tabella usa l'indicizzazione standard in base zero.

```

// Create 6 columns and add them to the table's Columns collection.
int numberOfRowsColumns = 6;
for (int x = 0; x < numberOfRowsColumns; x++)
{
    table1.Columns.Add(new TableColumn());

    // Set alternating background colors for the middle columns.
    if(x%2 == 0)
        table1.Columns[x].Background = Brushes.Beige;
    else
        table1.Columns[x].Background = Brushes.LightSteelBlue;
}

```

```

' Create 6 columns and add them to the table's Columns collection.
Dim numberOfRowsColumns = 6
Dim x
For x = 0 To numberOfRowsColumns
    table1.Columns.Add(new TableColumn())

    ' Set alternating background colors for the middle columns.
    If x Mod 2 = 0 Then
        table1.Columns(x).Background = Brushes.Beige
    Else
        table1.Columns(x).Background = Brushes.LightSteelBlue
    End If
Next x

```

Esempio

Viene quindi creata una riga del titolo che viene aggiunta alla tabella con l'applicazione di alcuni elementi di formattazione. La riga del titolo può contenere una sola cella che si estende su tutte e sei le colonne della tabella.

```

// Create and add an empty TableRowGroup to hold the table's Rows.
table1.RowGroups.Add(new TableRowGroup());

// Add the first (title) row.
table1.RowGroups[0].Rows.Add(new TableRow());

// Alias the current working row for easy reference.
TableRow currentRow = table1.RowGroups[0].Rows[0];

// Global formatting for the title row.
currentRow.Background = Brushes.Silver;
currentRow.FontSize = 40;
currentRow.FontWeight = System.Windows.FontWeights.Bold;

// Add the header row with content,
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("2004 Sales Project"))));
// and set the row to span all 6 columns.
currentRow.Cells[0].ColumnSpan = 6;

```

```

' Create and add an empty TableRowGroup to hold the table's Rows.
table1.RowGroups.Add(new TableRowGroup())

' Add the first (title) row.
table1.RowGroups(0).Rows.Add(new TableRow())

' Alias the current working row for easy reference.
Dim currentRow As New TableRow()
currentRow = table1.RowGroups(0).Rows(0)

' Global formatting for the title row.
currentRow.Background = Brushes.Silver
currentRow.FontSize = 40
currentRow.FontWeight = System.Windows.FontWeights.Bold

' Add the header row with content,
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("2004 Sales Project"))))

' and set the row to span all 6 columns.
currentRow.Cells(0).ColumnSpan = 6

```

Esempio

Successivamente, viene creata e aggiunta alla tabella una riga di intestazione, per la quale vengono create celle in cui viene inserito contenuto.

```

// Add the second (header) row.
table1.RowGroups[0].Rows.Add(new TableRow());
currentRow = table1.RowGroups[0].Rows[1];

// Global formatting for the header row.
currentRow.FontSize = 18;
currentRow.FontWeight = FontWeights.Bold;

// Add cells with content to the second row.
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Product"))));
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Quarter 1"))));
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Quarter 2"))));
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Quarter 3"))));
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Quarter 4"))));
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("TOTAL"))));

```

```

' Add the second (header) row.
table1.RowGroups(0).Rows.Add(new TableRow())
currentRow = table1.RowGroups(0).Rows(1)

' Global formatting for the header row.
currentRow.FontSize = 18
currentRow.FontWeight = FontWeights.Bold

' Add cells with content to the second row.
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Product"))))
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Quarter 1"))))
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Quarter 2"))))
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Quarter 3"))))
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Quarter 4"))))
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("TOTAL"))))

```

Esempio

Successivamente, viene creata e aggiunta alla tabella una riga per i dati, per la quale vengono create celle in cui

viene inserito contenuto. La compilazione di questa riga è simile alla compilazione della riga di intestazione, con l'applicazione di una formattazione leggermente diversa.

```
// Add the third row.  
table1.RowGroups[0].Rows.Add(new TableRow());  
currentRow = table1.RowGroups[0].Rows[2];  
  
// Global formatting for the row.  
currentRow.FontSize = 12;  
currentRow.FontWeight = FontWeights.Normal;  
  
// Add cells with content to the third row.  
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Widgets"))));  
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("$50,000"))));  
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("$55,000"))));  
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("$60,000"))));  
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("$65,000"))));  
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("$230,000"))));  
  
// Bold the first cell.  
currentRow.Cells[0].FontWeight = FontWeights.Bold;
```

```
' Add the third row.  
table1.RowGroups(0).Rows.Add(new TableRow())  
currentRow = table1.RowGroups(0).Rows(2)  
  
' Global formatting for the row.  
currentRow.FontSize = 12  
currentRow.FontWeight = FontWeights.Normal  
  
' Add cells with content to the third row.  
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Widgets"))))  
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("$50,000"))))  
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("$55,000"))))  
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("$60,000"))))  
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("$65,000"))))  
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("$230,000"))))  
  
' Bold the first cell.  
currentRow.Cells(0).FontWeight = FontWeights.Bold
```

Esempio

Infine, viene creata, aggiunta e formattata una riga di più di pagina. Analogamente alla riga del titolo, il più di pagina contiene una sola cella che si estende su tutte e sei le colonne della tabella.

```
table1.RowGroups[0].Rows.Add(new TableRow());  
currentRow = table1.RowGroups[0].Rows[3];  
  
// Global formatting for the footer row.  
currentRow.Background = Brushes.LightGray;  
currentRow.FontSize = 18;  
currentRow.FontWeight = System.Windows.FontWeights.Normal;  
  
// Add the header row with content,  
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Projected 2004 Revenue: $810,000"))));  
// and set the row to span all 6 columns.  
currentRow.Cells[0].ColumnSpan = 6;
```

```
table1.RowGroups(0).Rows.Add(new TableRow())
currentRow = table1.RowGroups(0).Rows(3)

' Global formatting for the footer row.
currentRow.Background = Brushes.LightGray
currentRow.FontSize = 18
currentRow.FontWeight = System.Windows.FontWeights.Normal

' Add the header row with content,
currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Projected 2004 Revenue: $810,000"))))
' and set the row to span all 6 columns.
currentRow.Cells(0).ColumnSpan = 6
```

Vedere anche

- [Cenni preliminari sull'elemento Table](#)

Procedura: Modificare l'oggetto FlowDirection del contenuto a livello di codice

23/10/2019 • 2 minutes to read • [Edit Online](#)

In questo esempio viene illustrato come modificare a livello di codice le [FlowDirection](#) proprietà di un [FlowDocumentReader](#).

Esempio

Due [Button](#) vengono creati gli elementi, ognuno dei quali rappresenta uno dei valori possibili di [FlowDirection](#). Quando viene selezionato un pulsante, il valore della proprietà associata viene applicato al contenuto di un [FlowDocumentReader](#) denominato `tf1`. Il valore della proprietà viene scritto anche a un [TextBlock](#) denominato `txt1`.

```

<StackPanel DockPanel.Dock="Top" Orientation="Horizontal" Margin="0,0,0,10">
    <Button Click="LR">LeftToRight</Button>
    <Button Click="RL">RightToLeft</Button>
</StackPanel>

<TextBlock Name="txt1" DockPanel.Dock="Bottom" Margin="0,50,0,0"/>

<FlowDocumentReader>
    <FlowDocument FontFamily="Arial" Name="tf1">
        <Paragraph>
            Lorem ipsum dolor sit amet, consectetuer adipiscing elit,
            sed diam nonummy nibh euismod tincidunt ut laboreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laboreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laboreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure.
        </Paragraph>
        <Paragraph>
            Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laboreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laboreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laboreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure.
        </Paragraph>
    </FlowDocument>
</FlowDocumentReader>

```

Esempio

Gli eventi associati ai clic sui pulsanti definiti in precedenza vengono gestiti in un C# file code-behind.

```

private void LR(object sender, RoutedEventArgs e)
{
    tf1.FlowDirection = FlowDirection.LeftToRight;
    txt1.Text = "FlowDirection is now " + tf1.FlowDirection;
}
private void RL(object sender, RoutedEventArgs e)
{
    tf1.FlowDirection = FlowDirection.RightToLeft;
    txt1.Text = "FlowDirection is now " + tf1.FlowDirection;
}

```

```
Private Sub LR(ByVal sender As Object, ByVal e As RoutedEventArgs)
    tf1.FlowDirection = FlowDirection.LeftToRight
    txt1.Text = "FlowDirection is now " & tf1.FlowDirection
End Sub
Private Sub RL(ByVal sender As Object, ByVal e As RoutedEventArgs)
    tf1.FlowDirection = FlowDirection.RightToLeft
    txt1.Text = "FlowDirection is now " & tf1.FlowDirection
End Sub
```

Procedura: Modificare la proprietà TextWrapping a livello di codice

23/10/2019 • 2 minutes to read • [Edit Online](#)

Esempio

Esempio di codice seguente viene illustrato come modificare il valore della [TextWrapping](#) proprietà a livello di codice.

Tre [Button](#) elementi vengono posizionati all'interno di un [StackPanel](#) elemento XAML. Ciascuna [Click](#) evento per un [Button](#) corrisponde a un gestore eventi nel codice. I gestori eventi usano lo stesso nome di [TextWrapping](#) valore verranno applicate a `txt2` quando fa clic sul pulsante. Inoltre, il testo nel `txt1` (una [TextBlock](#) non visualizzati nel XAML) viene aggiornato per riflettere la modifica della proprietà.

```
<StackPanel Orientation="Horizontal" Margin="0,0,0,20">
    <Button Name="btn1" Background="Silver" Width="100" Click="Wrap">Wrap</Button>
    <Button Name="btn2" Background="Silver" Width="100" Click="NoWrap">NoWrap</Button>
    <Button Name="btn4" Background="Silver" Width="100" Click="WrapWithOverflow">WrapWithOverflow</Button>
</StackPanel>

<TextBlock Name="txt2" TextWrapping="Wrap" Margin="0,0,0,20" Foreground="Black">
    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Lorem ipsum dolor sit amet,
    consectetur adipiscing elit. Lorem ipsum dolor sit aet, consectetur adipiscing elit.
    Lorem ipsum dolor sit amet, consectetur adipiscing elit.
</TextBlock>
```

```
private void Wrap(object sender, RoutedEventArgs e)
{
    txt2.TextWrapping = System.Windows.TextWrapping.Wrap;
    txt1.Text = "The TextWrap property is currently set to Wrap.";
}
private void NoWrap(object sender, RoutedEventArgs e)
{
    txt2.TextWrapping = System.Windows.TextWrapping.NoWrap;
    txt1.Text = "The TextWrap property is currently set to NoWrap.";
}
private void WrapWithOverflow(object sender, RoutedEventArgs e)
{
    txt2.TextWrapping = System.Windows.TextWrapping.WrapWithOverflow;
    txt1.Text = "The TextWrap property is currently set to WrapWithOverflow.";
}
```

```
Private Sub Wrap(ByVal sender As Object, ByVal e As System.Windows.RoutedEventArgs)
    txt2.TextWrapping = System.Windows.TextWrapping.Wrap
    txt1.Text = "The TextWrap property is currently set to Wrap."
End Sub

Private Sub NoWrap(ByVal sender As Object, ByVal e As System.Windows.RoutedEventArgs)
    txt2.TextWrapping = System.Windows.TextWrapping.NoWrap
    txt1.Text = "The TextWrap property is currently set to NoWrap."
End Sub

Private Sub WrapWithOverflow(ByVal sender As Object, ByVal e As System.Windows.RoutedEventArgs)
    txt2.TextWrapping = System.Windows.TextWrapping.WrapWithOverflow
    txt1.Text = "The TextWrap property is currently set to WrapWithOverflow."
End Sub
```

Vedere anche

- [TextWrapping](#)
- [TextWrapping](#)

Procedura: Definire una tabella tramite XAML

23/10/2019 • 2 minutes to read • [Edit Online](#)

Nell'esempio seguente viene illustrato come definire un **Table** usando Extensible Application Markup Language (XAML). La tabella di esempio contiene quattro colonne (rappresentato da **TableColumn** elementi) e più righe (rappresentato da **TableRow** elementi) che contengono dati anche a titolo, intestazione e piè di pagina informazioni. Le righe devono essere contenute in un **TableRowGroup** elemento. Ogni riga nella tabella è costituita da uno o più celle (rappresentato da **TableCell** elementi). Contenuto di una cella della tabella deve essere contenuto in un **Block** elemento; in questo caso **Paragraph** gli elementi vengono usati. La tabella ospita anche un collegamento ipertestuale (rappresentato dal **Hyperlink** elemento) nella riga del piè di pagina.

Esempio

```
<FlowDocumentReader>
<FlowDocument>

    <Table CellSpacing="5">

        <Table.Columns>
            < TableColumn />
            < TableColumn />
            < TableColumn />
            < TableColumn />
        </Table.Columns>

        < TableRowGroup >

            <!-- Title row for the table. -->
            < TableRow Background="SkyBlue" >
                < TableCell ColumnSpan="4" TextAlignment="Center" >
                    < Paragraph FontSize="24pt" FontWeight="Bold" >Planetary Information</ Paragraph >
                </ TableCell >
            </ TableRow >

            <!-- Header row for the table. -->
            < TableRow Background="LightGoldenrodYellow" >
                < TableCell >< Paragraph FontSize="14pt" FontWeight="Bold" >Planet</ Paragraph ></ TableCell >
                < TableCell >< Paragraph FontSize="14pt" FontWeight="Bold" >Mean Distance from Sun</ Paragraph >
                < TableCell >< Paragraph FontSize="14pt" FontWeight="Bold" >Mean Diameter</ Paragraph ></ TableCell >
                < TableCell >< Paragraph FontSize="14pt" FontWeight="Bold" >Approximate Mass</ Paragraph ></ TableCell >
            </ TableRow >

            <!-- Sub-title row for the inner planets. -->
            < TableRow >
                < TableCell ColumnSpan="4" >< Paragraph FontSize="14pt" FontWeight="Bold" >The Inner Planets</ Paragraph >
            </ TableCell >
            </ TableRow >

            <!-- Four data rows for the inner planets. -->
            < TableRow >
                < TableCell >< Paragraph >Mercury</ Paragraph ></ TableCell >
                < TableCell >< Paragraph >57,910,000 km</ Paragraph ></ TableCell >
                < TableCell >< Paragraph >4,880 km</ Paragraph ></ TableCell >
                < TableCell >< Paragraph >3.30e23 kg</ Paragraph ></ TableCell >
            </ TableRow >
            < TableRow Background="lightgray" >
                < TableCell >< Paragraph >Venus</ Paragraph ></ TableCell >
                < TableCell >< Paragraph >108,200,000 km</ Paragraph ></ TableCell >
```

```

<TableCell><Paragraph>12,103.6 km</Paragraph></TableCell>
<TableCell><Paragraph>4.869e24 kg</Paragraph></TableCell>
</TableRow>
<TableRow>
    <TableCell><Paragraph>Earth</Paragraph></TableCell>
    <TableCell><Paragraph>149,600,000 km</Paragraph></TableCell>
    <TableCell><Paragraph>12,756.3 km</Paragraph></TableCell>
    <TableCell><Paragraph>5.972e24 kg</Paragraph></TableCell>
</TableRow>
<TableRow Background="lightgray">
    <TableCell><Paragraph>Mars</Paragraph></TableCell>
    <TableCell><Paragraph>227,940,000 km</Paragraph></TableCell>
    <TableCell><Paragraph>6,794 km</Paragraph></TableCell>
    <TableCell><Paragraph>6.4219e23 kg</Paragraph></TableCell>
</TableRow>

<!-- Sub-title row for the outer planets. --&gt;
&lt;TableRow&gt;
    &lt;TableCell ColumnSpan="4"&gt;&lt;Paragraph FontSize="14pt" FontWeight="Bold"&gt;The Major Outer Planets&lt;/Paragraph&gt;&lt;/TableCell&gt;
&lt;/TableRow&gt;

<!-- Four data rows for the major outer planets. --&gt;
&lt;TableRow&gt;
    &lt;TableCell&gt;&lt;Paragraph&gt;Jupiter&lt;/Paragraph&gt;&lt;/TableCell&gt;
    &lt;TableCell&gt;&lt;Paragraph&gt;778,330,000 km&lt;/Paragraph&gt;&lt;/TableCell&gt;
    &lt;TableCell&gt;&lt;Paragraph&gt;142,984 km&lt;/Paragraph&gt;&lt;/TableCell&gt;
    &lt;TableCell&gt;&lt;Paragraph&gt;1.900e27 kg&lt;/Paragraph&gt;&lt;/TableCell&gt;
&lt;/TableRow&gt;
&lt;TableRow Background="lightgray"&gt;
    &lt;TableCell&gt;&lt;Paragraph&gt;Saturn&lt;/Paragraph&gt;&lt;/TableCell&gt;
    &lt;TableCell&gt;&lt;Paragraph&gt;1,429,400,000 km&lt;/Paragraph&gt;&lt;/TableCell&gt;
    &lt;TableCell&gt;&lt;Paragraph&gt;120,536 km&lt;/Paragraph&gt;&lt;/TableCell&gt;
    &lt;TableCell&gt;&lt;Paragraph&gt;5.68e26 kg&lt;/Paragraph&gt;&lt;/TableCell&gt;
&lt;/TableRow&gt;
&lt;TableRow&gt;
    &lt;TableCell&gt;&lt;Paragraph&gt;Uranus&lt;/Paragraph&gt;&lt;/TableCell&gt;
    &lt;TableCell&gt;&lt;Paragraph&gt;2,870,990,000 km&lt;/Paragraph&gt;&lt;/TableCell&gt;
    &lt;TableCell&gt;&lt;Paragraph&gt;51,118 km&lt;/Paragraph&gt;&lt;/TableCell&gt;
    &lt;TableCell&gt;&lt;Paragraph&gt;8.683e25 kg&lt;/Paragraph&gt;&lt;/TableCell&gt;
&lt;/TableRow&gt;
&lt;TableRow Background="lightgray"&gt;
    &lt;TableCell&gt;&lt;Paragraph&gt;Neptune&lt;/Paragraph&gt;&lt;/TableCell&gt;
    &lt;TableCell&gt;&lt;Paragraph&gt;4,504,000,000 km&lt;/Paragraph&gt;&lt;/TableCell&gt;
    &lt;TableCell&gt;&lt;Paragraph&gt;49,532 km&lt;/Paragraph&gt;&lt;/TableCell&gt;
    &lt;TableCell&gt;&lt;Paragraph&gt;1.0247e26 kg&lt;/Paragraph&gt;&lt;/TableCell&gt;
&lt;/TableRow&gt;

<!-- Footer row for the table. --&gt;
&lt;TableRow&gt;
    &lt;TableCell ColumnSpan="4"&gt;&lt;Paragraph FontSize="10pt" FontStyle="Italic"&gt;
        Information from the
        &lt;Hyperlink NavigateUri="http://encarta.msn.com/encnet/refpages/artcenter.aspx"&gt;Encarta&lt;/Hyperlink&gt;
        web site.
    &lt;/Paragraph&gt;&lt;/TableCell&gt;
&lt;/TableRow&gt;

&lt;/TableRowGroup&gt;
&lt;/Table&gt;
&lt;/FlowDocument&gt;
&lt;/FlowDocumentReader&gt;
</pre>

```

La figura seguente mostra come la tabella definita in questo esempio viene eseguito il rendering:

Planetary Information			
Planet	Mean Distance from Sun	Mean Diameter	Approximate Mass
The Inner Planets			
Mercury	57,910,000 km	4,880 km	3.30e23 kg
Venus	108,200,000 km	12,103.6 km	4.869e24 kg
Earth	149,600,000 km	12,756.3 km	5.972e24 kg
Mars	227,940,000 km	6,794 km	6.4219e23 kg
The Major Outer Planets			
Jupiter	778,330,000 km	142,984 km	1.900e27 kg
Saturn	1,429,400,000 km	120,536 km	5.68e26 kg
Uranus	2,870,990,000 km	51,118 km	8.683e25 kg

Procedura: Modificare la tipografia del testo

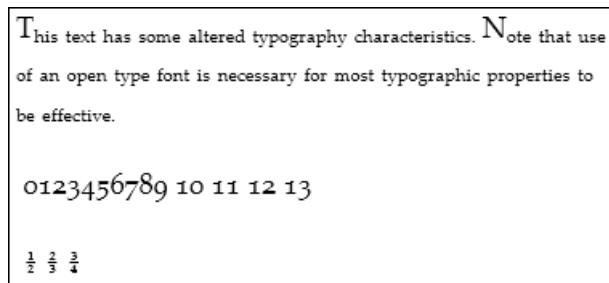
23/10/2019 • 2 minutes to read • [Edit Online](#)

Nell'esempio seguente viene illustrato come impostare il **Typography** dell'attributo, usando **Paragraph** come elemento di esempio.

Esempio

```
<Paragraph  
    TextAlignment="Left"  
    FontSize="18"  
    FontFamily="Palatino Linotype"  
    Typography.NumeralsStyle="OldStyle"  
    Typography.Fraction="Stacked"  
    Typography.Variants="Inferior"  
>  
<Run>  
    This text has some altered typography characteristics. Note  
    that use of an open type font is necessary for most typographic  
    properties to be effective.  
</Run>  
<LineBreak/><LineBreak/>  
<Run>  
    0123456789 10 11 12 13  
</Run>  
<LineBreak/><LineBreak/>  
<Run>  
    1/2 2/3 3/4  
</Run>  
</Paragraph>
```

La figura seguente illustra il rendering di questo esempio.

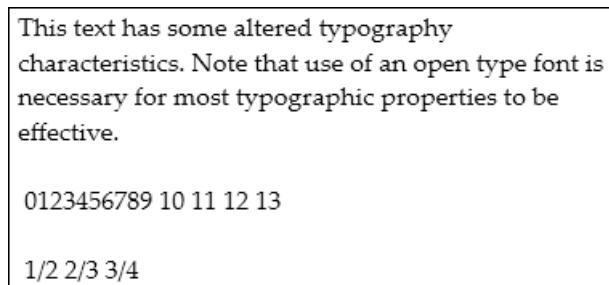


This text has some altered typography characteristics. Note that use of an open type font is necessary for most typographic properties to be effective.

0123456789 10 11 12 13

1/2 2/3 3/4

La figura seguente mostra invece il rendering dello stesso esempio con le proprietà tipografiche predefinite.



This text has some altered typography characteristics. Note that use of an open type font is necessary for most typographic properties to be effective.

0123456789 10 11 12 13

1/2 2/3 3/4

Esempio

Nell'esempio seguente viene illustrato come impostare il **Typography** proprietà a livello di codice.

```

Paragraph par = new Paragraph();

Run runText = new Run(
    "This text has some altered typography characteristics. Note" +
    "that use of an open type font is necessary for most typographic" +
    "properties to be effective.");
Run runNumerals = new Run("0123456789 10 11 12 13");
Run runFractions = new Run("1/2 2/3 3/4");

par.Inlines.Add(runText);
par.Inlines.Add(new LineBreak());
par.Inlines.Add(new LineBreak());
par.Inlines.Add(runNumerals);
par.Inlines.Add(new LineBreak());
par.Inlines.Add(new LineBreak());
par.Inlines.Add(runFractions);

par.TextAlignment = TextAlignment.Left;
par.FontSize = 18;
par.FontFamily = new FontFamily("Palatino Linotype");

par.Typography.NumeralStyle = FontNumeralStyle.OldStyle;
par.Typography.Fraction = FontFraction.Stacked;
par.Typography.Variants = FontVariants.Inferior;

```

```

Dim par As New Paragraph()

Dim runText As New Run("This text has some altered typography characteristics. Note" & "that use of an open
type font is necessary for most typographic" & "properties to be effective.")
Dim runNumerals As New Run("0123456789 10 11 12 13")
Dim runFractions As New Run("1/2 2/3 3/4")

par.Inlines.Add(runText)
par.Inlines.Add(New LineBreak())
par.Inlines.Add(New LineBreak())
par.Inlines.Add(runNumerals)
par.Inlines.Add(New LineBreak())
par.Inlines.Add(New LineBreak())
par.Inlines.Add(runFractions)

par.TextAlignment = TextAlignment.Left
par.FontSize = 18
par.FontFamily = New FontFamily("Palatino Linotype")

par.Typography.NumeralStyle = FontNumeralStyle.OldStyle
par.Typography.Fraction = FontFraction.Stacked
par.Typography.Variants = FontVariants.Inferior

```

Vedere anche

- [Cenni preliminari sui documenti dinamici](#)

Procedura: Abilitare l'enumerazione TextTrimming

23/10/2019 • 2 minutes to read • [Edit Online](#)

Questo esempio viene illustrato l'utilizzo e gli effetti dei valori disponibili nel [TextTrimming](#) enumerazione.

Esempio

L'esempio seguente definisce una [TextBlock](#) elemento con la [TextTrimming](#) set di attributi.

```
<TextBlock  
    Name="myTextBlock"  
    Margin="20" Background="LightGoldenrodYellow"  
    TextTrimming="WordEllipsis" TextWrapping="NoWrap"  
    FontSize="14"  
>  
    One<LineBreak/>  
    two two<LineBreak/>  
    Three Three<LineBreak/>  
    four four four<LineBreak/>  
    Five Five Five Five<LineBreak/>  
    six six six six six<LineBreak/>  
    Seven Seven Seven Seven Seven Seven  
</TextBlock>
```

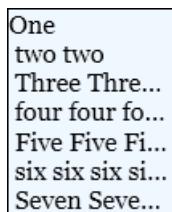
L'impostazione corrispondente [TextTrimming](#) proprietà nel codice come illustrato di seguito.

```
myTextBlock.TextTrimming = TextTrimming.CharacterEllipsis;
```

```
myTextBlock.TextTrimming = TextTrimming.CharacterEllipsis
```

Esistono attualmente tre opzioni per il taglio del testo: **CharacterEllipsis**, **WordEllipsis**, e **None**.

Quando [TextTrimming](#) è impostata su **CharacterEllipsis**, il testo viene tagliato e con i puntini di sospensione in corrispondenza del carattere più vicino al bordo del taglio. Questa impostazione consente di tagliare il testo in modo da adattarlo meglio al limite di taglio, ma potrebbe comportare la troncatura delle parole. La figura seguente mostra l'effetto di questa impostazione su un [TextBlock](#) simile a quello definito in precedenza.



One
two two
Three Thre...
four four fo...
Five Five Fi...
six six six si...
Seven Seve...

A screenshot of a Windows application window containing a single-line [TextBlock](#). The text "One two two Three Thre... four four fo... Five Five Fi... six six six si... Seven Seve..." is displayed. The text is truncated at the end of the fourth word ("four") and ends with three ellipsis characters (...). The text is rendered in a black sans-serif font on a white background.

Quando [TextTrimming](#) è impostata su **WordEllipsis**, il testo viene tagliato e con i puntini di sospensione alla fine della prima parola completa più vicina al bordo del taglio. Questa impostazione non visualizzerà le parole, ma non è possibile tagliare il testo come da vicino al bordo del taglio come le **CharacterEllipsis** impostazione. La figura seguente mostra l'effetto di questa impostazione per il [TextBlock](#) definiti in precedenza.

```
One  
two two  
Three...  
four four...  
Five Five...  
six six six...  
Seven...
```

Quando **TextTrimming** è impostata su **None**, non viene eseguita alcuna operazione di taglio del testo. In questo caso viene eseguita una semplice operazione di ritaglio in corrispondenza del limite del contenitore di testo padre. La figura seguente mostra l'effetto di questa impostazione su un **TextBlock** simile a quello definito in precedenza.

```
One  
two two  
Three Three T  
four four four  
Five Five Five  
six six six six s  
Seven Seven S
```

Procedura: Inserire un elemento in un testo a livello di codice

23/10/2019 • 2 minutes to read • [Edit Online](#)

Nell'esempio seguente viene illustrato come usare due [TextPointer](#) gli oggetti per specificare un intervallo all'interno del testo per applicare un [Span](#) elemento.

Esempio

```
using System;
using System.Windows;
using System.Windows.Media;
using System.Windows.Controls;
using System.Windows.Documents;

namespace SDKSample
{
    public partial class InsertInlineIntoTextExample : Page
    {
        public InsertInlineIntoTextExample()
        {

            // Create a paragraph with a short sentence
            Paragraph myParagraph = new Paragraph(new Run("Neptune has 72 times Earth's volume..."));

            // Create two TextPointers that will specify the text range the Span will cover
            TextPointer myTextPointer1 = myParagraph.ContentStart.GetPositionAtOffset(10);
            TextPointer myTextPointer2 = myParagraph.ContentEnd.GetPositionAtOffset(-5);

            // Create a Span that covers the range between the two TextPointers.
            Span mySpan = new Span(myTextPointer1, myTextPointer2);
            mySpan.Background = Brushes.Red;

            // Create a FlowDocument with the paragraph as its initial content.
            FlowDocument myFlowDocument = new FlowDocument(myParagraph);

            this.Content = myFlowDocument;
        }
    }
}
```

```

Imports System.Windows
Imports System.Windows.Media
Imports System.Windows.Controls
Imports System.Windows.Documents

Namespace SDKSample
    Partial Public Class InsertInlineIntoTextExample
        Inherits Page
        Public Sub New()

            ' Create a paragraph with a short sentence
            Dim myParagraph As New Paragraph(New Run("Neptune has 72 times Earth's volume..."))

            ' Create two TextPointers that will specify the text range the Span will cover
            Dim myTextPointer1 As TextPointer = myParagraph.ContentStart.GetPositionAtOffset(10)
            Dim myTextPointer2 As TextPointer = myParagraph.ContentEnd.GetPositionAtOffset(-5)

            ' Create a Span that covers the range between the two TextPointers.
            Dim mySpan As New Span(myTextPointer1, myTextPointer2)
            mySpan.Background = Brushes.Red

            ' Create a FlowDocument with the paragraph as its initial content.
            Dim myFlowDocument As New FlowDocument(myParagraph)

            Me.Content = myFlowDocument

        End Sub
    End Class
End Namespace

```

Nella figura riportata di seguito viene illustrato l'esempio in questione.

Neptune has 72 times Earth's volume...

Vedere anche

- [Cenni preliminari sui documenti dinamici](#)

Procedura: Modificare elementi di contenuto di flusso tramite la proprietà Blocks

23/10/2019 • 2 minutes to read • [Edit Online](#)

Questi esempi illustrano alcune delle operazioni più comuni che possono essere eseguite su elementi di contenuto dinamico tramite il **blocchi** proprietà. Questa proprietà viene utilizzata per aggiungere e rimuovere elementi da [BlockCollection](#). Flusso di contenuto gli elementi che presentano una **blocchi** proprietà includono:

- [Figure](#)
- [Floater](#)
- [ListItem](#)
- [Section](#)
- [TableCell](#)

Usare questi esempi si trovano [Section](#) come flusso di elemento di contenuto, ma queste tecniche sono applicabili a tutti gli elementi che ospitano un insieme di elementi di contenuto di flusso.

Esempio

L'esempio seguente crea una nuova [Section](#) e quindi Usa le **Add** metodo per aggiungere un nuovo paragrafo per il **sezione** contenuto.

```
Section secx = new Section();
secx.Blocks.Add(new Paragraph(new Run("A bit of text content...")));
```

```
Dim secx As New Section()
secx.Blocks.Add(New Paragraph(New Run("A bit of text content...")))
```

Esempio

L'esempio seguente crea una nuova [Paragraph](#) elemento e lo inserisce all'inizio del [Section](#).

```
Paragraph parx = new Paragraph(new Run("Text to insert..."));
secx.Blocks.InsertBefore(secx.Blocks.FirstBlock, parx);
```

```
Dim parx As New Paragraph(New Run("Text to insert..."))
secx.Blocks.InsertBefore(secx.Blocks.FirstBlock, parx)
```

Esempio

Nell'esempio seguente ottiene il numero di livello superiore [Block](#) gli elementi contenuti nel [Section](#).

```
int countTopLevelBlocks = secx.Blocks.Count;
```

```
Dim countTopLevelBlocks As Integer = secx.Blocks.Count
```

Esempio

L'esempio seguente elimina l'ultima [Block](#) elemento il [Section](#).

```
secx.Blocks.Remove(secx.Blocks.LastBlock);
```

```
secx.Blocks.Remove(secx.Blocks.LastBlock)
```

Esempio

L'esempio seguente Cancella tutto il contenuto ([Block](#) elementi) dal [Section](#).

```
secx.Blocks.Clear();
```

```
secx.Blocks.Clear()
```

Vedere anche

- [BlockCollection](#)
- [InlineCollection](#)
- [ListItemCollection](#)
- [Cenni preliminari sui documenti dinamici](#)
- [Modificare i gruppi di righe di una tabella tramite la proprietà RowGroups](#)
- [Modificare le colonne di una tabella tramite la proprietà Columns](#)
- [Modificare i gruppi di righe di una tabella tramite la proprietà RowGroups](#)

Procedura: Modificare elementi di contenuto di flusso tramite la proprietà Inlines

23/10/2019 • 2 minutes to read • [Edit Online](#)

Questi esempi illustrano alcune delle operazioni più comuni che possono essere eseguite su elementi di contenuto dinamico inline (e i contenitori di tali elementi, ad esempio `TextBlock`) tramite il **Inlines** proprietà. Questa proprietà viene utilizzata per aggiungere e rimuovere elementi da `InlineCollection`. Flusso di contenuto gli elementi che presentano un' **Inlines** proprietà includono:

- [Bold](#)
- [Hyperlink](#)
- [Italic](#)
- [Paragraph](#)
- [Span](#)
- [Underline](#)

Usare questi esempi si trovano [Span](#) come il flusso di elemento di contenuto, ma queste tecniche sono applicabili a tutti gli elementi o i controlli che ospitano un [InlineCollection](#) raccolta.

Esempio

L'esempio seguente crea un nuovo [Span](#) oggetto e quindi usare il **Add** metodo per aggiungere testo due sequenze come elementi figlio di contenuto il [Span](#).

```
Span spanx = new Span();
spanx.Inlines.Add(new Run("A bit of text content..."));
spanx.Inlines.Add(new Run("A bit more text content..."));
```

```
Dim spanx As New Span()
spanx.Inlines.Add(New Run("A bit of text content..."))
spanx.Inlines.Add(New Run("A bit more text content..."))
```

Esempio

L'esempio seguente crea una nuova [Run](#) elemento e lo inserisce all'inizio del [Span](#).

```
Run runx = new Run("Text to insert...");
spanx.Inlines.InsertBefore(spanx.Inlines.FirstInline, runx);
```

```
Dim runx As New Run("Text to insert...")
spanx.Inlines.InsertBefore(spanx.Inlines.FirstInline, runx)
```

Esempio

Nell'esempio seguente ottiene il numero di livello superiore [Inline](#) gli elementi contenuti nel [Span](#).

```
int countTopLevelInlines = spanx.Inlines.Count;
```

```
Dim countTopLevelInlines As Integer = spanx.Inlines.Count
```

Esempio

L'esempio seguente elimina l'ultima [Inline](#) elemento il [Span](#).

```
spanx.Inlines.Remove(spanx.Inlines.LastInline);
```

```
spanx.Inlines.Remove(spanx.Inlines.LastInline)
```

Esempio

L'esempio seguente Cancella tutto il contenuto ([Inline](#) elementi) dal [Span](#).

```
spanx.Inlines.Clear();
```

```
spanx.Inlines.Clear()
```

Vedere anche

- [BlockCollection](#)
- [InlineCollection](#)
- [ListItemCollection](#)
- [Cenni preliminari sui documenti dinamici](#)
- [Modificare un oggetto FlowDocument tramite la proprietà Blocks](#)
- [Modificare le colonne di una tabella tramite la proprietà Columns](#)
- [Modificare i gruppi di righe di una tabella tramite la proprietà RowGroups](#)

Procedura: Modificare un oggetto FlowDocument tramite la proprietà Blocks

23/10/2019 • 2 minutes to read • [Edit Online](#)

Questi esempi illustrano alcune delle operazioni più comuni che possono essere eseguite in un [FlowDocument](#) attraverso il [Blocks](#) proprietà.

Esempio

L'esempio seguente crea una nuova [FlowDocument](#) e quindi aggiunge un nuovo [Paragraph](#) elemento per il [FlowDocument](#).

```
FlowDocument flowDoc = new FlowDocument(new Paragraph(new Run("A bit of text content...")));
flowDoc.Blocks.Add(new Paragraph(new Run("Text to append...")));
```

```
Dim flowDoc As New FlowDocument(New Paragraph(New Run("A bit of text content...")))
flowDoc.Blocks.Add(New Paragraph(New Run("Text to append...")))
```

Esempio

L'esempio seguente crea una nuova [Paragraph](#) elemento e lo inserisce all'inizio del [FlowDocument](#).

```
Paragraph p = new Paragraph(new Run("Text to insert..."));
flowDoc.Blocks.InsertBefore(flowDoc.Blocks.FirstBlock, p);
```

```
Dim p As New Paragraph(New Run("Text to insert..."))
flowDoc.Blocks.InsertBefore(flowDoc.Blocks.FirstBlock, p)
```

Esempio

Nell'esempio seguente ottiene il numero di livello superiore [Block](#) gli elementi contenuti nel [FlowDocument](#).

```
int countTopLevelBlocks = flowDoc.Blocks.Count;
```

```
Dim countTopLevelBlocks As Integer = flowDoc.Blocks.Count
```

Esempio

L'esempio seguente elimina l'ultima [Block](#) elemento il [FlowDocument](#).

```
flowDoc.Blocks.Remove(flowDoc.Blocks.LastBlock);
```

```
flowDoc.Blocks.Remove(flowDoc.Blocks.LastBlock)
```

Esempio

L'esempio seguente Cancella tutto il contenuto ([Block](#) elementi) dal [FlowDocument](#).

```
flowDoc.Blocks.Clear();
```

```
flowDoc.Blocks.Clear()
```

Vedere anche

- [Modificare i gruppi di righe di una tabella tramite la proprietà RowGroups](#)
- [Modificare le colonne di una tabella tramite la proprietà Columns](#)
- [Modificare i gruppi di righe di una tabella tramite la proprietà RowGroups](#)

Procedura: Modificare le colonne di una tabella tramite la proprietà Columns

23/10/2019 • 2 minutes to read • [Edit Online](#)

In questo esempio vengono illustrate alcune delle operazioni più comuni che è possibile eseguire sulle colonne di una tabella [Columns](#) tramite la proprietà.

Esempio

Nell'esempio seguente viene creata una nuova tabella, quindi viene [Add](#) utilizzato il metodo per aggiungere colonne alla [Columns](#) raccolta della tabella.

```
Table tbl = new Table();
int columnsToAdd = 4;
for (int x = 0; x < columnsToAdd; x++)
    tbl.Columns.Add(new TableColumn());
```

```
Dim tbl As New Table()
Dim columnsToAdd As Integer = 4
For x As Integer = 0 To columnsToAdd - 1
    tbl.Columns.Add(New TableColumn())
Next x
```

Esempio

Nell'esempio seguente viene inserito un [TableColumn](#) nuovo oggetto. La nuova colonna viene inserita in corrispondenza della posizione di indice 0, rendendola la nuova prima colonna della tabella.

NOTE

La [TableColumnCollection](#) raccolta usa l'indicizzazione in base zero standard.

```
tbl.Columns.Insert(0, new TableColumn());
```

```
tbl.Columns.Insert(0, New TableColumn())
```

Esempio

Nell'esempio seguente vengono accessibili alcune proprietà arbitrarie per le [TableColumnCollection](#) colonne della raccolta, che fanno riferimento a colonne specifiche in base all'indice.

```
tbl.Columns[0].Width = new GridLength(20);
tbl.Columns[1].Background = Brushes.AliceBlue;
tbl.Columns[2].Width = new GridLength(20);
tbl.Columns[3].Background = Brushes.AliceBlue;
```

```
tbl.Columns(0).Width = New GridLength(20)
tbl.Columns(1).Background = Brushes.AliceBlue
tbl.Columns(2).Width = New GridLength(20)
tbl.Columns(3).Background = Brushes.AliceBlue
```

Esempio

Nell'esempio seguente viene ottenuto il numero di colonne attualmente ospitate dalla tabella.

```
int columns = tbl.Columns.Count;
```

```
Dim columns As Integer = tbl.Columns.Count
```

Esempio

Nell'esempio seguente viene rimossa una colonna specifica in base al riferimento.

```
tbl.Columns.Remove(tbl.Columns[3]);
```

```
tbl.Columns.Remove(tbl.Columns(3))
```

Esempio

Nell'esempio seguente viene rimossa una colonna specifica in base all'indice.

```
tbl.Columns.RemoveAt(2);
```

```
tbl.Columns.RemoveAt(2)
```

Esempio

Nell'esempio seguente vengono rimosse tutte le colonne dalla raccolta Columns della tabella.

```
tbl.Columns.Clear();
```

```
tbl.Columns.Clear()
```

Vedere anche

- [Cenni preliminari sull'elemento Table](#)
- [Definire un oggetto Table con XAML](#)
- [Compilare oggetti Table a livello di codice](#)
- [Modificare i gruppi di righe di una tabella tramite la proprietà RowGroups](#)
- [Modificare un oggetto FlowDocument tramite la proprietà Blocks](#)
- [Modificare i gruppi di righe di una tabella tramite la proprietà RowGroups](#)

Procedura: Modificare i gruppi di righe di una tabella tramite la proprietà RowGroups

23/10/2019 • 3 minutes to read • [Edit Online](#)

Questo esempio illustra alcune delle operazioni più comuni che è possibile eseguire sui gruppi di righe di una tabella tramite la [RowGroups](#) proprietà.

Esempio

Nell'esempio seguente viene creata una nuova tabella, quindi viene [Add](#) utilizzato il metodo per aggiungere colonne alla [RowGroups](#) raccolta della tabella.

```
Table tbl = new Table();
int rowGroupsToAdd = 4;
for (int x = 0; x < rowGroupsToAdd; x++)
    tbl.RowGroups.Add(new TableRowGroup());
```

```
Dim tbl As New Table()
Dim rowGroupsToAdd As Integer = 4
For x As Integer = 0 To rowGroupsToAdd - 1
    tbl.RowGroups.Add(New TableRowGroup())
Next x
```

Esempio

Nell'esempio seguente viene inserito un [TableRowGroup](#) nuovo oggetto. La nuova colonna viene inserita in corrispondenza della posizione di indice 0, rendendola il nuovo gruppo di righe nella tabella.

NOTE

La [TableRowGroupCollection](#) raccolta usa l'indicizzazione in base zero standard.

```
tbl.RowGroups.Insert(0, new TableRowGroup());
```

```
tbl.RowGroups.Insert(0, New TableRowGroup())
```

Esempio

Nell'esempio seguente vengono aggiunte più righe a un [TableRowGroup](#) particolare (specificato dall'indice) nella tabella.

```
int rowsToAdd = 10;
for (int x = 0; x < rowsToAdd; x++)
    tbl.RowGroups[0].Rows.Add(new TableRow());
```

```

Dim rowsToAdd As Integer = 10
For x As Integer = 0 To rowsToAdd - 1
    tbl.RowGroups(0).Rows.Add(New TableRow())
Next x

```

Esempio

Nell'esempio seguente vengono accessibili alcune proprietà arbitrarie sulle righe nel primo gruppo di righe della tabella.

```

// Alias the working TableRowGroup for ease in referencing.
TableRowGroup trg = tbl.RowGroups[0];
trg.Rows[0].Background = Brushes.CornflowerBlue;
trg.Rows[1].FontSize = 24;
trg.Rows[2].ToolTip = "This row's tooltip";

```

```

' Alias the working TableRowGroup for ease in referencing.
Dim trg As TableRowGroup = tbl.RowGroups(0)
trg.Rows(0).Background = Brushes.CornflowerBlue
trg.Rows(1).FontSize = 24
trg.Rows(2).ToolTip = "This row's tooltip"

```

Esempio

Nell'esempio seguente vengono aggiunte più celle a un [TableRow](#) particolare (specificato dall'indice) nella tabella.

```

int cellsToAdd = 10;
for (int x = 0; x < cellsToAdd; x++)
    tbl.RowGroups[0].Rows[0].Cells.Add(new TableCell(new Paragraph(new Run("Cell " + (x + 1)))));

```

```

Dim cellsToAdd As Integer = 10
For x As Integer = 0 To cellsToAdd - 1
    tbl.RowGroups(0).Rows(0).Cells.Add(New TableCell(New Paragraph(New Run("Cell " & (x + 1)))))
Next x

```

Esempio

Nell'esempio seguente vengono accessibili alcune proprietà e metodi arbitrari nelle celle della prima riga del primo gruppo di righe.

```

// Alias the working for for ease in referencing.
TableRow row = tbl.RowGroups[0].Rows[0];
row.Cells[0].Background = Brushes.PapayaWhip;
row.Cells[1].FontStyle = FontStyles.Italic;
// This call clears all of the content from this cell.
row.Cells[2].Blocks.Clear();

```

```
' Alias the working for for ease in referencing.  
Dim row As TableRow = tbl.RowGroups(0).Rows(0)  
row.Cells(0).Background = Brushes.PapayaWhip  
row.Cells(1).FontStyle = FontStyle.Italic  
' This call clears all of the content from this cell.  
row.Cells(2).Blocks.Clear()
```

Esempio

Nell'esempio seguente viene restituito il numero [TableRowGroup](#) di elementi ospitati dalla tabella.

```
int rowGroups = tbl.RowGroups.Count;
```

```
Dim rowGroups As Integer = tbl.RowGroups.Count
```

Esempio

Nell'esempio seguente viene rimosso un particolare gruppo di righe in base al riferimento.

```
tbl.RowGroups.Remove(tbl.RowGroups[0]);
```

```
tbl.RowGroups.Remove(tbl.RowGroups(0))
```

Esempio

Nell'esempio seguente viene rimosso un particolare gruppo di righe in base all'indice.

```
tbl.RowGroups.RemoveAt(0);
```

```
tbl.RowGroups.RemoveAt(0)
```

Esempio

Nell'esempio seguente vengono rimossi tutti i gruppi di righe dalla raccolta di gruppi di righe della tabella.

```
tbl.RowGroups.Clear();
```

```
tbl.RowGroups.Clear()
```

Vedere anche

- [Procedura: Modificare gli elementi di contenuto del flusso tramite la proprietà Inlines](#)
- [Modificare un oggetto FlowDocument tramite la proprietà Blocks](#)
- [Modificare le colonne di una tabella tramite la proprietà Columns](#)

Procedura: Usare elementi di contenuto di flusso

23/10/2019 • 2 minutes to read • [Edit Online](#)

L'esempio seguente illustra l'uso dichiarativo di vari elementi di contenuto dinamico degli attributi associati. Gli elementi e gli attributi illustrati includono:

- Elemento **Bold**
- **BreakPageBefore** Attributo
- **FontSize** Attributo
- Elemento **Italic**
- Elemento **LineBreak**
- Elemento **List**
- Elemento **ListItem**
- Elemento **Paragraph**
- Elemento **Run**
- Elemento **Section**
- Elemento **Span**
- **Variants** attributo (apice e pedice)
- Elemento **Underline**

Esempio

```
<FlowDocument
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
>
    <Paragraph FontSize="18">Flow Format Example</Paragraph>

    <Paragraph>
        Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy
        nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi
        enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis
        nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure.
    </Paragraph>
    <Paragraph>
        Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh
        euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim
        ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl
        ut aliquip ex ea commodo consequat. Duis autem vel eum iriure.
    </Paragraph>

    <Paragraph FontSize="18">More flow elements</Paragraph>
    <Paragraph FontSize="15">Inline, font type and weight, and a List</Paragraph>

    <List>
        <ListItem><Paragraph>ListItem 1</Paragraph></ListItem>
        <ListItem><Paragraph>ListItem 2</Paragraph></ListItem>
        <ListItem><Paragraph>ListItem 3</Paragraph></ListItem>
    </List>
```

```
<List><List Item="1">List Item 4</List Item>
<List Item="2">List Item 5</List Item>
</List>

<Paragraph><Bold>Bolded</Bold></Paragraph>
<Paragraph><Underline>Underlined</Underline></Paragraph>
<Paragraph><Bold><Underline>Bolded and Underlined</Underline></Bold></Paragraph>
<Paragraph><Italic>Italic</Italic></Paragraph>

<Paragraph><Span>The Span element, no inherent rendering</Span></Paragraph>
<Paragraph><Run>The Run element, no inherent rendering</Run></Paragraph>

<Paragraph FontSize="15">Subscript, Superscript</Paragraph>

<Paragraph>
    <Run Typography.Variants="Superscript">This text is Superscripted.</Run> This text isn't.
</Paragraph>
<Paragraph>
    <Run Typography.Variants="Subscript">This text is Subscripted.</Run> This text isn't.
</Paragraph>
<Paragraph>
    If a font does not support a particular form (such as Superscript) a default font form will be
    displayed.
</Paragraph>

<Paragraph FontSize="15">Blocks, breaks, paragraph</Paragraph>

<Section><Paragraph>A block section of text</Paragraph></Section>
<Section><Paragraph>Another block section of text</Paragraph></Section>

<Paragraph><LineBreak/></Paragraph>
<Section><Paragraph>... and another section, preceded by a LineBreak</Paragraph></Section>

<Section BreakPageBefore="True"/>
<Section><Paragraph>... and another section, preceded by a PageBreak</Paragraph></Section>

<Paragraph>Finally, a paragraph. Note the break between this paragraph ...</Paragraph>
<Paragraph TextIndent="25">... and this paragraph, and also the left indentation.</Paragraph>

<Paragraph><LineBreak/></Paragraph>

</FlowDocument>
```

Procedura: Usare gli attributi di separazione delle colonne di un oggetto FlowDocument

23/10/2019 • 2 minutes to read • [Edit Online](#)

In questo esempio viene illustrato come utilizzare le funzionalità di separazione delle colonne di una [FlowDocument](#).

Esempio

L'esempio seguente definisce una [FlowDocument](#) imposta la [ColumnGap](#), [ColumnRuleBrush](#), e [ColumnRuleWidth](#) attributi. Il [FlowDocument](#) contiene un paragrafo unico del contenuto di esempio.

```
<FlowDocumentReader>
  <FlowDocument
    ColumnGap="20.0"
    ColumnRuleBrush="DodgerBlue"
    ColumnRuleWidth="5.0"
    ColumnWidth="140.0"
  >
    <Paragraph Background="AntiqueWhite" TextAlignment="Left">
      This paragraph has the background set to antique white to make its
      boundaries obvious.

      The column gap is the space between columns; this FlowDocument will
      have a column gap of 20 device-independend pixels. The column rule
      is a vertical line drawn in the column gap, and is used to visually
      separate columns; this FlowDocument a Dodger-blue column rule that
      is 5 pixels wide.

      The column rule and column gap both take space between columns. In
      this case, a column gap width of 20 plus a column rule of width of 5
      results in the space between columns being 25 pixels wide, 5 pixels
      for the column rule, and 10 pixels of column gap on either side of the column rule.
    </Paragraph>
  </FlowDocument>
</FlowDocumentReader>
```

Nella figura seguente vengono illustrati gli effetti del [ColumnGap](#), [ColumnRuleBrush](#), e [ColumnRuleWidth](#) gli attributi in un rendering [FlowDocument](#).

This paragraph has the background set to antique white to make its boundaries obvious. The column gap is the space between columns; this FlowDocument will have a column gap of 20 device-independend pixels. The column rule is a vertical line drawn

in the column gap, and is used to visually separate columns; this FlowDocument a Dodger-blue column rule that is 5 pixels wide. The column rule and column gap both take space between columns. In this case, a column gap width of 20 plus a column rule of

width of 5 results in the space between columns being 25 pixels wide, 5 pixels for the column rule, and 10 pixels of column gap on either side of the column rule.



Opzioni tipografiche

23/10/2019 • 2 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) include il supporto per la presentazione dettagliata del contenuto di testo. Il rendering del testo in WPF viene eseguito utilizzando Microsoft ClearType, che migliora la chiarezza e la leggibilità del testo. WPF supporta anche i tipi di carattere OpenType, che offrono funzionalità aggiuntive oltre a quelle definite dal formato TrueType®.

In questa sezione

[Funzionalità tipografiche di WPF](#)

[Panoramica su ClearType](#)

[Impostazioni del Registro di sistema ClearType](#)

[Disegno di testo formattato](#)

[Formattazione del testo avanzata](#)

[Tipi di carattere](#)

[Glifi](#)

[Procedure relative alle proprietà](#)

Vedere anche

- [Typography](#)
- [Documenti in WPF](#)
- [Funzionalità dei tipi di carattere OpenType](#)
- [Ottimizzazione delle prestazioni di applicazioni WPF](#)

Funzionalità tipografiche di WPF

03/02/2020 • 14 minutes to read • [Edit Online](#)

Questo argomento presenta le principali funzionalità tipografiche di WPF. Queste funzionalità includono qualità e prestazioni migliorate per il rendering del testo, supporto tipografico OpenType, testo internazionale migliorato, supporto dei tipi di carattere migliorato e nuove API (Application Programming Interface) di testo.

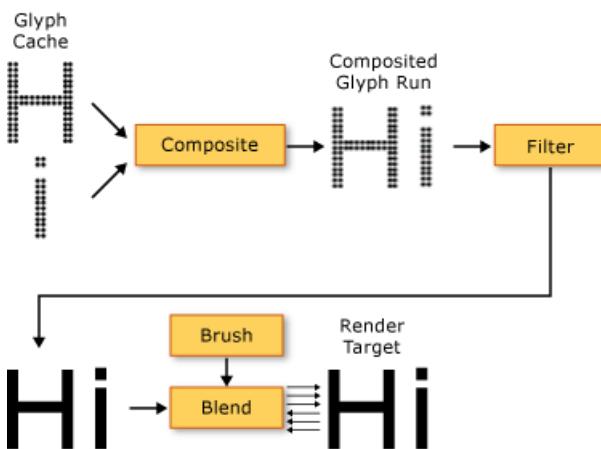
Qualità e prestazioni del testo migliorate

Il rendering del testo in WPF viene eseguito utilizzando Microsoft ClearType, che migliora la chiarezza e la leggibilità del testo. ClearType è una tecnologia software sviluppata da Microsoft che consente di migliorare la leggibilità del testo sugli schermi LCD (Liquid Crystal Display), ad esempio schermi portatili, schermate Pocket PC e monitor Flat Panel. ClearType utilizza il rendering dei subpixel, che consente di visualizzare il testo con una maggiore fedeltà alla propria forma reale allineando i caratteri in una parte frazionaria di un pixel. La risoluzione aggiuntiva aumenta la nitidezza dei piccoli dettagli nella visualizzazione del testo, rendendone più facile la lettura per periodi prolungati. Un altro miglioramento di ClearType in WPF è l'anti-aliasing della direzione y, che smussa le parti superiori e inferiori delle curve superficiali nei caratteri di testo. Per ulteriori informazioni sulle funzionalità ClearType, vedere [Cenni preliminari su ClearType](#).



Testo con anti-aliasing della direzione y ClearType

È possibile applicare l'accelerazione hardware all'intera pipeline di rendering del testo in WPF, purché il computer in uso soddisfi i requisiti hardware minimi richiesti. Se non è possibile eseguire il rendering tramite hardware, viene eseguito il rendering software. L'accelerazione hardware influisce su tutte le fasi della pipeline di rendering del testo, dall'archiviazione di singoli glifi, alla composizione di glifi in esecuzioni di glifi, all'applicazione di effetti all'applicazione dell'algoritmo di fusione ClearType all'output visualizzato finale. Per altre informazioni sull'accelerazione hardware, vedere [Livelli di rendering della grafica](#).



Il testo animato, mediante carattere o glifo, sfrutta inoltre completamente la funzionalità hardware grafica abilitata da WPF, con il risultato di ottenere un'animazione del testo uniforme.

Funzionalità tipografiche avanzate

Il formato del tipo di carattere OpenType è un'estensione del formato di carattere TrueType®. Il formato del tipo

di carattere OpenType è stato sviluppato congiuntamente da Microsoft e Adobe e offre un'ampia gamma di funzionalità tipografiche avanzate. L'oggetto [Typography](#) espone molte delle funzionalità avanzate dei tipi di carattere OpenType, ad esempio alternative stilistiche e ornati. Il Windows SDK fornisce un set di tipi di carattere OpenType di esempio progettati con funzionalità avanzate, ad esempio i tipi di carattere Pericle e Pescadero. Per altre informazioni, vedere [Esempio di pacchetto di tipi di carattere OpenType](#).

Il tipo di carattere OpenType di Pericle contiene glifi aggiuntivi che forniscono alternative stilistiche al set di glifi standard. Il testo seguente mostra glifi con stile alternativo.

ANCIENT GREEK MYTHOLOGY

I glifi ornati sono glifi decorativi che usano ornamenti elaborati spesso associati alla calligrafia. Il testo seguente mostra glifi standard e ornati per il tipo di carattere Pescadero.

A B C D E F G H I J K L M N
ΑΒ ΚΔΕΦ ΓΗΙΚΛΜΝ

Per ulteriori informazioni sulle funzionalità OpenType, vedere [funzionalità dei tipi di carattere OpenType](#).

Supporto del testo internazionale migliorato

WPF offre un migliore supporto del testo internazionale grazie alle funzionalità seguenti:

- Interlinea automatica in tutti i sistemi di scrittura, tramite misurazione adattiva.
- Ampio supporto per il testo internazionale. Per altre informazioni, vedere [Globalizzazione per WPF](#).
- Interruzione di riga, sillabazione e giustificazione in base alla lingua.

Supporto dei tipi di carattere migliorato

WPF offre un migliore supporto dei tipi di carattere grazie alle funzionalità seguenti:

- Unicode per ogni testo. Il comportamento e la selezione del tipo di carattere non richiedono più set di caratteri o tabelle codici.
- Comportamento del tipo di carattere indipendente dalle impostazioni globali, ad esempio le impostazioni locali del sistema.
- Separare i tipi di [FontWeight](#), [FontStretche](#) [FontStyle](#) per la definizione di una [FontFamily](#). Questo garantisce una maggiore flessibilità rispetto alla programmazione Win32, in cui le combinazioni booleane di corsivo e grassetto vengono usate per definire una famiglia di caratteri.
- Direzione di scrittura (orizzontale o verticale) gestita indipendentemente dal nome del carattere.
- Collegamento dei tipi di carattere e fallback dei tipi di carattere in un file XML portabile, usando la tecnologia composite font. I tipi di carattere composti consentono la costruzione di una gamma completa di tipi di carattere multilingua. Offrono inoltre un meccanismo che evita la visualizzazione di glifi mancanti. Per ulteriori informazioni, vedere la sezione Osservazioni nella classe [FontFamily](#).
- Tipi di carattere internazionali compilati da tipi di carattere composti, mediante un gruppo di tipi di carattere di una singola lingua. In questo modo, si risparmiano risorse durante lo sviluppo dei tipi di carattere per più lingue.
- Tipi di carattere composti incorporati in un documento, per offrire maggiore portabilità dei documenti. Per ulteriori informazioni, vedere la sezione Osservazioni nella classe [FontFamily](#).

Nuove API (Application Programming Interface) di testo

WPF fornisce diverse API di testo che gli sviluppatori possono usare quando si include il testo nelle applicazioni. Queste API sono raggruppate in tre categorie:

- **Layout e interfaccia utente.** Controlli di testo comuni per l'interfaccia utente grafica (GUI).
- **Disegno di testo leggero.** Consente di disegnare testo direttamente sugli oggetti.
- **Formattazione del testo avanzata.** Consente di implementare un motore di testo personalizzato.

Layout e interfaccia utente

Al livello più elevato di funzionalità, le API di testo forniscono controlli di interfaccia utente comuni, ad esempio `Label`, `TextBlock`, `TextBox`. Questi controlli offrono gli elementi di Interfaccia utente di base all'interno di un'applicazione e un modo semplice per presentare il testo e interagire con esso. I controlli come `RichTextBox` e `PasswordBox` consentono una gestione del testo più avanzata o specializzata. Le classi e, ad esempio `TextRange`, `TextSelection`, `TextPointer` consentono una manipolazione del testo utile. Questi controlli Interfaccia utente forniscono proprietà quali `FontFamily`, `FontSize`, `FontStyle`, che consentono di controllare il tipo di carattere utilizzato per il rendering del testo.

Uso di effetti bitmap, trasformazioni ed effetti di testo

WPF consente di usare il testo in modi visivamente interessanti grazie a funzionalità come effetti bitmap, trasformazioni ed effetti di testo. L'esempio seguente illustra un tipico effetto di ombreggiatura applicato al testo.



Shadow Text

L'esempio seguente illustra un effetto di ombreggiatura con rumore applicato al testo.



Shadow Text

L'esempio seguente illustra un effetto di alone esterno applicato al testo.



Shadow Text

L'esempio seguente illustra un effetto di sfocatura applicato al testo.



Shadow Text

Nell'esempio seguente la seconda riga del testo è ridimensionata del 150% lungo l'asse x, mentre la terza riga del testo è ridimensionata del 150% lungo l'asse y.



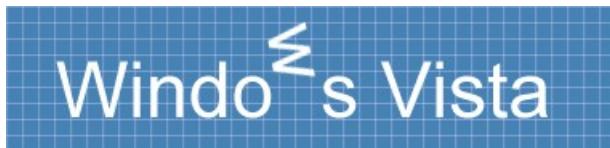
Scaled Text
Scaled Text
Scaled Text

Nell'esempio seguente il testo è inclinato lungo l'asse x.

Skewed Text

Skewed Text

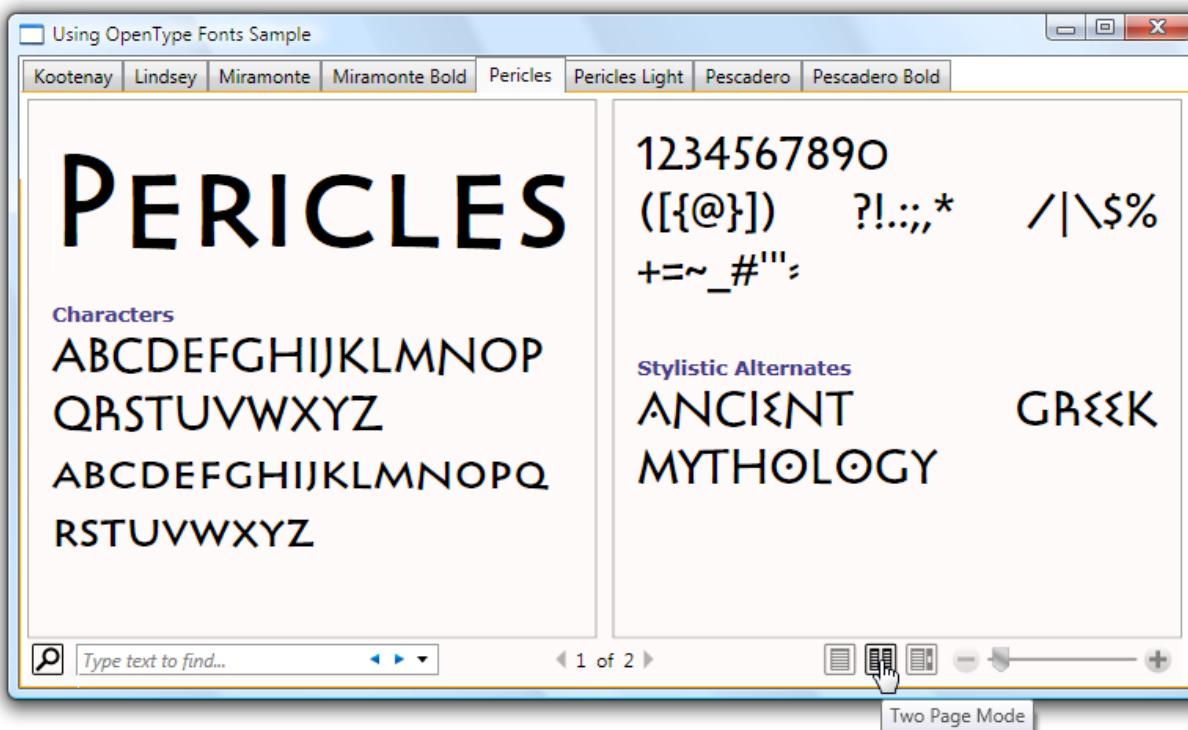
Un oggetto [TextEffect](#) è un oggetto helper che consente di trattare il testo come uno o più gruppi di caratteri in una stringa di testo. L'esempio seguente mostra la rotazione di un singolo carattere. Ogni carattere viene ruotato in modo indipendente a intervalli di 1 secondo.



Uso di documenti dinamici

Oltre ai controlli comuni di Interfaccia utente, WPF offre un controllo di layout per la presentazione del testo, ovvero l'elemento [FlowDocument](#). L'elemento [FlowDocument](#), insieme all'elemento [DocumentViewer](#), fornisce un controllo per grandi quantità di testo con requisiti di layout diversi. I controlli di layout consentono di accedere a funzionalità tipografiche avanzate tramite l'oggetto [Typography](#) e le proprietà correlate al tipo di carattere di altri controlli Interfaccia utente.

Nell'esempio seguente viene illustrato il contenuto di testo ospitato in una [FlowDocumentReader](#), che fornisce supporto per la ricerca, la navigazione, l'impaginazione e la scalabilità del contenuto.



Per altre informazioni, vedere [Documenti in WPF](#).

Disegno di testo leggero

È possibile creare direttamente testo per WPF oggetti usando il metodo [DrawText](#) dell'oggetto [DrawingContext](#). Per utilizzare questo metodo, è necessario creare un oggetto [FormattedText](#). Questo oggetto consente di creare testo su più righe, in cui ogni carattere può essere formattato singolarmente. La funzionalità dell'oggetto [FormattedText](#) contiene la maggior parte delle funzionalità dei flag [DrawText](#) nell'API Windows. Inoltre, l'oggetto [FormattedText](#) contiene funzionalità come il supporto dei puntini di sospensione, in cui vengono visualizzati i puntini di sospensione quando il testo supera i limiti. L'esempio seguente illustra un testo a cui sono stati applicati diversi formati, inclusa una sfumatura lineare sulla seconda e la terza parola.

#

dolor sit amet,
*consectetur
adipiscing elit,*
sed do eiusmod...

È possibile convertire il testo formattato in oggetti [Geometry](#), consentendo di creare altri tipi di testo visivamente interessante. Ad esempio, è possibile creare un oggetto [Geometry](#) in base alla struttura di una stringa di testo.

Spectrum Outline

Gli esempi seguenti illustrano diverse modalità di creazione di effetti visivi interessanti tramite la modifica del tratto, del riempimento e dell'evidenziazione del testo convertito.

Fancy Outlined Text

BUTTERFLIES



Per ulteriori informazioni sull'oggetto [FormattedText](#), vedere [disegno di testo formattato](#).

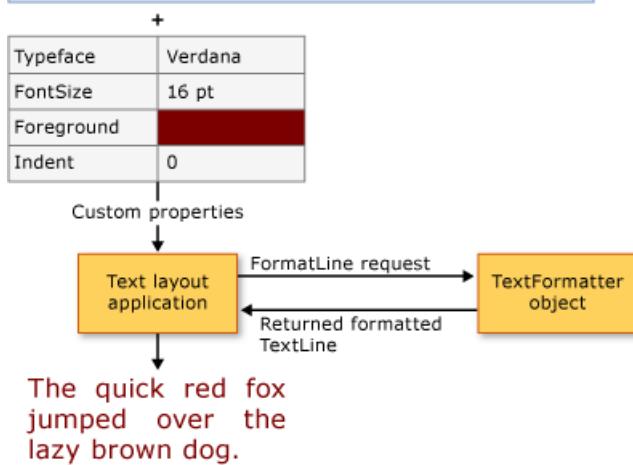
Formattazione del testo avanzata

Al livello più avanzato delle API di testo, WPF offre la possibilità di creare un layout di testo personalizzato usando l'oggetto [TextFormatter](#) e altri tipi nello spazio dei nomi [System.Windows.Media.TextFormatting](#). Il [TextFormatter](#) e le classi associate consentono di implementare un layout di testo personalizzato che supporta la definizione di formati carattere, stili di paragrafo, regole di interruzioni di riga e altre funzionalità di layout per il testo internazionale. Sono pochi i casi in cui occorre eseguire l'override dell'implementazione predefinita del supporto di layout di testo di WPF. Se tuttavia occorre creare un controllo o un'applicazione di modifica del testo, potrebbe essere necessaria un'implementazione diversa da quella predefinita di WPF.

Diversamente da un'API di testo tradizionale, il [TextFormatter](#) interagisce con un client di layout di testo tramite un set di metodi di callback. Richiede che il client fornisca questi metodi in un'implementazione della classe [TextSource](#). Nel diagramma seguente viene illustrata l'interazione del layout del testo tra l'applicazione client e [TextFormatter](#).

Text to display

```
The quick red fox jumped over the lazy brown dog.
```



Per informazioni dettagliate sulla creazione di layout di testo personalizzati, vedere [Formattazione del testo avanzata](#).

Vedere anche

- [FormattedText](#)
- [TextFormatter](#)
- [Panoramica su ClearType](#)
- [Funzionalità dei tipi di carattere OpenType](#)
- [Disegno di testo formattato](#)
- [Formattazione del testo avanzata](#)
- [Testo](#)
- [Microsoft Typography](#)

Cenni preliminari su ClearType

23/10/2019 • 8 minutes to read • [Edit Online](#)

In questo argomento viene fornita una panoramica della tecnologia Microsoft ClearType disponibile nella Windows Presentation Foundation (WPF).

Informazioni generali sulla tecnologia

ClearType è una tecnologia software sviluppata da Microsoft che consente di migliorare la leggibilità del testo sugli schermi LCD (Liquid Crystal Display), ad esempio schermi portatili, schermate Pocket PC e monitor Flat Panel. ClearType funziona tramite l'accesso ai singoli elementi stripe dei colori verticali in ogni pixel di uno schermo LCD. Prima di ClearType, il livello di dettaglio più basso che un computer poteva visualizzare era un singolo pixel, ma con ClearType in esecuzione su un monitor LCD, ora è possibile visualizzare le funzionalità del testo di piccole dimensioni come una frazione di un pixel in larghezza. La risoluzione aggiuntiva aumenta la nitidezza dei piccoli dettagli nella visualizzazione del testo, rendendone più facile la lettura per periodi prolungati.

ClearType disponibile in Windows Presentation Foundation (WPF) è la generazione più recente di ClearType con diversi miglioramenti rispetto alla versione disponibile in Microsoft Windows Graphics Device Interface (GDI).

Posizionamento dei subpixel

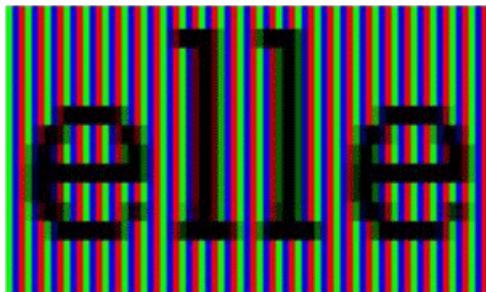
Un miglioramento significativo rispetto alla versione precedente di ClearType è l'utilizzo del posizionamento dei subpixel. A differenza dell'implementazione di ClearType presente in GDI, il ClearType trovato Windows Presentation Foundation (WPF) in consente l'avvio dei glifi all'interno del pixel e non solo del limite iniziale del pixel. Grazie a questa risoluzione aggiuntiva nel posizionamento dei glifi, la spaziatura e le proporzioni dei glifi risultano più precise e coerenti.

Nei due esempi seguenti viene mostrato come è possibile inserire i glifi su qualsiasi limite del subpixel se si usa il posizionamento dei subpixel. Nell'esempio a sinistra viene eseguito il rendering utilizzando la versione precedente del renderer ClearType, che non utilizza il posizionamento dei subpixel. L'esempio a destra viene sottoposto a rendering usando la nuova versione del renderer ClearType, usando il posizionamento dei subpixel. Notare che il rendering dei caratteri **e** e **I** nell'immagine a destra è leggermente diverso in quanto ognuno inizia in corrispondenza di un subpixel diverso. Quando il testo viene visualizzato sullo schermo con le dimensioni normali, questa differenza non è ben visibile a causa dell'elevato contrasto dell'immagine del glifo. Questa operazione è possibile solo grazie a un sofisticato filtro dei colori incorporato in ClearType.

Versione precedente di ClearType



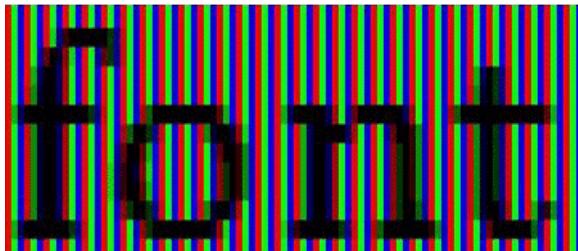
Versione successiva di ClearType



Testo visualizzato con la versione precedente e quella successiva di ClearType

Nei due esempi seguenti viene confrontato l'output del renderer ClearType precedente con la nuova versione del renderer ClearType. Il posizionamento dei subpixel, mostrato a destra, consente di migliorare notevolmente la spaziatura dei tipi sullo schermo, specialmente alle dimensioni più piccole, in cui la differenza tra un subpixel e un pixel intero rappresenta una parte significativa della larghezza del glifo. Notare che la spaziatura tra le lettere è più uniforme nella seconda immagine. Il vantaggio cumulativo del posizionamento dei subpixel all'aspetto complessivo di una schermata di testo è notevolmente maggiore e rappresenta un'evoluzione significativa nella tecnologia ClearType.

Versione precedente di ClearType



Versione successiva di ClearType

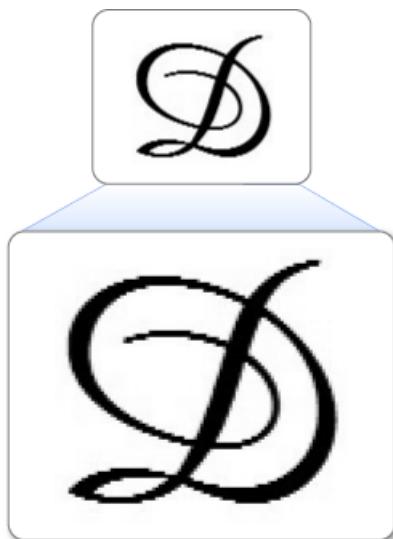


Confronto tra il testo visualizzato con la versione precedente di ClearType e quello visualizzato con la versione successiva

Anti-aliasing della direzione Y

Un altro miglioramento di ClearType Windows Presentation Foundation (WPF) in è l'anti-aliasing della direzione y. ClearType in GDI senza anti-aliasing della direzione y offre una risoluzione migliore sull'asse x, ma non sull'asse y. Nelle parti superiori e inferiori delle curve poco pronunciate, i bordi frastagliati riducono la leggibilità.

L'esempio seguente mostra l'effetto dell'assenza di anti-aliasing della direzione y. In questo caso, i bordi frastagliati nella parte superiore e inferiore della lettera sono chiaramente visibili.

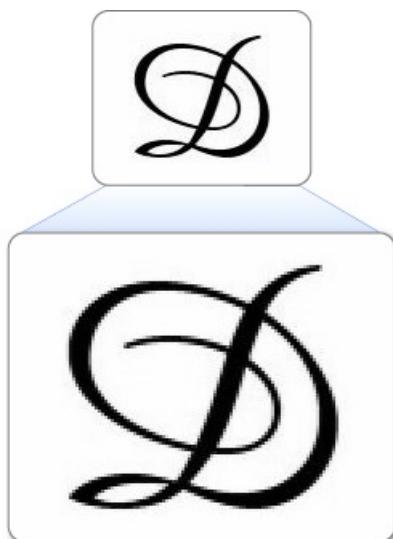


La visualizzazione ingrandita della D Script maiuscola mostra bordi frastagliati sulle curve poco pronunciate

Testo con bordi frastagliati su curve poco pronunciate

ClearType in Windows Presentation Foundation (WPF) fornisce l'anti-aliasing sul livello di direzione y per smussare tutti i bordi frastagliati. Tale funzionalità è particolarmente importante per migliorare la leggibilità delle lingue asiatiche, nelle quali gli ideogrammi presentano un quantità quasi uguale di curve poco pronunciate orizzontali e verticali.

L'esempio seguente illustra gli effetti dell'anti-aliasing della direzione Y. In questo caso, la parte superiore e quella inferiore della lettera mostrano una curva uniforme.



ClearType con anti-aliasing della direzione y smussa i bordi frastagliati sulle curve poco pronunciate e altre caratteristiche della direzione y

Testo con anti-aliasing della direzione y ClearType

Accelerazione hardware

ClearType in Windows Presentation Foundation (WPF) può trarre vantaggio dall'accelerazione hardware per migliorare le prestazioni e ridurre i requisiti di memoria di sistema e di carico della CPU. Utilizzando i pixel shader e la memoria video di una scheda grafica, ClearType fornisce un rendering più veloce del testo, in particolare quando viene utilizzata l'animazione.

ClearType in Windows Presentation Foundation (WPF) non modifica le impostazioni ClearType a livello di sistema. La disabilitazione di ClearType in Windows Presentation Foundation (WPF) Windows imposta l'anti-aliasing sulla modalità scala di grigi. Inoltre, ClearType in Windows Presentation Foundation (WPF) non modifica le impostazioni del PowerToy del sintonizzatore [ClearType](#).

Una delle decisioni relative alla progettazione dell'architettura di Windows Presentation Foundation (WPF) consiste nel disporre di monitor DPI a risoluzione più elevata con supporto migliorato del layout indipendente dalla risoluzione, di diffusione sempre maggiore. Come conseguenza di tale scelta, Windows Presentation

Foundation (WPF) non supporta il rendering del testo con anti-aliasing o le bitmap in alcuni tipi di carattere asiatici in quanto entrambi sono dipendenti dalla risoluzione.

Ulteriori informazioni

[Informazioni su ClearType](#)

[ClearType Tuner PowerToy](#)

Vedere anche

- [Impostazioni del Registro di sistema ClearType](#)

Impostazioni del Registro di sistema ClearType

10/01/2020 • 10 minutes to read • [Edit Online](#)

In questo argomento viene fornita una panoramica delle impostazioni del registro di sistema di Microsoft ClearType utilizzate dalle applicazioni WPF.

Informazioni generali sulla tecnologia

WPF le applicazioni che eseguono il rendering del testo in un dispositivo di visualizzazione utilizzano le funzionalità ClearType per offrire un'esperienza di lettura avanzata. ClearType è una tecnologia software sviluppata da Microsoft che consente di migliorare la leggibilità del testo sugli schermi LCD (Liquid Crystal Display), ad esempio schermi portatili, schermate Pocket PC e monitor Flat Panel. ClearType funziona tramite l'accesso ai singoli elementi stripe dei colori verticali in ogni pixel di uno schermo LCD. Per ulteriori informazioni su ClearType, vedere [Cenni preliminari su ClearType](#).

Il testo di cui è stato eseguito il rendering con ClearType può apparire significativamente diverso quando viene visualizzato in diversi dispositivi di visualizzazione. Ad esempio, un numero ridotto di monitoraggi implementa gli elementi stripe dei colori in ordine blu, verde, rosso anziché l'ordine rosso, verde, blu (RGB) più comune.

Il testo di cui è stato eseguito il rendering con ClearType può anche essere significativamente diverso quando viene visualizzato da utenti con diversi livelli di sensibilità del colore. Alcune persone rilevano leggere differenze di colore meglio di altre.

In ognuno di questi casi, è necessario modificare le funzionalità ClearType per fornire la migliore esperienza di lettura per ogni singolo utente.

Impostazioni Registro di sistema

WPF specifica quattro impostazioni del registro di sistema per il controllo delle funzionalità ClearType:

IMPOSTAZIONE DI	DESCRIZIONE
Livello ClearType	Describe il livello di chiarezza dei colori ClearType.
Livello di gamma	Describe il livello della componente cromatica del pixel per un dispositivo di visualizzazione.
Struttura del pixel	Describe la disposizione dei pixel per un dispositivo di visualizzazione.
Livello di contrasto del testo	Describe il livello di contrasto per il testo visualizzato.

È possibile accedere a queste impostazioni mediante un'utilità di configurazione esterna in grado di fare riferimento alle impostazioni del registro di sistema di WPFClearType identificato. È possibile creare o modificare queste impostazioni anche accedendo direttamente ai valori usando l'editor del registro di sistema di Windows.

Se le impostazioni del registro di sistema WPFClearType non sono impostate (ovvero lo stato predefinito), l'applicazione WPF esegue una query sulle informazioni sui parametri di sistema di Windows per le impostazioni di smussatura dei caratteri.

NOTE

Per informazioni sull'enumerazione dei nomi dei dispositivi di visualizzazione, vedere la `SystemParametersInfo` funzione Win32.

Livello ClearType

Il livello ClearType consente di modificare il rendering del testo in base alla sensibilità del colore e alla percezione di un singolo utente. Per alcune persone, il rendering del testo che usa ClearType al livello più alto non produce la migliore esperienza di lettura.

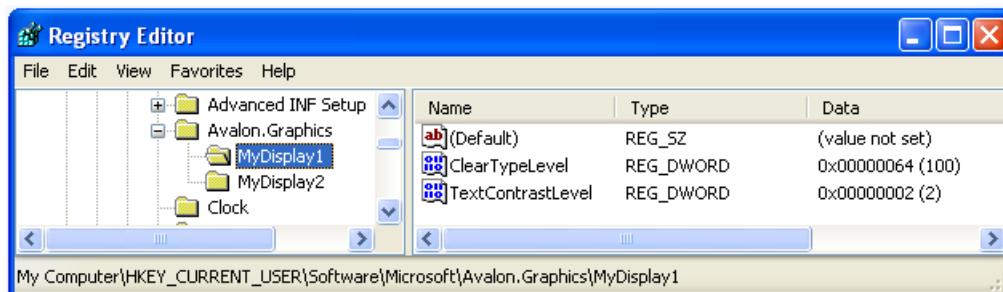
Il livello ClearType è un valore intero compreso tra 0 e 100. Il livello predefinito è 100, che indica che ClearType utilizza la funzionalità massima degli elementi stripe del dispositivo di visualizzazione. Tuttavia, un livello ClearType pari a 0 esegue il rendering del testo come scala di grigi. Impostando il livello ClearType in un punto compreso tra 0 e 100, è possibile creare un livello intermedio adatto alla sensibilità del colore di un utente.

Impostazione del Registro di sistema

Il percorso dell'impostazione del registro di sistema per il livello ClearType è una singola impostazione utente che corrisponde a un nome di dispositivo di visualizzazione specifico:

`HKEY_CURRENT_USER\Software\Microsoft\Avalon.Graphics\<displayname>`

Per ogni nome di dispositivo visualizzato per un utente, viene definito un `ClearTypeLevel` valore DWORD. Lo screenshot seguente mostra l'impostazione dell'editor del registro di sistema per il livello ClearType.



NOTE

Le applicazioni WPF eseguono il rendering del testo in una delle due modalità, con e senza ClearType. Quando viene eseguito il rendering del testo senza ClearType, viene definito rendering in scala di grigi.

Livello di gamma

Il livello di gamma fa riferimento alla relazione non lineare tra un valore in pixel e la luminanza. L'impostazione del livello di gamma deve corrispondere alle caratteristiche fisiche del dispositivo di visualizzazione, in caso contrario possono verificarsi distorsioni nell'output del rendering. Ad esempio, il testo potrebbe sembrare troppo ampio o troppo piccolo oppure le frange dei colori possono apparire sui bordi delle steli verticali dei glifi.

Il livello di gamma è un valore intero compreso tra 1000 e 2200. Il livello predefinito è 1900.

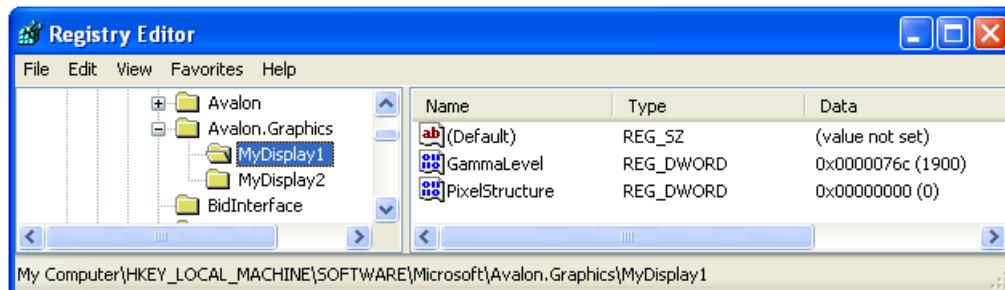
Impostazione del Registro di sistema

Il percorso dell'impostazione del Registro di sistema per il livello di gamma è un'impostazione del computer locale che corrisponde al nome di uno specifico dispositivo di visualizzazione:

`HKEY_LOCAL_MACHINE\Software\Microsoft\Avalon.Graphics\<displayname>`

Per ogni nome di dispositivo visualizzato per un utente, viene definito un `GammaLevel` valore DWORD. Lo

screenshot seguente mostra l'impostazione dell'Editor del Registro di sistema per il livello di gamma.



Struttura del pixel

La struttura del pixel descrive il tipo di pixel alla base di un dispositivo di visualizzazione. La struttura del pixel è definita come uno di tre tipi:

TIPO DI	VALORE	DESCRIZIONE
Semplice	0	Il dispositivo di visualizzazione non ha struttura del pixel. In questo caso le sorgenti di luce per ogni colore sono distribuite in modo uniforme nell'area dei pixel, condizione nota come rendering in scala di grigi. Questo è il funzionamento di un dispositivo di visualizzazione standard. ClearType non viene mai applicato al testo sottoposto a rendering.
RGB	1	Il dispositivo di visualizzazione dispone di pixel costituiti da tre strisce nell'ordine seguente: rosso, verde e blu. ClearType viene applicato al testo sottoposto a rendering.
BGR	2	Il dispositivo di visualizzazione dispone di pixel costituiti da tre strisce nell'ordine seguente: blu, verde e rosso. ClearType viene applicato al testo sottoposto a rendering. Si noti come l'ordine viene invertito rispetto al tipo RGB.

La struttura dei pixel corrisponde a un valore intero compreso tra 0 e 2. Il livello predefinito è 0, che rappresenta una struttura del pixel flat.

NOTE

Per informazioni sull'enumerazione dei nomi dei dispositivi di visualizzazione, vedere la [EnumDisplayDevices](#) funzione Win32.

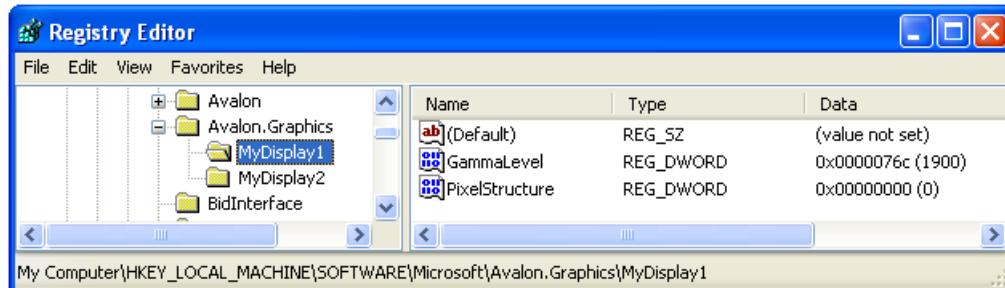
Impostazione del Registro di sistema

Il percorso dell'impostazione del Registro di sistema per la struttura del pixel è un'impostazione del computer locale che corrisponde al nome di uno specifico dispositivo di visualizzazione:

`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Avalon.Graphics\<display Name>`

Per ogni nome di dispositivo visualizzato per un utente, viene definito un `PixelStructure` valore DWORD. Lo

screenshot seguente mostra l'impostazione dell'Editor del Registro di sistema per la struttura del pixel.



Livello di contrasto del testo

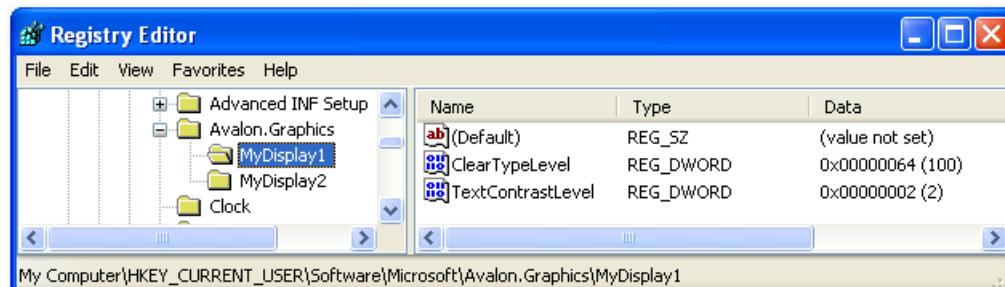
Il livello di contrasto del testo consente di regolare il rendering del testo in base alle larghezze del gambo dei glifi. Il livello di contrasto del testo è un valore intero compreso tra 0 e 6, dove il valore più grande indica una maggiore larghezza. Il livello predefinito è 1.

Impostazione del Registro di sistema

Il percorso dell'impostazione del Registro di sistema per il livello di contrasto del testo è un'impostazione del singolo utente che corrisponde al nome di uno specifico dispositivo di visualizzazione:

```
HKEY_CURRENT_USER\Software\Microsoft\Avalon.Graphics\<displayName>
```

Per ogni nome di dispositivo visualizzato per un utente, viene definito un `TextContrastLevel` valore DWORD. Lo screenshot seguente mostra l'impostazione dell'Editor del Registro di sistema per il livello di contrasto del testo.



Vedere anche

- [Panoramica su ClearType](#)
- [ClearType Antialiasing \(Anti-aliasing ClearType\)](#)

Disegno di testo formattato

10/02/2020 • 13 minutes to read • [Edit Online](#)

In questo argomento viene fornita una panoramica delle funzionalità dell'oggetto [FormattedText](#), che offre un controllo di basso livello per il disegno di testo nelle applicazioni Windows Presentation Foundation (WPF).

Informazioni generali sulla tecnologia

L'oggetto [FormattedText](#) consente di creare testo su più righe, in cui ogni carattere del testo può essere formattato singolarmente. L'esempio seguente mostra un testo a cui sono stati applicati diversi formati.

Lorem ipsum
dolor sit amet,
consectetur
adipisicing elit,
sed do eiusmod...

NOTE

Per gli sviluppatori che eseguono la migrazione dall'API Win32, nella tabella della sezione [migrazione Win32](#) sono elencati i flag DrawText di Win32 e l'equivalente approssimativo in Windows Presentation Foundation (WPF).

Motivi per l'uso del testo formattato

WPF include più controlli per la creazione di testo sullo schermo. Ogni controllo è destinato a uno scenario diverso e dispone di un proprio elenco di funzionalità e limitazioni. In generale, l'elemento [TextBlock](#) deve essere utilizzato quando è necessario il supporto di testo limitato, ad esempio una breve frase in una interfaccia utente. [Label](#) può essere utilizzato quando è richiesto un supporto di testo minimo. Per altre informazioni, vedere [Documenti in WPF](#).

L'oggetto [FormattedText](#) fornisce funzionalità di formattazione del testo maggiori rispetto ai controlli Windows Presentation Foundation (WPF) testo e può essere utile nei casi in cui si desidera utilizzare il testo come elemento decorativo. Per altre informazioni, vedere la sezione seguente [Conversione del testo formattato in una geometria](#).

Inoltre, l'oggetto [FormattedText](#) è utile per la creazione di oggetti derivati da [DrawingVisual](#)testo. [DrawingVisual](#) è una classe di disegno semplificata utilizzata per il rendering di forme, immagini o testo. Per altre informazioni, vedere [Esempio di hit test mediante DrawingVisual](#).

Uso dell'oggetto FormattedText

Per creare un testo formattato, chiamare il costruttore [FormattedText](#) per creare un oggetto [FormattedText](#). Dopo aver creato la stringa di testo formattato iniziale, è possibile applicare una gamma di stili di formattazione.

Utilizzare la proprietà [MaxTextWidth](#) per vincolare il testo a una larghezza specifica. Il testo andrà a capo automaticamente per evitare di superare la larghezza specificata. Utilizzare la proprietà [MaxTextHeight](#) per vincolare il testo a un'altezza specifica. Saranno visualizzati dei puntini di sospensione ("...") nel caso in cui il testo superi l'altezza specificata.

→
Lorem ipsum dolor
sit amet,
consectetur...

Text wordwraps when it
exceeds the width

adipisicing elit, sed do eiusmod tempor

Text shows ellipsis when
it exceeds the height

È possibile applicare vari stili di formattazione a uno o più caratteri. È ad esempio possibile chiamare entrambi i metodi [SetFontSize](#) e [SetForegroundBrush](#) per modificare la formattazione dei primi cinque caratteri del testo.

Nell'esempio di codice seguente viene creato un oggetto [FormattedText](#), quindi vengono applicati diversi stili di formattazione al testo.

```
protected override void OnRender(DrawingContext drawingContext)
{
    string testString = "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor";

    // Create the initial formatted text string.
    FormattedText formattedText = new FormattedText(
        testString,
        CultureInfo.GetCultureInfo("en-us"),
        FlowDirection.LeftToRight,
        new Typeface("Verdana"),
        32,
        Brushes.Black);

    // Set a maximum width and height. If the text overflows these values, an ellipsis "..." appears.
    formattedText.MaxTextWidth = 300;
    formattedText.MaxTextHeight = 240;

    // Use a larger font size beginning at the first (zero-based) character and continuing for 5 characters.
    // The font size is calculated in terms of points -- not as device-independent pixels.
    formattedText.SetFontSize(36 * (96.0 / 72.0), 0, 5);

    // Use a Bold font weight beginning at the 6th character and continuing for 11 characters.
    formattedText.SetFontWeight(FontWeights.Bold, 6, 11);

    // Use a linear gradient brush beginning at the 6th character and continuing for 11 characters.
    formattedText.SetForegroundBrush(
        new LinearGradientBrush(
            Colors.Orange,
            Colors.Teal,
            90.0),
        6, 11);

    // Use an Italic font style beginning at the 28th character and continuing for 28 characters.
    formattedText.SetFontStyle(FontStyles.Italic, 28, 28);

    // Draw the formatted text string to the DrawingContext of the control.
    drawingContext.DrawText(formattedText, new Point(10, 0));
}
```

```

Protected Overrides Sub OnRender(ByVal drawingContext As DrawingContext)
    Dim testString As String = "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor"

    ' Create the initial formatted text string.
    Dim formattedText As New FormattedText(testString, CultureInfo.GetCultureInfo("en-us"),
    FlowDirection.LeftToRight, New Typeface("Verdana"), 32, Brushes.Black)

    ' Set a maximum width and height. If the text overflows these values, an ellipsis "..." appears.
    formattedText.MaxTextWidth = 300
    formattedText.MaxTextHeight = 240

    ' Use a larger font size beginning at the first (zero-based) character and continuing for 5 characters.
    ' The font size is calculated in terms of points -- not as device-independent pixels.
    formattedText.SetFontSize(36 * (96.0 / 72.0), 0, 5)

    ' Use a Bold font weight beginning at the 6th character and continuing for 11 characters.
    formattedText.SetFontWeight(FontWeights.Bold, 6, 11)

    ' Use a linear gradient brush beginning at the 6th character and continuing for 11 characters.
    formattedText.SetForegroundBrush(New LinearGradientBrush(Colors.Orange, Colors.Teal, 90.0), 6, 11)

    ' Use an Italic font style beginning at the 28th character and continuing for 28 characters.
    formattedTextSetFontStyle(FontStyles.Italic, 28, 28)

    ' Draw the formatted text string to the DrawingContext of the control.
    drawingContext.DrawText(formattedText, New Point(10, 0))
End Sub

```

Unità di misura delle dimensioni del carattere

Come per gli altri oggetti di testo nelle applicazioni Windows Presentation Foundation (WPF), l'oggetto [FormattedText](#) usa pixel indipendenti dal dispositivo come unità di misura. Tuttavia, la maggior parte delle applicazioni Win32 utilizza punti come unità di misura. Se si vuole usare il testo visualizzato in unità di punti in Windows Presentation Foundation (WPF) applicazioni, è necessario convertire le unità indipendenti dal dispositivo (1/1/96 pollice per unità) in punti. Nell'esempio di codice seguente viene illustrato come eseguire questa conversione.

```
// The font size is calculated in terms of points -- not as device-independent pixels.
formattedText.SetFontSize(36 * (96.0 / 72.0), 0, 5);
```

```
' The font size is calculated in terms of points -- not as device-independent pixels.
formattedText.SetFontSize(36 * (96.0 / 72.0), 0, 5)
```

Conversione del testo formattato in una geometria

È possibile convertire il testo formattato in oggetti [Geometry](#), consentendo di creare altri tipi di testo visivamente interessante. Ad esempio, è possibile creare un oggetto [Geometry](#) in base alla struttura di una stringa di testo.

Spectrum Outline

Gli esempi seguenti illustrano diverse modalità di creazione di effetti visivi interessanti tramite la modifica del tratto, del riempimento e dell'evidenziazione del testo convertito.



BUTTERFLIES

WILDFIRE

Quando il testo viene convertito in un oggetto [Geometry](#), non è più una raccolta di caratteri, non è possibile modificare i caratteri nella stringa di testo. Tuttavia, è possibile intervenire sull'aspetto del testo convertito modificandone le proprietà del tratto e del riempimento. Il tratto fa riferimento alla struttura del testo convertito, mentre il riempimento fa riferimento all'area all'interno della struttura del testo convertito. Per altre informazioni, vedere [Creare testo con contorni](#).

È anche possibile convertire il testo formattato in un oggetto [PathGeometry](#) e usare l'oggetto per evidenziare il testo. Ad esempio, è possibile applicare un'animazione all'oggetto [PathGeometry](#) in modo che l'animazione segua il contorno del testo formattato.

Nell'esempio seguente viene illustrato il testo formattato che è stato convertito in un oggetto [PathGeometry](#). Un'ellisse animata segue il percorso dei tratti del testo di cui è stato eseguito il rendering.

Hello World!

Sfera che segue la geometria del percorso del testo

Per altre informazioni, vedere [Procedura: Creare animazioni PathGeometry per il testo](#).

È possibile creare altri usi interessanti per il testo formattato dopo che è stato convertito in un oggetto [PathGeometry](#). Ad esempio, è possibile ritagliare video da visualizzare all'interno del testo.



Migrazione Win32

Le funzionalità di [FormattedText](#) per il disegno del testo sono simili alle funzionalità della funzione DrawText di Win32. Per gli sviluppatori che eseguono la migrazione dall'API Win32, nella tabella seguente sono elencati i flag DrawText di Win32 e l'equivalente approssimativo in Windows Presentation Foundation (WPF).

FLAG DRAWTEXT	ELEMENTO CORRISPONDENTE IN WPF	NOTE
DT_BOTTOM	Height	Utilizzare la proprietà Height per calcolare una posizione "y" di DrawText Win32 appropriata.

FLAG DRAWTEXT	ELEMENTO CORRISPONDENTE IN WPF	NOTE
DT_CALCRECT	Height , Width	Utilizzare le proprietà Height e Width per calcolare il rettangolo di output.
DT_CENTER	TextAlignment	Utilizzare la proprietà TextAlignment con il valore impostato su Center .
DT_EDITCONTROL	nessuno	Non obbligatorio. La larghezza dello spazio e il rendering dell'ultima riga sono uguali a quelli del controllo di modifica del framework.
DT_END_ELLIPSIS	Trimming	Utilizzare la proprietà Trimming con il valore CharacterEllipsis . Usare WordEllipsis per ottenere DT_END_ELLIPSIS Win32 con i puntini di sospensione DT_WORD_ELLIPSIS, in questo caso i puntini di sospensione dei caratteri si verificano solo su parole che non rientrano in una singola riga.
DT_EXPAND_TABS	nessuno	Non obbligatorio. Le tabulazioni vengono espanso automaticamente per interrompersi ogni 4 em, all'incirca a una larghezza di 8 caratteri indipendenti dalla lingua.
DT_EXTERNALLEADING	nessuno	Non obbligatorio. L'interlinea esterna è sempre inclusa nell'interlinea. Utilizzare la proprietà LineHeight per creare spaziatura riga definita dall'utente.
DT_HIDEPREFIX	nessuno	Non supportato. Rimuovere ' &' dalla stringa prima di costruire l'oggetto FormattedText .
DT_LEFT	TextAlignment	Impostazione predefinita per l'allineamento del testo. Utilizzare la proprietà TextAlignment con il valore impostato su Left . (solo WPF).
DT_MODIFYSTRING	nessuno	Non supportato.
DT_NOCLIP	VisualClip	Il ritaglio non viene eseguito automaticamente. Se si desidera ritagliare il testo, utilizzare la proprietà VisualClip .
DT_NOFULLWIDTHCHARBREAK	nessuno	Non supportato.
DT_NOPREFIX	nessuno	Non obbligatorio. Il carattere "&" nelle stringhe viene sempre considerato un carattere normale.
DT_PATHELLIPSIS	nessuno	Utilizzare la proprietà Trimming con il valore WordEllipsis .

FLAG DRAWTEXT	ELEMENTO CORRISPONDENTE IN WPF	NOTE
DT_PREFIX	nessuno	Non supportato. Se si desidera utilizzare caratteri di sottolineatura per il testo, ad esempio un tasto di scelta rapida o un collegamento, utilizzare il metodo SetTextDecorations .
DT_PREFIXONLY	nessuno	Non supportato.
DT_RIGHT	TextAlignment	Utilizzare la proprietà TextAlignment con il valore impostato su Right . (solo WPF).
DT_RTLREADING	FlowDirection	Impostare la proprietà FlowDirection su RightToLeft .
DT_SINGLELINE	nessuno	Non obbligatorio. FormattedText gli oggetti si comportano come un controllo a riga singola, a meno che non sia impostata la proprietà MaxTextWidth o il testo contenga un ritorno a capo/avanzamento riga (CR/LF).
DT_TABSTOP	nessuno	Nessun supporto per le posizioni delle tabulazioni definite dall'utente.
DT_TOP	Height	Non obbligatorio. Il testo è giustificato nella parte superiore per impostazione predefinita. È possibile definire altri valori di posizionamento verticale utilizzando la proprietà Height per calcolare una posizione "y" di DrawText Win32 appropriata.
DT_VCENTER	Height	Utilizzare la proprietà Height per calcolare una posizione "y" di DrawText Win32 appropriata.
DT_WORDBREAK	nessuno	Non obbligatorio. L'interruzione delle parole si verifica automaticamente con FormattedText oggetti. Non è possibile disabilitare questa impostazione.
DT_WORD_ELLIPSIS	Trimming	Utilizzare la proprietà Trimming con il valore WordEllipsis .

Vedere anche

- [FormattedText](#)
- [Documenti in WPF](#)
- [Funzionalità tipografiche di WPF](#)
- [Creare testo con contorni](#)
- [Procedura: Creare animazioni PathGeometry per il testo](#)

Formattazione del testo avanzata

10/02/2020 • 13 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) fornisce un set di API affidabile per includere testo nell'applicazione. Le API di layout e interfaccia utente, ad esempio [TextBlock](#), forniscono gli elementi più comuni e di uso generale per la presentazione del testo. Il disegno di API, ad esempio [GlyphRunDrawing](#) e [FormattedText](#), fornisce un mezzo per includere testo formattato nei disegni. Al livello più avanzato, WPF fornisce un motore di formattazione del testo estendibile per controllare ogni aspetto della presentazione del testo, ad esempio la gestione dell'archivio di testo, la gestione della formattazione delle esecuzioni del testo e la gestione di oggetti incorporati.

In questo argomento viene fornita un'introduzione alla formattazione del testo WPF. Si concentra sull'implementazione del client e sull'uso del motore di formattazione del testo WPF.

NOTE

Tutti gli esempi di codice all'interno di questo documento sono disponibili nell'esempio relativo alla [formattazione avanzata del testo](#).

Prerequisites

Questo argomento presuppone che l'utente abbia familiarità con le API di livello superiore usate per la presentazione del testo. La maggior parte degli scenari utente non richiede le API avanzate di formattazione del testo illustrate in questo argomento. Per un'introduzione alle diverse API di testo, vedere [documenti in WPF](#).

Formattazione del testo avanzata

Il layout del testo e i controlli Interfaccia utente in WPF forniscono le proprietà di formattazione che consentono di includere facilmente testo formattato nell'applicazione. Questi controlli espongono numerose proprietà per la gestione della presentazione del testo, inclusi il carattere tipografico, le dimensioni e il colore. In circostanze normali, questi controlli sono in grado di gestire la maggior parte degli scenari di presentazione del testo nell'applicazione. Tuttavia, alcuni scenari avanzati richiedono il controllo dell'archiviazione del testo e della presentazione del testo. WPF fornisce un motore di formattazione del testo estendibile a questo scopo.

Le funzionalità avanzate di formattazione del testo disponibili in WPF sono costituite da un motore di formattazione del testo, da un archivio di testo, da sequenze di testo e da proprietà di formattazione. Il motore di formattazione del testo [TextFormatter](#) crea righe di testo da utilizzare per la presentazione. Questa operazione viene eseguita avviando il processo di formattazione della riga e chiamando il [FormatLine](#) del formattatore di testo. Il formattatore di testo recupera le esecuzioni di testo dall'archivio di testo chiamando il metodo [GetTextRun](#) dell'archivio. Gli oggetti [TextRun](#) vengono quindi formati in [TextLine](#) oggetti dal formattatore di testo e assegnati all'applicazione per l'ispezione o la visualizzazione.

Uso del formattatore di testo

[TextFormatter](#) è il motore di formattazione del testo WPF e fornisce servizi per la formattazione e l'infrazione di righe di testo. Il formattatore di testo può gestire diversi formati di carattere del testo e stili di paragrafo e include il supporto per il layout di testo internazionale.

Diversamente da un'API di testo tradizionale, il [TextFormatter](#) interagisce con un client di layout di testo tramite un set di metodi di callback. Richiede che il client fornisca questi metodi in un'implementazione della classe [TextSource](#). Nel diagramma seguente viene illustrata l'interazione del layout del testo tra l'applicazione client e

TextFormatter.

Text to display

```
The quick red fox jumped over the lazy brown dog.
```

+

Typeface	Verdana
FontSize	16 pt
Foreground	
Indent	0

Custom properties

Text layout application

TextFormatter object

FormatLine request

Returned formatted
TextLine

The quick red fox
jumped over the
lazy brown dog.

Il formattatore di testo viene utilizzato per recuperare le righe di testo formattato dall'archivio di testo, che è un'implementazione di [TextSource](#). Questa operazione viene eseguita creando prima un'istanza del formattatore di testo utilizzando il metodo [Create](#). Questo metodo crea un'istanza del formattatore di testo e imposta i valori massimi di altezza e larghezza della riga. Non appena viene creata un'istanza del formattatore di testo, il processo di creazione della riga viene avviato chiamando il metodo [FormatLine](#). [TextFormatter](#) richiama l'origine del testo per recuperare il testo e i parametri di formattazione per le esecuzioni del testo che formano una riga.

L'esempio seguente illustra il processo di formattazione di un archivio di testo. L'oggetto [TextFormatter](#) viene utilizzato per recuperare righe di testo dall'archivio di testo e quindi formattare la riga di testo per il disegno nel [DrawingContext](#).

```
// Create a DrawingGroup object for storing formatted text.
textDest = new DrawingGroup();
DrawingContext dc = textDest.Open();

// Update the text store.
_textStore.Text = textToFormat.Text;
_textStore.FontRendering = _currentRendering;

// Create a TextFormatter object.
TextFormatter formatter = TextFormatter.Create();

// Format each line of text from the text store and draw it.
while (textStorePosition < _textStore.Text.Length)
{
    // Create a textline from the text store using the TextFormatter object.
    using (TextLine myTextLine = formatter.FormatLine(
        _textStore,
        textStorePosition,
        96*6,
        new GenericTextParagraphProperties(_currentRendering),
        null))
    {
        // Draw the formatted text into the drawing context.
        myTextLine.Draw(dc, linePosition, InvertAxes.None);

        // Update the index position in the text store.
        textStorePosition += myTextLine.Length;

        // Update the line position coordinate for the displayed line.
        linePosition.Y += myTextLine.Height;
    }
}

// Persist the drawn text content.
dc.Close();

// Display the formatted text in the DrawingGroup object.
myDrawingBrush.Drawing = textDest;
```

```

' Create a DrawingGroup object for storing formatted text.
textDest = New DrawingGroup()
Dim dc As DrawingContext = textDest.Open()

' Update the text store.
_textStore.Text = textToFormat.Text
_textStore.FontRendering = _currentRendering

' Create a TextFormatter object.
Dim formatter As TextFormatter = TextFormatter.Create()

' Format each line of text from the text store and draw it.
Do While textStorePosition < _textStore.Text.Length
    ' Create a textline from the text store using the TextFormatter object.
    Using myTextLine As TextLine = formatter.FormatLine(_textStore, textStorePosition, 96*6, New
GenericTextParagraphProperties(_currentRendering), Nothing)
        ' Draw the formatted text into the drawing context.
        myTextLine.Draw(dc, linePosition, InvertAxes.None)

        ' Update the index position in the text store.
        textStorePosition += myTextLine.Length

        ' Update the line position coordinate for the displayed line.
        linePosition.Y += myTextLine.Height
    End Using
Loop

' Persist the drawn text content.
dc.Close()

' Display the formatted text in the DrawingGroup object.
myDrawingBrush.Drawing = textDest

```

Implementazione dell'archivio di testo del client

Quando si estende il motore di formattazione del testo, è necessario implementare e gestire tutti gli aspetti dell'archivio di testo. Non si tratta di un'attività elementare. L'archivio di testo è responsabile del rilevamento delle proprietà delle sequenze di testo, delle proprietà dei paragrafi, degli oggetti incorporati e di altri contenuti simili. Fornisce anche il formattatore di testo con singoli oggetti [TextRun](#) che il formattatore di testo usa per creare [TextLine](#) oggetti.

Per gestire la virtualizzazione dell'archivio di testo, l'archivio di testo deve essere derivato da [TextSource](#). [TextSource](#) definisce il metodo usato dal formattatore di testo per recuperare le sequenze di testo dall'archivio di testo. [GetTextRun](#) è il metodo utilizzato dal formattatore di testo per recuperare le esecuzioni di testo utilizzate nella formattazione della riga. La chiamata a [GetTextRun](#) viene eseguita ripetutamente dal formattatore di testo fino a quando non si verifica una delle condizioni seguenti:

- Viene restituito un [TextEndOfLine](#) o una sottoclasse.
- La larghezza accumulata delle esecuzioni di testo supera la lunghezza massima consentita nella chiamata per creare il formattatore di testo o la chiamata al metodo [FormatLine](#) del formattatore di testo.
- Viene restituita una sequenza di nuova riga Unicode, ad esempio "CF", "LF" o "CRLF".

Inserimento delle sequenze di testo

Il fulcro del processo di formattazione del testo è rappresentato dall'interazione tra il formattatore di testo e l'archivio di testo. L'implementazione di [TextSource](#) fornisce il formattatore di testo con gli oggetti di [TextRun](#) e le proprietà con le quali formattare le esecuzioni di testo. Questa interazione viene gestita dal metodo [GetTextRun](#), che viene chiamato dal formattatore di testo.

Nella tabella seguente vengono illustrati alcuni degli oggetti [TextRun](#) predefiniti.

TIPO DI TEXTRUN	USO
TextCharacters	Sequenza di testo specializzata usata per passare nuovamente al formattatore di testo una rappresentazione dei glifi dei caratteri.
TextEmbeddedObject	Sequenza di testo specializzata usata per fornire contenuto nel quale la misurazione, l'hit testing e il disegno vengono eseguite come un'unica operazione, ad esempio un pulsante o un'immagine all'interno del testo.
TextEndOfLine	Sequenza di testo specializzata usata per contrassegnare la fine di una riga.
TextEndOfParagraph	Sequenza di testo specializzata usata per contrassegnare la fine di un paragrafo.
TextEndOfSegment	Sequenza di testo specializzata usata per contrassegnare la fine di un segmento, ad esempio per terminare l'ambito interessato da un'esecuzione TextModifier precedente.
TextHidden	Sequenza di testo specializzata usata per contrassegnare un intervallo di caratteri nascosti.
TextModifier	Sequenza di testo specializzata usata per modificare le proprietà delle sequenze di testo nel relativo ambito. L'ambito si estende alla successiva sequenza di testo TextEndOfSegment corrispondente o al TextEndOfParagraph successivo.

Uno degli oggetti [TextRun](#) predefiniti può essere sottoclassato. Ciò consente all'origine del testo di fornire al formattatore di testo sequenze di testo che includono dati personalizzati.

Nell'esempio seguente viene illustrato un metodo [GetTextRun](#). Questo archivio di testo restituisce [TextRun](#) oggetti al formattatore di testo per l'elaborazione.

```

// Used by the TextFormatter object to retrieve a run of text from the text source.
public override TextRun GetTextRun(int textSourceCharacterIndex)
{
    // Make sure text source index is in bounds.
    if (textSourceCharacterIndex < 0)
        throw new ArgumentOutOfRangeException("textSourceCharacterIndex", "Value must be greater than 0.");
    if (textSourceCharacterIndex >= _text.Length)
    {
        return new TextEndOfParagraph(1);
    }

    // Create TextCharacters using the current font rendering properties.
    if (textSourceCharacterIndex < _text.Length)
    {
        return new TextCharacters(
            _text,
            textSourceCharacterIndex,
            _text.Length - textSourceCharacterIndex,
            new GenericTextRunProperties(_currentRendering));
    }

    // Return an end-of-paragraph if no more text source.
    return new TextEndOfParagraph(1);
}

```

```

' Used by the TextFormatter object to retrieve a run of text from the text source.
Public Overrides Function GetTextRun(ByVal textSourceCharacterIndex As Integer) As TextRun
    ' Make sure text source index is in bounds.
    If textSourceCharacterIndex < 0 Then
        Throw New ArgumentOutOfRangeException("textSourceCharacterIndex", "Value must be greater than 0.")
    End If
    If textSourceCharacterIndex >= _text.Length Then
        Return New TextEndOfParagraph(1)
    End If

    ' Create TextCharacters using the current font rendering properties.
    If textSourceCharacterIndex < _text.Length Then
        Return New TextCharacters(_text, textSourceCharacterIndex, _text.Length - textSourceCharacterIndex, New
GenericTextRunProperties(_currentRendering))
    End If

    ' Return an end-of-paragraph if no more text source.
    Return New TextEndOfParagraph(1)
End Function

```

NOTE

In questo esempio, l'archivio di testo fornisce a tutto il testo le stesse proprietà. Gli archivi di testo avanzati devono implementare una gestione personalizzata dell'estensione in modo da consentire ai singoli caratteri di avere proprietà diverse.

Specifiche delle proprietà di formattazione

gli oggetti [TextRun](#) vengono formattati usando le proprietà fornite dall'archivio di testo. Queste proprietà sono disponibili in due tipi, [TextParagraphProperties](#) e [TextRunProperties](#). [TextParagraphProperties](#) gestisce le proprietà inclusivi di paragrafo, ad esempio [TextAlignment](#) e [FlowDirection](#). [TextRunProperties](#) sono proprietà che possono essere diverse per ogni sequenza di testo all'interno di un paragrafo, ad esempio il pennello in primo piano, [Typeface](#) e le dimensioni del carattere. Per implementare il paragrafo personalizzato e i tipi di proprietà della sequenza di testo personalizzata, l'applicazione deve creare classi che derivano rispettivamente da

[TextParagraphProperties](#) e [TextRunProperties](#).

Vedere anche

- [Funzionalità tipografiche di WPF](#)
- [Documenti in WPF](#)

Tipi di carattere (WPF)

03/02/2020 • 2 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) include il supporto per la presentazione dettagliata del testo con i tipi di carattere OpenType. Con la Windows SDK è incluso un pacchetto di esempio di tipi di carattere OpenType.

Contenuto della sezione

[Funzionalità dei tipi di carattere OpenType](#)

[Includere i tipi di carattere nel pacchetto delle applicazioni](#)

[Esempio di pacchetto di tipi di carattere OpenType](#)

[Procedure relative alla struttura ad albero e alla serializzazione degli elementi](#)

Vedere anche

- [FontStyle](#)
- [SystemFonts](#)
- [Documenti in WPF](#)
- [Funzionalità tipografiche di WPF](#)

Funzionalità dei tipi di carattere OpenType

10/02/2020 • 22 minutes to read • [Edit Online](#)

In questo argomento viene fornita una panoramica di alcune delle principali funzionalità della tecnologia dei tipi di carattere OpenType in Windows Presentation Foundation (WPF).

Formato dei tipi di carattere OpenType

Il formato del tipo di carattere OpenType è un'estensione del formato di carattere TrueType®, che aggiunge il supporto per i dati del tipo di carattere PostScript. Il formato del tipo di carattere OpenType è stato sviluppato congiuntamente da Microsoft e Adobe Corporation. I tipi di carattere OpenType e i servizi del sistema operativo che supportano i tipi di carattere OpenType consentono agli utenti di installare e utilizzare in modo semplice i tipi di carattere, indipendentemente dal fatto che i tipi di carattere contengano strutture TrueType o CFF (PostScript).

Il formato del tipo di carattere OpenType soddisfa le seguenti richieste di sviluppo:

- Supporto multipiattaforma più esteso.
- Supporto migliore per set di caratteri internazionali.
- Migliore protezione per i dati dei tipi di carattere.
- Dimensioni di file minori per rendere più efficiente la distribuzione dei tipi di carattere.
- Supporto più ampio per il controllo tipografico avanzato.

NOTE

Il Windows SDK contiene un set di tipi di carattere OpenType di esempio che è possibile usare con le applicazioni Windows Presentation Foundation (WPF). Questi tipi di carattere offrono la maggior parte delle funzionalità descritte nelle altre sezioni di questo argomento. Per altre informazioni, vedere [Esempio di pacchetto di tipi di carattere OpenType](#).

Per informazioni dettagliate sul formato dei tipi di carattere OpenType, vedere la [specifica OpenType](#).

Estensioni tipografiche avanzate

Le tabelle tipografiche avanzate (tabelle di layout OpenType) estendono le funzionalità dei tipi di carattere con i contorni TrueType o CFF. I tipi di carattere del layout OpenType contengono informazioni aggiuntive che estendono le funzionalità dei tipi di carattere per supportare la tipografia internazionale di alta qualità. La maggior parte dei tipi di carattere OpenType espone solo un subset delle funzionalità OpenType totali disponibili. I tipi di carattere OpenType forniscono le funzionalità seguenti.

- Mapping avanzato tra caratteri e glifi che supportano legature, formati posizionali, glifi alternativi e altre sostituzioni di tipi di carattere.
- Supporto per posizionamento bidimensionale e collegamento di glifi.
- Informazioni esplicite su script e lingua contenute in un carattere, per permettere a un'applicazione di elaborazione del testo di modificare il proprio comportamento di conseguenza.

Le tabelle di layout OpenType sono descritte più dettagliatamente nella sezione "[tabelle di file dei tipi di carattere](#)" della specifica OpenType.

Il resto di questa panoramica introduce l'ampiezza e la flessibilità di alcune delle funzionalità OpenType

visivamente interessanti esposte dalle proprietà dell'oggetto [Typography](#). Per altre informazioni su questo oggetto, vedere [Classe Typography](#).

Varianti

Le varianti vengono usate per eseguire il rendering di stili tipografici diversi, come gli apici e i pedici.

Apici e pedici

La proprietà [Variants](#) consente di impostare valori di apice e pedice per un tipo di carattere OpenType.

Il testo seguente mostra apici per il tipo di carattere Palatino Linotype.

2³ 14th

L'esempio di markup seguente mostra come definire apici per il tipo di carattere Palatino Linotype, usando le proprietà dell'oggetto [Typography](#).

```
<Paragraph FontFamily="Palatino Linotype">
  2<Run Typography.Variants="Superscript">3</Run>
  14<Run Typography.Variants="Superscript">th</Run>
</Paragraph>
```

Il testo seguente mostra pedici per il tipo di carattere Palatino Linotype.

H₂O Footnote₄

L'esempio di markup seguente mostra come definire pedici per il tipo di carattere Palatino Linotype, usando le proprietà dell'oggetto [Typography](#).

```
<Paragraph FontFamily="Palatino Linotype">
  H<Run Typography.Variants="Subscript">2</Run>O
  Footnote<Run Typography.Variants="Subscript">4</Run>
</Paragraph>
```

Uso decorativo di apici e pedici

È anche possibile usare apici e pedici per creare effetti decorativi di testo con maiuscole e minuscole miste. Il testo seguente mostra testo in apice e pedice per il tipo di carattere Palatino Linotype. Si noti che questa formattazione non influisce sulle maiuscole.

Chapter One

Chapter One

L'esempio di markup seguente mostra come definire apici e pedici per un tipo di carattere, usando le proprietà dell'oggetto [Typography](#).

```
<Paragraph FontFamily="Palatino Linotype" Typography.Variants="Superscript">
  Chapter One
</Paragraph>
<Paragraph FontFamily="Palatino Linotype" Typography.Variants="Subscript">
  Chapter One
</Paragraph>
```

Caratteri maiuscoli

I caratteri maiuscoli sono un set di caratteri tipografici per il rendering del testo in glifi in stile maiuscolo. In genere, quando il testo viene visualizzato come tutto maiuscolo, la spaziatura tra le lettere può risultare insufficiente, mentre lo spessore e le proporzioni delle lettere possono apparire eccessivi. OpenType supporta diversi formati di stile per i caratteri maiuscoli, tra cui maiuscole e minuscole, maiuscole e minuscole, titoli e spaziatura dei capitali. Questi formati permettono di controllare l'aspetto dei caratteri maiuscoli.

Il testo seguente visualizza lettere maiuscole standard per il tipo di carattere Pescadero, seguite da lettere in stile "SmallCaps" e "AllSmallCaps". In questo caso, viene usata la stessa dimensione di carattere per tutte e tre le parole.

CAPITALS CAPITALS CAPITALS

L'esempio di markup seguente mostra come definire i caratteri maiuscoli per il tipo di carattere Pescadero usando le proprietà dell'oggetto [Typography](#). Quando si usa il formato "SmallCaps", tutte le iniziali maiuscole vengono ignorate.

```
<Paragraph FontFamily="Pescadero" FontSize="48">
  <Run>CAPITALS</Run>
  <Run Typography.Capitals="SmallCaps">Capitals</Run>
  <Run Typography.Capitals="AllSmallCaps">Capitals</Run>
</Paragraph>
```

Caratteri maiuscoli di titolazione

I caratteri maiuscoli di titolazione hanno spessore e proporzioni minori e sono progettati per conferire un aspetto più elegante rispetto alle lettere maiuscole normali. I caratteri maiuscoli di titolazione vengono in genere usati con dimensioni di carattere maggiori come intestazioni. Il testo seguente mostra caratteri maiuscoli normali e di titolazione per il tipo di carattere Pescadero. Notare la larghezza minore delle aste nel testo della seconda riga.

CHAPTER ONE

CHAPTER ONE

L'esempio di markup seguente mostra come definire i caratteri maiuscoli di titolazione per il tipo di carattere Pescadero usando le proprietà dell'oggetto [Typography](#).

```
<Paragraph FontFamily="Pescadero">
  <Run Typography.Capitals="Titling">chapter one</Run>
</Paragraph>
```

Spaziatura tra caratteri maiuscoli

La spaziatura tra caratteri maiuscoli è una funzionalità che permette di aggiungere più spazio quando si usano caratteri tutti maiuscoli nel testo. Le lettere maiuscole sono in genere progettate per essere combinate con lettere minuscole. La gradevole spaziatura tra una lettera maiuscola e una minuscola può risultare troppo ridotta quando si usano lettere tutte maiuscole. Il testo seguente mostra una spaziatura normale e una tra caratteri maiuscoli per il tipo di carattere Pescadero.

CHAPTER ONE

CHAPTER ONE

Nell'esempio di markup seguente viene illustrato come definire la spaziatura tra maiuscole e minuscole per il tipo di carattere Pescadero, utilizzando le proprietà dell'oggetto [Typography](#).

```
<Paragraph FontFamily="Pescadero">
  <Run Typography.CapitalSpacing="True">CHAPTER ONE</Run>
</Paragraph>
```

Legature

Le legature sono due o più glifi uniti a formare un singolo glifo per creare un testo più leggibile o gradevole. I tipi di carattere OpenType supportano quattro tipi di legature:

- **Legature standard.** Progettate per migliorare la leggibilità. Le legature standard includono "fi", "fl" e "ff".
- **Legature contestuali.** Progettate per migliorare la leggibilità applicando un accostamento migliore tra i caratteri che costituiscono la legatura.
- **Legature discrezionali.** Progettate per scopi ornamentali e non appositamente ideate per favorire la leggibilità.
- **Legature storiche.** Progettate per testi di tipo storico e non appositamente ideate per favorire la leggibilità.

Il testo seguente mostra glifi con legature standard per il tipo di carattere Pericles.

FI FL TH TT TV TW TY VT WT YT

L'esempio di markup seguente mostra come definire glifi di legature standard per il tipo di carattere Pericle, usando le proprietà dell'oggetto [Typography](#).

```
<Paragraph FontFamily="Pericles" Typography.StandardLigatures="True">
  <Run Typography.StylisticAlternates="1">FI</Run>
  <Run Typography.StylisticAlternates="1">FL</Run>
  <Run Typography.StylisticAlternates="1">TH</Run>
  <Run Typography.StylisticAlternates="1">TT</Run>
  <Run Typography.StylisticAlternates="1">TV</Run>
  <Run Typography.StylisticAlternates="1">TW</Run>
  <Run Typography.StylisticAlternates="1">TY</Run>
  <Run Typography.StylisticAlternates="1">VT</Run>
  <Run Typography.StylisticAlternates="1">WT</Run>
  <Run Typography.StylisticAlternates="1">YT</Run>
</Paragraph>
```

Il testo seguente mostra glifi con legature discrezionali per il tipo di carattere Pericles.

CO LA LE LI LL LO LU

L'esempio di markup seguente mostra come definire glifi di legature discrezionali per il tipo di carattere Pericle, usando le proprietà dell'oggetto [Typography](#).

```
<Paragraph FontFamily="Pericles" Typography.DiscretionaryLigatures="True">
  <Run Typography.StylisticAlternates="1">CO</Run>
  <Run Typography.StylisticAlternates="1">LA</Run>
  <Run Typography.StylisticAlternates="1">LE</Run>
  <Run Typography.StylisticAlternates="1">LI</Run>
  <Run Typography.StylisticAlternates="1">LL</Run>
  <Run Typography.StylisticAlternates="1">LO</Run>
  <Run Typography.StylisticAlternates="1">LU</Run>
</Paragraph>
```

Per impostazione predefinita, i tipi di carattere OpenType in Windows Presentation Foundation (WPF) abilitano legature standard. Se si usa, ad esempio, il tipo di carattere Palatino Linotype, le legature standard "fi", "ff" e "fl" verranno visualizzate come glifi di caratteri combinati. Si noti che nella coppia di caratteri per ogni legatura standard i caratteri si toccano tra loro.

fi ff fl

Tuttavia, è possibile disabilitare le funzionalità di legatura standard in modo che una legatura standard come "ff" venga visualizzata come due glifi separati invece che come glifo di caratteri combinati.

fi ff fl

Nell'esempio di markup seguente viene illustrato come disabilitare i glifi di legature standard per il tipo di carattere Palatino Linotype, usando le proprietà dell'oggetto [Typography](#).

```
<!-- Set standard ligatures to false in order to disable feature. -->
<Paragraph Typography.StandardLigatures="False" FontFamily="Palatino Linotype" FontSize="72">
  fi ff fl
</Paragraph>
```

Glifi ornati

I glifi ornati sono glifi decorativi che usano ornamenti elaborati spesso associati alla calligrafia. Il testo seguente mostra glifi standard e ornati per il tipo di carattere Pescadero.

A B C D E F G H I J K L M N
AB CDEF GHIJKLMN

I glifi ornati vengono spesso usati come elementi decorativi in brevi frasi come gli annunci di eventi. Il testo seguente usa glifi ornati per porre in risalto le lettere maiuscole del nome dell'evento.

**Wishing you a
Happy New Year!**

L'esempio di markup seguente mostra come definire ornati per un tipo di carattere, usando le proprietà dell'oggetto [Typography](#).

```
<Paragraph FontFamily="Pescadero" TextBlock.TextAlignment="Center">
    Wishing you a<LineBreak/>
    <Run Typography.StandardSwashes="1" FontSize="36">Happy New Year!</Run>
</Paragraph>
```

Glifi ornati contestuali

Alcune combinazioni di glifi ornati possono creare un aspetto poco gradevole, ad esempio con tratti discendenti che si sovrappongono nelle lettere adiacenti. L'uso di un glifo ornato contestuale permette di usare un glifo ornato sostitutivo che conferisce un aspetto migliore. Il testo seguente mostra la stessa parola prima e dopo l'applicazione di un glifo ornato contestuale.

Lyon Lyon

L'esempio di markup seguente mostra come definire un oggetto ornato contestuale per il tipo di carattere Pescadero usando le proprietà dell'oggetto [Typography](#).

```
<Paragraph FontFamily="Pescadero" Typography.StandardSwashes="1">
    Lyon <Run Typography.ContextualSwashes="1">L</Run>yon
</Paragraph>
```

Glifi alternativi

I glifi alternativi possono essere usati per sostituire un glifo standard. I tipi di carattere OpenType, ad esempio il tipo di carattere Pericle usato negli esempi seguenti, possono contenere glifi alternativi che è possibile usare per creare aspetti diversi per il testo. Il testo seguente mostra i glifi standard per il tipo di carattere Pericles.

ANCIENT GREEK MYTHOLOGY

Il tipo di carattere OpenType di Pericle contiene glifi aggiuntivi che forniscono alternative stilistiche al set di glifi standard. Il testo seguente mostra glifi con stile alternativo.

ANCIENT GREEK MYTHOLOGY

L'esempio di markup seguente mostra come definire glifi alternativi stilistici per il tipo di carattere Pericle, usando le proprietà dell'oggetto [Typography](#).

```
<Paragraph FontFamily="Pericles">
    <Run Typography.StylisticAlternates="1">A</Run>NCIENT
    GR<Run Typography.StylisticAlternates="1">EE</Run>K
    MYTH<Run Typography.StylisticAlternates="1">O</Run>LOGY
</Paragraph>
```

Il testo seguente mostra altri glifi con stile alternativo per il tipo di carattere Pericles.

Α Α Ά Έ Κ Ο Ρ Σ Υ

L'esempio di markup seguente mostra come definire questi altri glifi con stile alternativo.

```
<Paragraph FontFamily="Pericles">
  <Run Typography.StylisticAlternates="1">A</Run>
  <Run Typography.StylisticAlternates="2">A</Run>
  <Run Typography.StylisticAlternates="3">A</Run>
  <Run Typography.StylisticAlternates="1">C</Run>
  <Run Typography.StylisticAlternates="1">E</Run>
  <Run Typography.StylisticAlternates="1">G</Run>
  <Run Typography.StylisticAlternates="1">O</Run>
  <Run Typography.StylisticAlternates="1">Q</Run>
  <Run Typography.StylisticAlternates="1">R</Run>
  <Run Typography.StylisticAlternates="2">R</Run>
  <Run Typography.StylisticAlternates="1">S</Run>
  <Run Typography.StylisticAlternates="1">Y</Run>
</Paragraph>
```

Alternative contestuali casuali

Le alternative contestuali casuali forniscono più glifi sostitutivi per un singolo carattere. Se implementate con tipi di carattere di tipo script, questa funzionalità può simulare la scrittura manuale tramite un set di glifi scelti casualmente con differenze di aspetto minime. Il testo seguente usa alternative contestuali casuali per il tipo di carattere Lindsey. Notare come la lettera "a" varia leggermente nell'aspetto.

a banana in a cabana

L'esempio di markup seguente mostra come definire alternative contestuali casuali per il tipo di carattere Lindsey, usando le proprietà dell'oggetto [Typography](#).

```
<TextBlock FontFamily="Lindsey">
  <Run Typography.ContextualAlternates="True">
    a banana in a cabana
  </Run>
</TextBlock>
```

Formati di tipo storico

I formati di tipo storico sono convenzioni tipografiche usate comunemente in passato. Il testo seguente mostra la frase "Boston, Massachusetts" usando glifi con un formato di tipo storico per il tipo di carattere Palatino Linotype.

Boston, Maffachufettf

L'esempio di markup seguente mostra come definire i form cronologici per il tipo di carattere Palatino Linotype, usando le proprietà dell'oggetto [Typography](#).

```
<Paragraph FontFamily="Palatino Linotype">
  <Run Typography.HistoricalForms="True">Boston, Massachusetts</Run>
</Paragraph>
```

Stili numerici

I tipi di carattere OpenType supportano numerose funzionalità che possono essere usate con i valori numerici nel testo.

Frazioni

I tipi di carattere OpenType supportano gli stili per le frazioni, incluse le barre e le impilazioni.

Il testo seguente mostra stili di frazione per il tipo di carattere Palatino Linotype.

1/8 1/4 3/8 1/2 5/8 3/4 7/8

$\frac{1}{8}$ $\frac{1}{4}$ $\frac{3}{8}$ $\frac{1}{2}$ $\frac{5}{8}$ $\frac{3}{4}$ $\frac{7}{8}$

L'esempio di markup seguente mostra come definire gli stili di frazione per il tipo di carattere Palatino Linotype, usando le proprietà dell'oggetto [Typography](#).

```
<Paragraph FontFamily="Palatino Linotype" Typography.Fraction="Slashed">
  1/8 1/4 3/8 1/2 5/8 3/4 7/8
</Paragraph>
<Paragraph FontFamily="Palatino Linotype" Typography.Fraction="Stacked">
  1/8 1/4 3/8 1/2 5/8 3/4 7/8
</Paragraph>
```

Caratteri numerici in stile antico

I tipi di carattere OpenType supportano un formato numerico obsoleto. Questo formato è utile per visualizzare valori numerici in stili che non sono più quelli standard. Il testo seguente mostra una data del 18° secolo con formati numerici standard e in stile antico per il tipo di carattere Palatino Linotype.

July 4, 1776 July 4, 1776

Il testo seguente mostra caratteri numerici standard per il tipo di carattere Palatino Linotype, seguiti da caratteri numerici in stile antico.

1234567890 1234567890

L'esempio di markup seguente mostra come definire i numeri di stile obsoleti per il tipo di carattere Palatino Linotype, usando le proprietà dell'oggetto [Typography](#).

```
<Paragraph FontFamily="Palatino Linotype">
  <Run Typography.NumeralStyle="Normal">1234567890</Run>
  <Run Typography.NumeralStyle="OldStyle">1234567890</Run>
</Paragraph>
```

Cifre proporzionali e tabulari

I tipi di carattere OpenType supportano una funzionalità di figura proporzionale e tabulare per controllare l'allineamento delle larghezze quando si usano i numeri. Le cifre proporzionali gestiscono ogni carattere numerico applicando una larghezza diversa: "1" è più stretto di "5". Le cifre tabulari vengono gestite come caratteri numerici di uguale larghezza in modo da poter essere allineate in verticale, per aumentare la leggibilità delle informazioni di tipo finanziario.

Il testo seguente mostra due cifre proporzionali nella prima colonna usando il tipo di carattere Miramonte. Notare la differenza in larghezza tra i caratteri numerici "5" e "1". La seconda colonna mostra gli stessi valori numerici, le cui larghezze sono state modificate tramite la funzionalità per le cifre tabulari.

550,689 550,689
114,131 114,131

L'esempio di markup seguente mostra come definire le cifre proporzionali e tabulari per il tipo di carattere Miramonte usando le proprietà dell'oggetto [Typography](#).

```
<TextBlock FontFamily="Miramonte">
  <Run Typography.NumeralAlignment="Proportional">114,131</Run>
</TextBlock>
<TextBlock FontFamily="Miramonte">
  <Run Typography.NumeralAlignment="Tabular">114,131</Run>
</TextBlock>
```

Zero barrato

I tipi di carattere OpenType supportano un formato numerico con zero barrato per evidenziare la differenza tra la lettera "O" e il numero "0". Il carattere numerico zero barrato viene spesso usato per gli identificatori all'interno di informazioni finanziarie e aziendali.

Il testo seguente mostra un identificatore di ordine di esempio con il tipo di carattere Miramonte. La prima riga usa caratteri numerici standard. La seconda riga usa caratteri numerici con zero barrato per distinguere meglio il carattere numerico dalla lettera "O" maiuscola.

Order #0048-OTC-390

Order #0048-OTC-390

Nell'esempio di markup seguente viene illustrato come definire i numeri con zero barrato per il tipo di carattere Miramonte, utilizzando le proprietà dell'oggetto [Typography](#).

```
<Paragraph FontFamily="Miramonte">
  <Run>Order #0048-OTC-390</Run>
  <LineBreak/>
  <Run Typography.SlashedZero="True">Order #0048-OTC-390</Run>
</Paragraph>
```

Classe Typography

L'oggetto [Typography](#) espone il set di funzionalità supportate da un tipo di carattere OpenType. Impostando le proprietà di [Typography](#) nel markup, è possibile creare facilmente documenti che sfruttano le funzionalità OpenType.

Il testo seguente visualizza lettere maiuscole standard per il tipo di carattere Pescadero, seguite da lettere in stile "SmallCaps" e "AllSmallCaps". In questo caso, viene usata la stessa dimensione di carattere per tutte e tre le parole.

CAPITALS CAPITALS CAPITALS

L'esempio di markup seguente mostra come definire i caratteri maiuscoli per il tipo di carattere Pescadero usando le proprietà dell'oggetto [Typography](#). Quando si usa il formato "SmallCaps", tutte le iniziali maiuscole vengono ignorate.

```
<Paragraph FontFamily="Pescadero" FontSize="48">
  <Run>CAPITALS</Run>
  <Run Typography.Capitals="SmallCaps">Capitals</Run>
  <Run Typography.Capitals="AllSmallCaps">Capitals</Run>
</Paragraph>
```

L'esempio di codice seguente esegue la stessa attività dell'esempio di markup precedente.

```

MyParagraph.FontFamily = new FontFamily("Pescadero");
MyParagraphFontSize = 48;

Run run_1 = new Run("CAPITALS ");
MyParagraph.Inlines.Add(run_1);

Run run_2 = new Run("Capitals ");
run_2.Typography.Capitals = FontCapitals.SmallCaps;
MyParagraph.Inlines.Add(run_2);

Run run_3 = new Run("Capitals");
run_3.Typography.Capitals = FontCapitals.AllSmallCaps;
MyParagraph.Inlines.Add(run_3);

MyParagraph.Inlines.Add(new LineBreak());

```

```

MyParagraph.FontFamily = New FontFamily("Pescadero")
MyParagraph.FontSize = 48

Dim run_1 As New Run("CAPITALS ")
MyParagraph.Inlines.Add(run_1)

Dim run_2 As New Run("Capitals ")
run_2.Typography.Capitals = FontCapitals.SmallCaps
MyParagraph.Inlines.Add(run_2)

Dim run_3 As New Run("Capitals")
run_3.Typography.Capitals = FontCapitals.AllSmallCaps
MyParagraph.Inlines.Add(run_3)

MyParagraph.Inlines.Add(New LineBreak())

```

Proprietà della classe **Typography**

Nella tabella seguente sono elencate le proprietà, i valori e le impostazioni predefinite dell'oggetto **Typography**.

PROPRIETÀ	VALORE/I	DEFAULT VALUE
AnnotationAlternates	Valore numerico, byte	0
Capitals	AllPetiteCaps AllSmallCaps Normal PetiteCaps SmallCaps Titling Unicase	FontCapitals.Normal
CapitalSpacing	Boolean	false
CaseSensitiveForms	Boolean	false
ContextualAlternates	Boolean	true
ContextualLigatures	Boolean	true
ContextualSwashes	Valore numerico, byte	0
DiscretionaryLigatures	Boolean	false
EastAsianExpertForms	Boolean	false

PROPRIETÀ	VALORE/I	DEFAULT VALUE
EastAsianLanguage	HojoKanji Jis04 Jis78 Jis83 Jis90 NlcKanji Simplified Traditional TraditionalNamesNormal	FontEastAsianLanguage.Normal
EastAsianWidths	Full Half Normal Proportional Quarter Third	FontEastAsianWidths.Normal
Fraction	Normal Slashed Stacked	FontFraction.Normal
HistoricalForms	Boolean	false
HistoricalLigatures	Boolean	false
Kerning	Boolean	true
MathematicalGreek	Boolean	false
NumeralAlignment	Normal Proportional Tabular	FontNumeralAlignment.Normal
NumeralStyle	Boolean	FontNumeralStyle.Normal
SlashedZero	Boolean	false
StandardLigatures	Boolean	true
StandardSwashes	Valore numerico, byte	0
StylisticAlternates	Valore numerico, byte	0
StylisticSet1	Boolean	false
StylisticSet2	Boolean	false
StylisticSet3	Boolean	false
StylisticSet4	Boolean	false
StylisticSet5	Boolean	false
StylisticSet6	Boolean	false
StylisticSet7	Boolean	false
StylisticSet8	Boolean	false
StylisticSet9	Boolean	false
StylisticSet10	Boolean	false

PROPRIETÀ	VALORE/I	DEFAULT VALUE
StylisticSet11	Boolean	<code>false</code>
StylisticSet12	Boolean	<code>false</code>
StylisticSet13	Boolean	<code>false</code>
StylisticSet14	Boolean	<code>false</code>
StylisticSet15	Boolean	<code>false</code>
StylisticSet16	Boolean	<code>false</code>
StylisticSet17	Boolean	<code>false</code>
StylisticSet18	Boolean	<code>false</code>
StylisticSet19	Boolean	<code>false</code>
StylisticSet20	Boolean	<code>false</code>
Variants	Inferior Normal Ordinal Ruby Subscript Superscript	FontVariants.Normal

Vedere anche

- [Typography](#)
- [Specifica OpenType](#)
- [Funzionalità tipografiche di WPF](#)
- [Esempio di pacchetto di tipi di carattere OpenType](#)
- [Includere i tipi di carattere nel pacchetto delle applicazioni](#)

Includere i tipi di carattere nel pacchetto delle applicazioni

23/10/2019 • 14 minutes to read • [Edit Online](#)

In questo argomento viene fornita una panoramica su come creare un pacchetto di tipi di carattere con l'applicazione Windows Presentation Foundation (WPF).

NOTE

Come con la maggior parte delle applicazioni software, i file dei tipi di carattere vengono concessi in licenza e non venduti. Le licenze che regolano l'utilizzo dei tipi di carattere variano a seconda del fornitore, ma in generale la maggior parte delle licenze, incluse quelle relative ai tipi di carattere forniti da Microsoft con le applicazioni e Windows, non consentono di incorporare i tipi di carattere all'interno delle applicazioni o di ridistribuirli. Per questo motivo, gli sviluppatori sono tenuti ad assicurarsi che l'utente abbia i diritti di licenza separati per qualsiasi tipo di carattere incorporato in un'applicazione o ridistribuito in altro modo.

Introduzione all'inclusione dei tipi di carattere nel pacchetto

È possibile creare facilmente un pacchetto di tipi di carattere come risorse all'interno delle applicazioni WPF per visualizzare il testo dell'interfaccia utente e altri tipi di contenuto basato su testo. I tipi di carattere possono essere separati o incorporati nei file di assembly dell'applicazione. È anche possibile creare una libreria di tipi di carattere di sole risorse, a cui l'applicazione può fare riferimento.

I tipi di carattere OpenType e TrueType® contengono un flag di tipo, `fsType`, che indica i diritti di licenza per l'incorporamento del tipo di carattere. Questo flag di tipo, tuttavia, fa riferimento solo ai tipi di carattere incorporati archiviati in un documento e non ai tipi di carattere incorporati in un'applicazione. È possibile recuperare i diritti di incorporamento del tipo di carattere per un tipo di carattere creando un oggetto `Glyph Typeface` e facendo riferimento alla relativa proprietà `EmbeddingRights`. Per ulteriori informazioni sul flag `fsType`, vedere la sezione "metriche del sistema operativo/2 e Windows" della [specifica OpenType](#).

Il sito Web [Microsoft Typography](#) include informazioni di contatto che consentono di individuare un particolare fornitore di tipi di carattere o trovare un fornitore di tipi di carattere per il lavoro personalizzato.

Aggiunta di tipi di carattere come elementi di contenuto

È possibile aggiungere tipi di carattere all'applicazione come elementi di contenuto del progetto separati dai file di assembly dell'applicazione. Gli elementi di contenuto non vengono quindi incorporati come risorse in un assembly. L'esempio di file di progetto seguente illustra come definire gli elementi di contenuto.

```
<Project DefaultTargets="Build"
      xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <!-- Other project build settings ... -->

  <ItemGroup>
    <Content Include="Peric.ttf" />
    <Content Include="Peric1.ttf" />
  </ItemGroup>
</Project>
```

Per assicurarsi che l'applicazione possa usare i tipi di carattere in fase di esecuzione, i tipi di carattere devono

essere accessibili nella directory di distribuzione dell'applicazione. L'elemento `<CopyToOutputDirectory>` nel file di progetto dell'applicazione consente di copiare automaticamente i tipi di carattere nella directory di distribuzione dell'applicazione durante il processo di compilazione. L'esempio di file di progetto seguente illustra come copiare i tipi di carattere nella directory di distribuzione.

```
<ItemGroup>
  <Content Include="Peric.ttf">
    <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
  </Content>
  <Content Include="Peric1.ttf">
    <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
  </Content>
</ItemGroup>
```

L'esempio di codice seguente illustra come fare riferimento al tipo di carattere dell'applicazione come elemento di contenuto. L'elemento di contenuto a cui si fa riferimento deve essere nella stessa directory dei file di assembly dell'applicazione.

```
<TextBlock FontFamily="./#Pericles Light">
  Aegean Sea
</TextBlock>
```

Aggiunta di tipi di carattere come elementi risorsa

È possibile aggiungere tipi di carattere all'applicazione come elementi risorsa del progetto incorporati nei file di assembly dell'applicazione. L'uso di una sottodirectory separata per le risorse consente di organizzare i file di progetto dell'applicazione. L'esempio di file di progetto seguente illustra come definire i tipi di carattere come elementi risorsa in una sottodirectory separata.

```
<Project DefaultTargets="Build"
  xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <!-- Other project build settings ... -->

  <ItemGroup>
    <Resource Include="resources\Peric.ttf" />
    <Resource Include="resources\Peric1.ttf" />
  </ItemGroup>
</Project>
```

NOTE

Quando si aggiungono tipi di carattere come risorse all'applicazione, assicurarsi di impostare l'elemento `<Resource>` e non l'elemento `<EmbeddedResource>` nel file di progetto dell'applicazione. L'elemento `<EmbeddedResource>` per l'azione di compilazione non è supportato.

L'esempio di markup seguente illustra come fare riferimento alle risorse dei tipi di carattere dell'applicazione.

```
<TextBlock FontFamily=".//resources/#Pericles Light">
  Aegean Sea
</TextBlock>
```

Riferimento agli elementi risorsa dei tipi di carattere dal codice

Per fare riferimento agli elementi delle risorse del tipo di carattere dal codice, è necessario fornire un riferimento a una risorsa del tipo di carattere in due parti: URI (Uniform Resource Identifier) di base. e il riferimento al percorso

del tipo di carattere. Questi valori vengono usati come parametri per il metodo `FontFamily`. Nell'esempio di codice seguente viene illustrato come fare riferimento alle risorse dei tipi di carattere dell'applicazione nella sottodirectory del progetto denominata `resources`.

```
// The font resource reference includes the base URI reference (application directory level),  
// and a relative URI reference.  
myTextBlock.FontFamily = new FontFamily(new Uri("pack://application:,,,/"), "./resources/#Pericles Light");
```

```
' The font resource reference includes the base URI reference (application directory level),  
' and a relative URI reference.  
myTextBlock.FontFamily = New FontFamily(New Uri("pack://application:,,,/"), "./resources/#Pericles Light")
```

L'URI (Uniform Resource Identifier) di base può includere la sottodirectory dell'applicazione in cui risiede la risorsa del tipo di carattere. In questo caso, non è necessario specificare una directory per il riferimento al percorso dei tipi di carattere, ma è necessario includere un "`./`" leader, che indica che la risorsa del tipo di carattere si trova nella stessa directory specificata dall'URI (Uniform Resource Identifier) di base. L'esempio di codice seguente illustra un modo alternativo per fare riferimento all'elemento risorsa dei tipi di carattere, che equivale all'esempio di codice precedente.

```
// The base URI reference can include an application subdirectory.  
myTextBlock.FontFamily = new FontFamily(new Uri("pack://application:,,,/resources/"), "./#Pericles Light");
```

```
' The base URI reference can include an application subdirectory.  
myTextBlock.FontFamily = New FontFamily(New Uri("pack://application:,,,/resources/"), "./#Pericles Light")
```

Riferimento ai tipi di carattere dalla stessa sottodirectory dell'applicazione

È possibile inserire sia il contenuto dell'applicazione che i file di risorse nella stessa sottodirectory definita dall'utente del progetto dell'applicazione. L'esempio di file di progetto seguente illustra una pagina contenuto e le risorse dei tipi di carattere definite nella stessa sottodirectory.

```
<ItemGroup>  
  <Page Include="pages\HomePage.xaml" />  
</ItemGroup>  
<ItemGroup>  
  <Resource Include="pages\Peric.ttf" />  
  <Resource Include="pages\Peric1.ttf" />  
</ItemGroup>
```

Poiché il tipo di carattere e il contenuto dell'applicazione sono nella stessa sottodirectory, il riferimento al tipo di carattere è relativo al contenuto dell'applicazione. Gli esempi seguenti illustrano come fare riferimento alla risorsa del tipo di carattere dell'applicazione quando il tipo di carattere è nella stessa directory dell'applicazione.

```
<TextBlock FontFamily="./#Pericles Light">  
  Aegean Sea  
</TextBlock>
```

```
// The font resource reference includes the base Uri (application directory level),  
// and the file resource location, which is relative to the base Uri.  
myTextBlock.FontFamily = new FontFamily(new Uri("pack://application:,,,/"), "/pages/#Pericles Light");
```

```
' The font resource reference includes the base Uri (application directory level),  
' and the file resource location, which is relative to the base Uri.  
myTextBlock.FontFamily = New FontFamily(New Uri("pack://application:,,,/"), "/pages/#Pericles Light")
```

Enumerazione di tipi di carattere in un'applicazione

Per enumerare i tipi di carattere come elementi risorsa nell'applicazione, usare il metodo [GetFontFamilies](#) o [GetTypefaces](#). Nell'esempio seguente viene illustrato come utilizzare il metodo [GetFontFamilies](#) per restituire la raccolta di oggetti [FontFamily](#) dal percorso dei tipi di carattere dell'applicazione. In questo caso, l'applicazione contiene una sottodirectory denominata "resources".

```
foreach (FontFamily fontFamily in Fonts.GetFontFamilies(new Uri("pack://application:,,,/"), "./resources/"))  
{  
    // Perform action.  
}
```

```
For Each fontFamily As FontFamily In Fonts.GetFontFamilies(New Uri("pack://application:,,,/"), "./resources/")  
    ' Perform action.  
Next fontFamily
```

Nell'esempio seguente viene illustrato come utilizzare il metodo [GetTypefaces](#) per restituire la raccolta di oggetti [Typeface](#) dal percorso dei tipi di carattere dell'applicazione. In questo caso, l'applicazione contiene una sottodirectory denominata "resources".

```
foreach (Typeface typeface in Fonts.GetTypefaces(new Uri("pack://application:,,,/"), "./resources/"))  
{  
    // Perform action.  
}
```

```
For Each typeface As Typeface In Fonts.GetTypefaces(New Uri("pack://application:,,,/"), "./resources/")  
    ' Perform action.  
Next typeface
```

Creazione di una libreria di risorse tipo di carattere

È possibile creare una libreria di sole risorse contenente solo tipi di carattere. Nessun codice fa parte di questo tipo di progetto libreria. La creazione di una libreria di sole risorse è una tecnica comune per disaccoppiare le risorse dal codice dell'applicazione che le usa e consente anche di includere l'assembly di librerie in più progetti di applicazione. L'esempio di file di progetto seguente illustra le parti chiave di un progetto di libreria di sole risorse.

```
<PropertyGroup>  
    <AssemblyName>FontLibrary</AssemblyName>  
    <OutputType>library</OutputType>  
    ...  
</PropertyGroup>  
...  
<ItemGroup>  
    <Resource Include="Kooten.ttf" />  
    <Resource Include="Pesca.ttf" />  
</ItemGroup>
```

Riferimento a un tipo di carattere in una libreria di risorse

Per fare riferimento a un tipo di carattere in una libreria di risorse dall'applicazione, è necessario far precedere il

riferimento al tipo di carattere dal nome dell'assembly di librerie. In questo caso, l'assembly di risorse del tipo di carattere è "FontLibrary". Per separare il nome dell'assembly dal riferimento nell'assembly, usare un carattere ":". Aggiungendo la parola chiave "Component" seguita dal riferimento al nome del tipo di carattere, si completa l'intero riferimento alla risorsa della libreria di tipi di carattere. L'esempio di codice seguente illustra come fare riferimento a un tipo di carattere in un assembly di librerie di risorse.

```
<Run FontFamily="/FontLibrary;Component/#Kootenay" FontSize="36">  
    ABCDEFGHIJKLMNOPQRSTUVWXYZ  
</Run>
```

NOTE

Questo SDK contiene un set di tipi di carattere OpenType di esempio che è possibile usare con WPF applicazioni. I tipi di carattere sono definiti in una libreria di sole risorse. Per altre informazioni, vedere [Esempio di pacchetto di tipi di carattere OpenType](#).

Limitazioni all'utilizzo dei tipi di carattere

Nell'elenco seguente vengono descritte diverse limitazioni relative alla creazione di pacchetti e all'uso di tipi di carattere nelle applicazioni WPF:

- **Bit autorizzazione per l'incorporamento tipi di carattere:** WPF le applicazioni non controllano né applicano bit di autorizzazione per l'incorporamento tipi di carattere. Per ulteriori informazioni, vedere la sezione relativa ai [tipi di carattere Introduction_to_Packing](#).
- **Tipi di carattere del sito di origine:** le applicazioni WPF non consentono un riferimento a un tipo di carattere a un URI (Uniform Resource Identifier) http o FTP.
- **URI assoluto che utilizza la notazione Pack:** WPF le applicazioni non consentono di creare un oggetto `FontFamily` a livello di codice utilizzando "Pack:" come parte del riferimento URI (Uniform Resource Identifier) assoluto a un tipo di carattere. Ad esempio, `"pack://application:,,,/resources/#Pericles_Light"` è un riferimento al tipo di carattere non valido.
- **Incorporamento tipi di carattere automatico:** durante la fase di progettazione, non è disponibile il supporto per cercare l'uso dei tipi di carattere di un'applicazione e incorporare automaticamente i tipi di carattere nelle risorse dell'applicazione.
- **Subset di tipi di carattere:** le applicazioni WPF non supportano la creazione di subset di tipi di carattere per documenti non statici.
- Se è presente un riferimento non corretto, l'applicazione torna a usare un tipo di carattere disponibile.

Vedere anche

- [Typography](#)
- [FontFamily](#)
- [Microsoft Typography: collegamenti, notizie e contatti](#)
- [Specifica OpenType](#)
- [Funzionalità dei tipi di carattere OpenType](#)
- [Esempio di pacchetto di tipi di carattere OpenType](#)

Esempio di pacchetto di tipi di carattere OpenType

10/02/2020 • 3 minutes to read • [Edit Online](#)

In questo argomento viene fornita una panoramica dei tipi di carattere OpenType di esempio distribuiti con la Windows SDK. I tipi di carattere di esempio supportano le funzionalità OpenType estese che possono essere utilizzate dalle applicazioni Windows Presentation Foundation (WPF).

Tipi di carattere nel pacchetto di caratteri OpenType

Il Windows SDK fornisce un set di tipi di carattere OpenType di esempio che è possibile utilizzare per la creazione di applicazioni Windows Presentation Foundation (WPF). I tipi di carattere di esempio sono concessi in licenza da Ascender Corporation. Questi tipi di carattere implementano solo un subset delle funzionalità totali definite dal formato OpenType. Nella tabella seguente sono elencati i nomi dei tipi di carattere OpenType di esempio.

NOME	FILE
Kootenay	Kooten.ttf
Lindsey	Linds.ttf
Miramonte	Miramo.ttf
Miramonte Bold	Miramob.ttf
Pericles	Peric.ttf
Pericles Light	Pericl.ttf
Pescadero	Pesca.ttf
Pescadero Bold	Pescab.ttf

La figura seguente mostra l'aspetto dei tipi di carattere OpenType di esempio.

Kootenay

Lindsey

Miramonte

Miramonte Bold

PERICLES

PERICLES LIGHT

Pescadero

Pescadero Bold

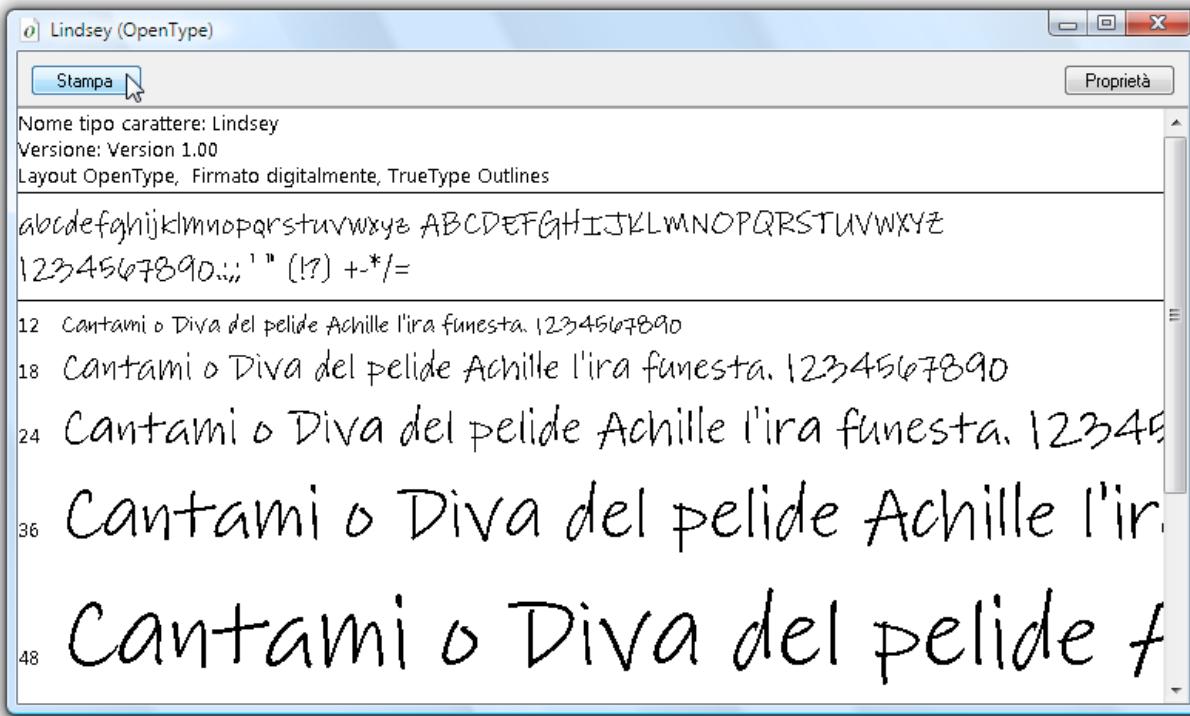
I tipi di carattere di esempio sono concessi in licenza da Ascender Corporation. Ascender è un provider di tipi di carattere avanzati. Per ottenere in licenza versioni estese o personalizzate dei tipi di carattere di esempio, vedere il [sito Web di Ascender Corporation](#).

NOTE

È responsabilità degli sviluppatori verificare di disporre dei diritti di licenza necessari per qualsiasi tipo di carattere incorporato all'interno di un'applicazione o ridistribuito in altro modo.

Installazione dei tipi di carattere

È possibile installare i tipi di carattere OpenType di esempio nella directory predefinita dei tipi di carattere di Windows, **\Windows\Fonts**. Per l'installazione, usare il pannello di controllo Tipi di carattere. Una volta che questi tipi di carattere si trovano nel computer, sono accessibili a tutte le applicazioni che fanno riferimento ai tipi di carattere predefiniti di Windows. È possibile visualizzare un set di caratteri rappresentativo in diverse dimensioni facendo doppio clic sul file del tipo di carattere. La screenshot seguente mostra il file del tipo di carattere Lindsey, ovvero Linds.ttf.



Visualizzazione del tipo di carattere Lindsey

Uso dei tipi di carattere

Nell'applicazione si possono usare i tipi di carattere in due modi. È possibile aggiungere tipi di carattere all'applicazione come elementi di contenuto del progetto non incorporati come risorse in un assembly. In alternativa, si possono aggiungere tipi di carattere all'applicazione come elementi risorsa del progetto incorporati nei file di assembly dell'applicazione. Per altre informazioni, vedere [Includere i tipi di carattere nel pacchetto delle applicazioni](#).

Vedere anche

- [Typography](#)
- [Funzionalità dei tipi di carattere OpenType](#)
- [Includere i tipi di carattere nel pacchetto delle applicazioni](#)

Procedure relative ai tipi di caratteri

23/10/2019 • 2 minutes to read • [Edit Online](#)

Negli argomenti di questa sezione viene illustrato come utilizzare le caratteristiche del carattere incluse con Windows Presentation Foundation (WPF).

In questa sezione

[Enumerare i tipi di carattere del sistema](#)

[Usare la classe FontSizeConverter](#)

Vedere anche

- [FontStyle](#)
- [SystemFonts](#)
- [Documenti in WPF](#)
- [Funzionalità tipografiche di WPF](#)

Procedura: Enumerare i tipi di carattere del sistema

23/10/2019 • 2 minutes to read • [Edit Online](#)

Esempio

Nell'esempio seguente viene illustrato come enumerare i tipi di carattere nella raccolta del tipo di carattere del sistema. Il nome della famiglia della ognuno `FontFamily` all'interno di `SystemFontFamilies` viene aggiunto come un elemento a una casella combinata.

```
public void FillFontComboBox(ComboBox comboBoxFonts)
{
    // Enumerate the current set of system fonts,
    // and fill the combo box with the names of the fonts.
    foreach (FontFamily fontFamily in Fonts.SystemFontFamilies)
    {
        // FontFamily.Source contains the font family name.
        comboBoxFonts.Items.Add(fontFamily.Source);
    }

    comboBoxFonts.SelectedIndex = 0;
}
```

```
Public Sub FillFontComboBox(ByVal comboBoxFonts As ComboBox)
    ' Enumerate the current set of system fonts,
    ' and fill the combo box with the names of the fonts.
    For Each fontFamily As FontFamily In Fonts.SystemFontFamilies
        ' FontFamily.Source contains the font family name.
        comboBoxFonts.Items.Add(fontFamily.Source)
    Next fontFamily

    comboBoxFonts.SelectedIndex = 0
End Sub
```

Se più versioni della stessa famiglia di caratteri si trovano nella stessa directory, il Windows Presentation Foundation (WPF) enumerazione del tipo di carattere restituisce la versione più recente del tipo di carattere. Se le informazioni sulla versione non fornisce la risoluzione, viene restituito il tipo di carattere con timestamp più recente. Se le informazioni relative al timestamp sono equivalenti, viene restituito il file del tipo di carattere che compare per primo in ordine alfabetico.

Procedura: Usare la classe FontSizeConverter

23/10/2019 • 2 minutes to read • [Edit Online](#)

Esempio

In questo esempio viene illustrato come creare un'istanza di [FontSizeConverter](#) e usarlo per modificare una dimensione del carattere.

L'esempio definisce un metodo personalizzato denominato `changeSize` che converte il contenuto di un [ListBoxItem](#), come definito in un oggetto separato Extensible Application Markup Language (XAML) file, a un'istanza di [Double](#) successivamente in un [String](#). Questo metodo passa il [ListBoxItem](#) a un [FontSizeConverter](#) oggetto, che converte le [Content](#) di un [ListBoxItem](#) a un'istanza di [Double](#). Questo valore viene quindi passato nuovamente come valore dei [FontSize](#) proprietà del [TextBlock](#) elemento.

In questo esempio definisce anche un secondo metodo personalizzato che viene chiamato `changeFamily`. Questo metodo converte il [Content](#) del [ListBoxItem](#) per un [String](#) quindi passa tale valore per il [FontFamily](#) proprietà del [TextBlock](#) elemento.

Questo esempio non viene eseguito.

```
private void changeSize(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem li = ((sender as ListBox).SelectedItem as ListBoxItem);
    FontSizeConverter myFontSizeConverter = new FontSizeConverter();
    text1.FontSize = (Double)myFontSizeConverter.ConvertFromString(li.Content.ToString());
}

private void changeFamily(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem li2 = ((sender as ListBox).SelectedItem as ListBoxItem);
    text1.FontFamily = new System.Windows.Media.FontFamily(li2.Content.ToString());
}
```

Vedere anche

- [FontSizeConverter](#)

Glifi

23/10/2019 • 2 minutes to read • [Edit Online](#)

Glifi sono una rappresentazione di basso livello di un carattere da disegnare sullo schermo. Windows Presentation Foundation (WPF) fornisce accesso diretto alle icone per i clienti che vogliono intercettare e salvare in modo permanente il testo dopo la formattazione.

In questa sezione

[Introduzione all'oggetto GlyphRun e all'elemento Glyphs](#)

[Procedura: Creare testo utilizzando glifi](#)

Vedere anche

- [GlyphRun](#)
- [DrawText](#)
- [Glyphs](#)
- [Documenti in WPF](#)
- [Funzionalità tipografiche di WPF](#)

Introduzione all'oggetto GlyphRun e all'elemento Glyphs

10/01/2020 • 6 minutes to read • [Edit Online](#)

In questo argomento vengono descritti l'oggetto [GlyphRun](#) e l'elemento [Glyphs](#).

Introduzione a GlyphRun

Windows Presentation Foundation (WPF) offre supporto avanzato per il testo, incluso il markup a livello di glifo, con accesso diretto ai [Glyphs](#) per i clienti che desiderano intercettare e mantenere il testo dopo la formattazione. Queste funzionalità forniscono il supporto fondamentale per i diversi requisiti di rendering del testo in ognuno degli scenari seguenti.

1. Visualizzazione di documenti a formato fisso.
2. Scenari di stampa.
 - Extensible Application Markup Language (XAML) come linguaggio della stampante.
 - Microsoft XPS Document Writer.
 - Driver della stampante precedenti, output delle applicazioni Win32 nel formato fisso.
 - Formato dello spooling di stampa.
3. Rappresentazione di documenti a formato fisso, inclusi i client per le versioni precedenti di Windows e altri dispositivi di elaborazione.

NOTE

[Glyphs](#) e [GlyphRun](#) sono progettati per scenari di presentazione e stampa di documenti a formato fisso. Windows Presentation Foundation (WPF) fornisce diversi elementi per il layout generale e scenari di interfaccia utente, ad esempio [Label](#) e [TextBlock](#). Per altre informazioni sugli scenari di layout e dell'Interfaccia utente, vedere [Funzionalità tipografiche di WPF](#).

Oggetto GlyphRun

L'oggetto [GlyphRun](#) rappresenta una sequenza di glifi da una singola faccia di un singolo tipo di carattere a una singola dimensione e con un unico stile di rendering.

[GlyphRun](#) include sia i dettagli del tipo di carattere, ad esempio le posizioni del glifo [Indices](#) che le singole icone. Include anche i punti di codice Unicode originali da cui è stata generata l'esecuzione, le informazioni sul mapping dell'offset del buffer da carattere a glifo e i flag per carattere e per glifo.

[GlyphRun](#) dispone di un [FrameworkElement](#) di alto livello corrispondente, [Glyphs](#). [Glyphs](#) possibile utilizzare nell'albero degli elementi e in XAML markup per rappresentare [GlyphRun](#) output.

Elemento Glyphs

L'elemento [Glyphs](#) rappresenta l'output di un [GlyphRun](#) in XAML. Per descrivere l'elemento [Glyphs](#), viene utilizzata la sintassi di markup seguente.

```

<!-- The example shows how to use a Glyphs object. -->
<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    >

    <StackPanel Background="PowderBlue">

        <Glyphs
            FontUri          = "C:\WINDOWS\Fonts\TIMES.TTF"
            FontRenderingEmSize = "100"
            StyleSimulations   = "BoldSimulation"
            UnicodeString     = "Hello World!"
            Fill              = "Black"
            OriginX           = "100"
            OriginY           = "200"
        />

    </StackPanel>
</Page>

```

Le definizioni di proprietà seguenti corrispondono ai primi quattro attributi del markup di esempio.

GLI	DESCRIZIONE
FontUri	Specifica un identificatore di risorsa: il nome file, l'URI (Uniform Resource Identifier) Web o il riferimento a una risorsa nell'applicazione. exe o nel contenitore.
FontRenderingEmSize	Specifica le dimensioni del carattere nelle unità della superficie di disegno (l'impostazione predefinita è 0,96 pollici).
StyleSimulations	Specifica i flag per gli stili grassetto e corsivo.
BidiLevel	Specifica il livello di layout bidirezionale. I valori pari e lo zero implicano un layout da sinistra a destra, mentre i valori dispari implicano un layout da destra a sinistra.

Proprietà Indices

La proprietà [Indices](#) è una stringa di specifiche di glifi. Quando una sequenza di glifi forma un cluster singolo, la specifica del primo glifo nel cluster viene preceduta da una specifica del numero di glifi e di punti di codice combinati per formare il cluster. La proprietà [Indices](#) raccoglie in una stringa le proprietà seguenti.

- Indici del glifo
- Distanze di avanzamento del glifo
- Combinazione dei vettori di connessione dei glifi
- Mapping del cluster tra i punti di codice e i glifi
- Flag del glifo

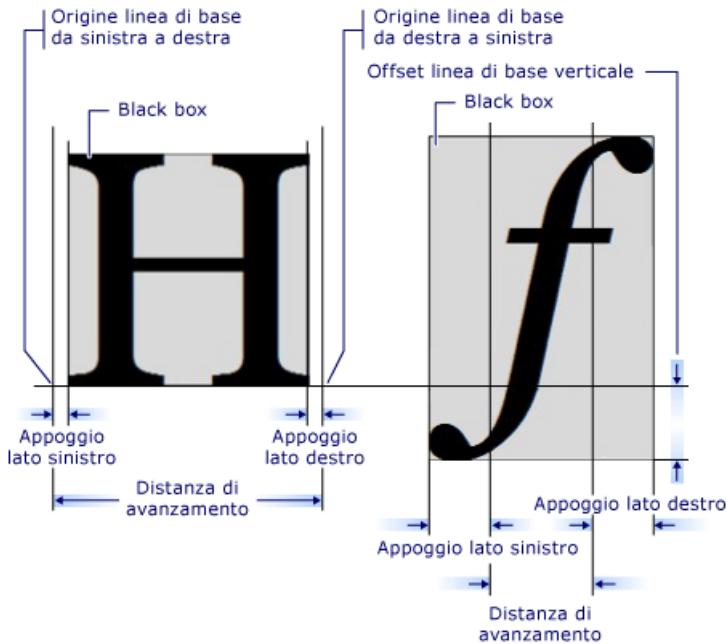
Ogni specifica del glifo presenta la forma seguente:

```
[GlyphIndex][,[Advance][,[uOffset][,[vOffset][,[Flags]]]]]
```

Metrica del glifo

Ogni glifo definisce le metriche che specificano la modalità di allineamento con altre [Glyphs](#). Nell'immagine

seguente vengono definite le varie qualità tipografiche di due diversi caratteri glifi.



Markup dei glifi

Nell'esempio di codice seguente viene illustrato come utilizzare varie proprietà dell'elemento `Glyphs` in XAML.

```
<!-- The example shows how to use different property settings of Glyphs objects. -->
<Canvas
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Background="PowderBlue"
>

<Glyphs
    FontUri          = "C:\WINDOWS\Fonts\ARIAL.TTF"
    FontRenderingEmSize = "36"
    StyleSimulations   = "ItalicSimulation"
    UnicodeString     = "Hello World!"
    Fill              = "SteelBlue"
    OriginX           = "50"
    OriginY           = "75"
/>

<!-- "Hello World!" with default kerning -->
<Glyphs
    FontUri          = "C:\WINDOWS\Fonts\ARIAL.TTF"
    FontRenderingEmSize = "36"
    UnicodeString     = "Hello World!"
    Fill              = "Maroon"
    OriginX           = "50"
    OriginY           = "150"
/>

<!-- "Hello World!" with explicit character widths for proportional font -->
<Glyphs
    FontUri          = "C:\WINDOWS\Fonts\ARIAL.TTF"
    FontRenderingEmSize = "36"
    UnicodeString     = "Hello World!"
    Indices           = ",80;,80;,80;,80;,80;,80;,80;,80;,80;,80"
    Fill              = "Maroon"
    OriginX           = "50"
    OriginY           = "225"
/>

<!-- "Hello World!" with fixed-width font -->
```

```

<Glyphs
    FontUri      = "C:\WINDOWS\Fonts\COUR.TTF"
    FontRenderingEmSize = "36"
    StyleSimulations = "BoldSimulation"
    UnicodeString   = "Hello World!"
    Fill           = "Maroon"
    OriginX        = "50"
    OriginY        = "300"
/>

<!-- "Open file" without "fi" ligature -->
<Glyphs
    FontUri      = "C:\WINDOWS\Fonts\TIMES.TTF"
    FontRenderingEmSize = "36"
    StyleSimulations = "BoldSimulation"
    UnicodeString   = "Open file"
    Fill           = "SlateGray"
    OriginX        = "400"
    OriginY        = "75"
/>

<!-- "Open file" with "fi" ligature -->
<Glyphs
    FontUri      = "C:\WINDOWS\Fonts\TIMES.TTF"
    FontRenderingEmSize = "36"
    StyleSimulations = "BoldSimulation"
    UnicodeString   = "Open file"
    Indices        = ";;;;;(2:1)191"
    Fill           = "SlateGray"
    OriginX        = "400"
    OriginY        = "150"
/>

</Canvas>

```

Vedere anche

- [Funzionalità tipografiche di WPF](#)
- [Documenti in WPF](#)
- [Testo](#)

Creare testo utilizzando i glifi

23/10/2019 • 2 minutes to read • [Edit Online](#)

In questo argomento illustra come usare il livello basso [Glyphs](#) oggetto per la visualizzazione testo Extensible Application Markup Language (XAML).

Esempio

Gli esempi seguenti illustrano come definire le proprietà per un [Glyphs](#) dell'oggetto Extensible Application Markup Language (XAML). Il [Glyphs](#) oggetto rappresenta l'output di un [GlyphRun](#) in XAML. Negli esempi si presuppone che i tipi di carattere Arial, Courier New e Times New Roman siano installati nella cartella C:\WINDOWS\Fonts nel computer locale.

```
<!-- The example shows how to use a Glyphs object. -->
<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    >

    <StackPanel Background="PowderBlue">

        <Glyphs
            FontUri          = "C:\WINDOWS\Fonts\TIMES.TTF"
            FontRenderingEmSize = "100"
            StyleSimulations   = "BoldSimulation"
            UnicodeString     = "Hello World!"
            Fill              = "Black"
            OriginX           = "100"
            OriginY           = "200"
        />

    </StackPanel>
</Page>
```

In questo esempio viene illustrato come definire altre proprietà degli [Glyphs](#) oggetti XAML.

```
<!-- The example shows how to use different property settings of Glyphs objects. -->
<Canvas
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Background="PowderBlue"
    >

    <Glyphs
        FontUri          = "C:\WINDOWS\Fonts\ARIAL.TTF"
        FontRenderingEmSize = "36"
        StyleSimulations   = "ItalicSimulation"
        UnicodeString     = "Hello World!"
        Fill              = "SteelBlue"
        OriginX           = "50"
        OriginY           = "75"
    />

    <!-- "Hello World!" with default kerning -->
    <Glyphs
        FontUri          = "C:\WINDOWS\Fonts\ARIAL.TTF"
        FontRenderingEmSize = "36"
        UnicodeString     = "Hello World!"
```

```

        Fill          = "Maroon"
        OriginX      = "50"
        OriginY      = "150"
    />

<!-- "Hello World!" with explicit character widths for proportional font -->
<Glyphs
    FontUri          = "C:\WINDOWS\Fonts\ARIAL.TTF"
    FontRenderingEmSize = "36"
    UnicodeString     = "Hello World!"
    Indices           = ",80;,80;,80;,80;,80;,80;,80;,80;,80;,80;"
    Fill              = "Maroon"
    OriginX          = "50"
    OriginY          = "225"
/>

<!-- "Hello World!" with fixed-width font -->
<Glyphs
    FontUri          = "C:\WINDOWS\Fonts\COUR.TTF"
    FontRenderingEmSize = "36"
    StyleSimulations = "BoldSimulation"
    UnicodeString     = "Hello World!"
    Fill              = "Maroon"
    OriginX          = "50"
    OriginY          = "300"
/>

<!-- "Open file" without "fi" ligature -->
<Glyphs
    FontUri          = "C:\WINDOWS\Fonts\TIMES.TTF"
    FontRenderingEmSize = "36"
    StyleSimulations = "BoldSimulation"
    UnicodeString     = "Open file"
    Fill              = "SlateGray"
    OriginX          = "400"
    OriginY          = "75"
/>

<!-- "Open file" with "fi" ligature -->
<Glyphs
    FontUri          = "C:\WINDOWS\Fonts\TIMES.TTF"
    FontRenderingEmSize = "36"
    StyleSimulations = "BoldSimulation"
    UnicodeString     = "Open file"
    Indices           = ";;;;;(2:1)191"
    Fill              = "SlateGray"
    OriginX          = "400"
    OriginY          = "150"
/>

</Canvas>

```

Vedere anche

- [Funzionalità tipografiche di WPF](#)

Procedure relative alla tipografia

23/10/2019 • 2 minutes to read • [Edit Online](#)

Negli argomenti di questa sezione viene descritto come utilizzare Windows Presentation Foundation (WPF) Sopporti avanzati di presentazione del testo nelle applicazioni.

In questa sezione

[Creare un effetto di testo](#)

[Specificare se un collegamento ipertestuale è sottolineato](#)

[Applicare trasformazioni al testo](#)

[Applicare animazioni al testo](#)

[Creare un testo con un'ombreggiatura](#)

[Creare testo con contorni](#)

[Disegnare un testo sullo sfondo di un controllo](#)

[Disegnare testo in un oggetto Visual](#)

[Usare caratteri speciali in XAML](#)

Vedere anche

- [Typography](#)
- [Documenti in WPF](#)
- [Funzionalità dei tipi di carattere OpenType](#)

Procedura: Creare un effetto testo

23/10/2019 • 5 minutes to read • [Edit Online](#)

Oggetto [TextDecoration](#) oggetto è un ornamento visivo è possibile aggiungere al testo. Esistono quattro tipi di effetti del testo: sottolineato, linea di base, quali il barrato e linea sopra. Nell'esempio seguente mostra la posizione delle decorazioni di testo rispetto al testo.



Per aggiungere un effetto di testo al testo, creare un [TextDecoration](#) dell'oggetto e modificarne le proprietà. Usare il [Location](#) proprietà per specificare dove viene visualizzato l'effetto di testo, quali sottolineato. Usare il [Pen](#) proprietà per specificare l'aspetto dell'effetto di testo, ad esempio un riempimento a tinta unita o sfumatura di colore. Se non si specifica un valore per il [Pen](#) proprietà, i valori predefiniti delle decorazioni sullo stesso colore del testo. Dopo aver definito una [TextDecoration](#) dell'oggetto, aggiungerla al [TextDecorations](#) insieme dell'oggetto testo desiderato.

L'esempio seguente illustra un effetto di testo che è stato disegnato con un pennello sfumato lineare e una penna tratteggiata.

emphasis

Il [Hyperlink](#) oggetto è un elemento di contenuto di flusso di livello inline che consente di ospitare collegamenti ipertestuali all'interno del contenuto dinamico. Per impostazione predefinita [Hyperlink](#) Usa un [TextDecoration](#) oggetto per visualizzare un carattere di sottolineatura. [TextDecoration](#) gli oggetti possono essere prestazioni elevate per creare un'istanza, in particolare se si dispone di numerosi [Hyperlink](#) oggetti. Se si usano ampiamente [Hyperlink](#) elementi, è possibile provare a visualizzare una sottolineatura solo al momento della generazione di un evento, ad esempio il [MouseEnter](#) evento.

Nell'esempio seguente, la sottolineatura per il collegamento "My MSN" è dinamica, viene visualizzata solo quando il [MouseEnter](#) evento viene attivato.



Per altre informazioni, vedere [Specificare se un collegamento ipertestuale è sottolineato](#).

Esempio

Nell'esempio di codice seguente, un effetto di testo sottolineato viene utilizzato il valore del tipo di carattere predefinito.

```
// Use the default font values for the strikethrough text decoration.
private void SetDefaultStrikethrough()
{
    // Set the underline decoration directly to the text block.
    TextBlock1.TextDecorations = TextDecorations.Strikethrough;
}
```

```
' Use the default font values for the strikethrough text decoration.
Private Sub SetDefaultStrikethrough()
    ' Set the underline decoration directly to the text block.
    TextBlock1.TextDecorations = TextDecorations.Strikethrough
End Sub
```

```
<!-- Use the default font values for the strikethrough text decoration. -->
<TextBlock
    TextDecorations="Strikethrough"
    FontSize="36" >
    The quick red fox
</TextBlock>
```

Nell'esempio di codice seguente, un effetto di testo sottolineato viene creato con un pennello di colore a tinta unita per la penna.

```
// Use a Red pen for the underline text decoration.
private void SetRedUnderline()
{
    // Create an underline text decoration. Default is underline.
    TextDecoration myUnderline = new TextDecoration();

    // Create a solid color brush pen for the text decoration.
    myUnderline.Pen = new Pen(Brushes.Red, 1);
    myUnderline.PenThicknessUnit = TextDecorationUnit.FontRecommended;

    // Set the underline decoration to a TextDecorationCollection and add it to the text block.
    TextDecorationCollection myCollection = new TextDecorationCollection();
    myCollection.Add(myUnderline);
    TextBlock2.TextDecorations = myCollection;
}
```

```
' Use a Red pen for the underline text decoration.
Private Sub SetRedUnderline()
    ' Create an underline text decoration. Default is underline.
    Dim myUnderline As New TextDecoration()

    ' Create a solid color brush pen for the text decoration.
    myUnderline.Pen = New Pen(Brushes.Red, 1)
    myUnderline.PenThicknessUnit = TextDecorationUnit.FontRecommended

    ' Set the underline decoration to a TextDecorationCollection and add it to the text block.
    Dim myCollection As New TextDecorationCollection()
    myCollection.Add(myUnderline)
    TextBlock2.TextDecorations = myCollection
End Sub
```

```

<!-- Use a Red pen for the underline text decoration -->
<TextBlock
    FontSize="36" >
    jumps over
<TextBlock.TextDecorations>
    <TextDecorationCollection>
        <TextDecoration
            PenThicknessUnit="FontRecommended">
            <TextDecoration.Pen>
                <Pen Brush="Red" Thickness="1" />
            </TextDecoration.Pen>
        </TextDecoration>
    </TextDecorationCollection>
</TextBlock.TextDecorations>
</TextBlock>

```

Nell'esempio di codice seguente, un effetto di testo sottolineato viene creato con un pennello sfumato lineare per la penna tratteggiata.

```

// Use a linear gradient pen for the underline text decoration.
private void SetLinearGradientUnderline()
{
    // Create an underline text decoration. Default is underline.
    TextDecoration myUnderline = new TextDecoration();

    // Create a linear gradient pen for the text decoration.
    Pen myPen = new Pen();
    myPen.Brush = new LinearGradientBrush(Colors.Yellow, Colors.Red, new Point(0, 0.5), new Point(1, 0.5));
    myPen.Opacity = 0.5;
    myPen.Thickness = 1.5;
    myPen.DashStyle = DashStyles.Dash;
    myUnderline.Pen = myPen;
    myUnderline.PenThicknessUnit = TextDecorationUnit.FontRecommended;

    // Set the underline decoration to a TextDecorationCollection and add it to the text block.
    TextDecorationCollection myCollection = new TextDecorationCollection();
    myCollection.Add(myUnderline);
    TextBlock3.TextDecorations = myCollection;
}

```

```

' Use a linear gradient pen for the underline text decoration.
Private Sub SetLinearGradientUnderline()
    ' Create an underline text decoration. Default is underline.
    Dim myUnderline As New TextDecoration()

    ' Create a linear gradient pen for the text decoration.
    Dim myPen As New Pen()
    myPen.Brush = New LinearGradientBrush(Colors.Yellow, Colors.Red, New Point(0, 0.5), New Point(1, 0.5))
    myPen.Opacity = 0.5
    myPen.Thickness = 1.5
    myPen.DashStyle = DashStyles.Dash
    myUnderline.Pen = myPen
    myUnderline.PenThicknessUnit = TextDecorationUnit.FontRecommended

    ' Set the underline decoration to a TextDecorationCollection and add it to the text block.
    Dim myCollection As New TextDecorationCollection()
    myCollection.Add(myUnderline)
    TextBlock3.TextDecorations = myCollection
End Sub

```

```
<!-- Use a linear gradient pen for the underline text decoration. -->
<TextBlock FontSize="36">the lazy brown dog.
<TextBlock.TextDecorations>
  <TextDecorationCollection>
    <TextDecoration
      PenThicknessUnit="FontRecommended">
      <TextDecoration.Pen>
        <Pen Thickness="1.5">
          <Pen.Brush>
            <LinearGradientBrush Opacity="0.5"
              StartPoint="0,0.5" EndPoint="1,0.5">
              <LinearGradientBrush.GradientStops>
                <GradientStop Color="Yellow" Offset="0" />
                <GradientStop Color="Red" Offset="1" />
              </LinearGradientBrush.GradientStops>
            </LinearGradientBrush>
          </Pen.Brush>
        <Pen.DashStyle>
          <DashStyle Dashes="2"/>
        </Pen.DashStyle>
      </Pen>
    </TextDecoration.Pen>
  </TextDecoration>
</TextDecorationCollection>
</TextBlock.TextDecorations>
</TextBlock>
```

Vedere anche

- [TextDecoration](#)
- [Hyperlink](#)
- [Specificare se un collegamento ipertestuale è sottolineato](#)

Procedura: Specificare se un collegamento ipertestuale è sottolineato

23/10/2019 • 2 minutes to read • [Edit Online](#)

Il [Hyperlink](#) oggetto è un elemento di contenuto di flusso di livello inline che consente di ospitare collegamenti ipertestuali all'interno del contenuto dinamico. Per impostazione predefinita [Hyperlink](#) Usa un [TextDecoration](#) oggetto per visualizzare un carattere di sottolineatura. [TextDecoration](#) gli oggetti possono essere prestazioni elevate per creare un'istanza, in particolare se si dispone di numerosi [Hyperlink](#) oggetti. Se si usano ampiamente [Hyperlink](#) elementi, è possibile provare a visualizzare una sottolineatura solo al momento della generazione di un evento, ad esempio il [MouseEnter](#) evento.

Nell'esempio seguente, la sottolineatura per il collegamento "My MSN" è dinamica, vale a dire, essa viene visualizzata solo quando il [MouseEnter](#) evento viene attivato.



Esempio

L'esempio di markup seguente mostra un [Hyperlink](#) definito con e senza sottolineatura:

```
<!-- Hyperlink with default underline. -->
<Hyperlink NavigateUri="http://www.msn.com">
    MSN Home
</Hyperlink>

<Run Text=" | " />

<!-- Hyperlink with no underline. -->
<Hyperlink Name="myHyperlink" TextDecorations="None">
    MouseEnter="OnMouseEnter"
    MouseLeave="OnMouseLeave"
    NavigateUri="http://www.msn.com">
    My MSN
</Hyperlink>
```

Esempio di codice seguente viene illustrato come creare un carattere di sottolineatura per il [Hyperlink](#) nella [MouseEnter](#) evento e rimuoverlo nel [MouseLeave](#) evento.

```
// Display the underline on only the MouseEnter event.  
private void OnMouseEnter(object sender, EventArgs e)  
{  
    myHyperlink.TextDecorations = TextDecorations.Underline;  
}  
  
// Remove the underline on the MouseLeave event.  
private void OnMouseLeave(object sender, EventArgs e)  
{  
    myHyperlink.TextDecorations = null;  
}
```

```
' Display the underline on only the MouseEnter event.  
Private Overloads Sub OnMouseEnter(ByVal sender As Object, ByVal e As EventArgs)  
    myHyperlink.TextDecorations = TextDecorations.Underline  
End Sub  
  
' Remove the underline on the MouseLeave event.  
Private Overloads Sub OnMouseLeave(ByVal sender As Object, ByVal e As EventArgs)  
    myHyperlink.TextDecorations = Nothing  
End Sub
```

Vedere anche

- [TextDecoration](#)
- [Hyperlink](#)
- [Ottimizzazione delle prestazioni di applicazioni WPF](#)
- [Creare un effetto di testo](#)

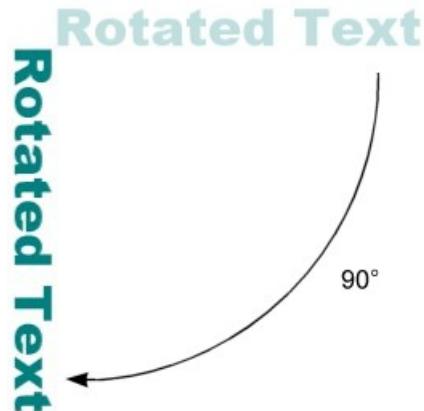
Procedura: Applicare trasformazioni al testo

23/10/2019 • 3 minutes to read • [Edit Online](#)

Le trasformazioni possono modificare la visualizzazione del testo nell'applicazione. Negli esempi seguenti vengono usati tipi diversi di trasformazioni di rendering per influenzare la visualizzazione del testo **TextBlock** in un controllo.

Esempio

L'esempio seguente illustra la rotazione del testo intorno a un punto specifico del piano x-y bidimensionale.



Nell'esempio di codice seguente viene **RotateTransform** usato un oggetto per ruotare il testo. Il **Angle** valore 90 ruota l'elemento 90 gradi in senso orario.

```
<!-- Rotate the text 90 degrees using a RotateTransform. -->
<TextBlock FontFamily="Arial Black" FontSize="64" Foreground="Moccasin" Margin ="80, 10, 0, 0">
    Text Transforms
    <TextBlock.RenderTransform>
        <RotateTransform Angle="90" />
    </TextBlock.RenderTransform>
</TextBlock>
```

Nell'esempio seguente la seconda riga del testo è ridimensionata del 150% lungo l'asse x, mentre la terza riga del testo è ridimensionata del 150% lungo l'asse y.

Scaled Text

Scaled Text

Scaled Text

Nell'esempio di codice seguente viene **ScaleTransform** usato un oggetto per ridimensionare il testo dalle dimensioni originali.

```

<!-- Scale the text using a ScaleTransform. -->
<TextBlock
    Name="textblockScaleMaster"
    FontSize="32"
    Foreground="SteelBlue"
    Text="Scaled Text"
    Margin="100, 0, 0, 0"
    Grid.Column="0" Grid.Row="0">
</TextBlock>
<TextBlock
    FontSize="32"
    FontWeight="Bold"
    Foreground="SteelBlue"
    Text="{Binding Path=Text, ElementName=textblockScaleMaster}"
    Margin="100, 0, 0, 0"
    Grid.Column="0" Grid.Row="1">
    <TextBlock.RenderTransform>
        <ScaleTransform ScaleX="1.5" ScaleY="1.0" />
    </TextBlock.RenderTransform>
</TextBlock>
<TextBlock
    FontSize="32"
    FontWeight="Bold"
    Foreground="SteelBlue"
    Text="{Binding Path=Text, ElementName=textblockScaleMaster}"
    Margin="100, 0, 0, 0"
    Grid.Column="0" Grid.Row="2">
    <TextBlock.RenderTransform>
        <ScaleTransform ScaleX="1.0" ScaleY="1.5" />
    </TextBlock.RenderTransform>
</TextBlock>

```

NOTE

Il ridimensionamento del testo non coincide con l'aumento delle dimensioni dei caratteri del testo. Le dimensioni dei caratteri vengono calcolate in modo indipendente per fornire la migliore risoluzione a dimensioni diverse. Il testo ridimensionato mantiene invece le proporzioni del testo nelle dimensioni originali.

Nell'esempio seguente il testo è inclinato lungo l'asse x.

Skewed Text
Skewed Text

Nell'esempio di codice seguente viene [SkewTransform](#) usato un oggetto per inclinare il testo. L'inclinazione, nota anche come distorsione, è una trasformazione che estende lo spazio delle coordinate in modo non uniforme. In questo esempio le due stringhe di testo sono inclinate di -30° e 30° lungo la coordinata x.

```

<!-- Skew the text using a SkewTransform. -->
<TextBlock
    Name="textblockSkewMaster"
    FontSize="32"
    FontWeight="Bold"
    Foreground="Maroon"
    Text="Skewed Text"
    Margin="125, 0, 0, 0"
    Grid.Column="0" Grid.Row="0">
    <TextBlock.RenderTransform>
        <SkewTransform AngleX="-30" AngleY="0" />
    </TextBlock.RenderTransform>
</TextBlock>
<TextBlock
    FontSize="32"
    FontWeight="Bold"
    Foreground="Maroon"
    Text="{Binding Path=Text, ElementName=textblockSkewMaster}"
    Margin="100, 0, 0, 0"
    Grid.Column="0" Grid.Row="1">
    <TextBlock.RenderTransform>
        <SkewTransform AngleX="30" AngleY="0" />
    </TextBlock.RenderTransform>
</TextBlock>

```

Nell'esempio seguente il testo è traslato, o spostato, lungo l'asse x e y.

Translated Text

Nell'esempio di codice seguente viene [TranslateTransform](#) usato un oggetto per eseguire l'offset del testo. In questo esempio una copia del testo con un leggero offset sotto il testo primario crea un effetto di ombreggiatura.

```

<!-- Skew the text using a TranslateTransform. -->
<TextBlock
    FontSize="32"
    FontWeight="Bold"
    Foreground="Black"
    Text="{Binding Path=Text, ElementName=textblockTranslateMaster}"
    Margin="100, 0, 0, 0"
    Grid.Column="0" Grid.Row="0">
    <TextBlock.RenderTransform>
        <TranslateTransform X="2" Y="2" />
    </TextBlock.RenderTransform>
</TextBlock>
<TextBlock
    Name="textblockTranslateMaster"
    FontSize="32"
    FontWeight="Bold"
    Foreground="Coral"
    Text="Translated Text"
    Margin="100, 0, 0, 0"
    Grid.Column="0" Grid.Row="0"/>

```

NOTE

In [DropShadowBitmapEffect](#) è disponibile un set completo di funzionalità per la fornitura di effetti ombreggiatura. Per altre informazioni, vedere [creare un testo con un'ombreggiatura](#).

Vedere anche

- Applicare animazioni al testo

Procedura: Applicare animazioni al testo

23/10/2019 • 2 minutes to read • [Edit Online](#)

Le animazioni possono modificare la visualizzazione e l'aspetto del testo nell'applicazione. Gli esempi seguenti usano diversi tipi di animazioni per modificare la visualizzazione di testo in un **TextBlock** controllo.

Esempio

L'esempio seguente usa un **DoubleAnimation** per animare la larghezza del blocco di testo. Il valore della larghezza varia dalla larghezza del blocco di testo a 0 per una durata pari a 10 secondi, quindi inverte i valori della larghezza e continua. Questo tipo di animazione crea un effetto a comparsa.

```
<TextBlock  
    Name="MyWipedText"  
    Margin="20"  
    Width="480" Height="100" FontSize="48" FontWeight="Bold" Foreground="Maroon">  
    This is wiped text  
  
    <!-- Animates the text block's width. -->  
    <TextBlock.Triggers>  
        <EventTrigger RoutedEvent="TextBlock.Loaded">  
            <BeginStoryboard>  
                <Storyboard>  
                    <DoubleAnimation  
                        Storyboard.TargetName="MyWipedText"  
                        Storyboard.TargetProperty="(TextBlock.Width)"  
                        To="0.0" Duration="0:0:10"  
                        AutoReverse="True" RepeatBehavior="Forever" />  
                </Storyboard>  
            </BeginStoryboard>  
        </EventTrigger>  
    </TextBlock.Triggers>  
</TextBlock>
```

L'esempio seguente usa un **DoubleAnimation** animare l'opacità del blocco di testo. Il valore dell'opacità varia da 1,0 a 0 per una durata pari a 5 secondi, quindi inverte i valori dell'opacità e continua.

```

<TextBlock
    Name="MyFadingText"
    Margin="20"
    Width="640" Height="100" FontSize="48" FontWeight="Bold" Foreground="Maroon">
    This is fading text

    <!-- Animates the text block's opacity. -->
    <TextBlock.Triggers>
        <EventTrigger RoutedEvent="TextBlock.Loaded">
            <BeginStoryboard>
                <Storyboard>
                    <DoubleAnimation
                        Storyboard.TargetName="MyFadingText"
                        Storyboard.TargetProperty="(TextBlock.Opacity)"
                        From="1.0" To="0.0" Duration="0:0:5"
                        AutoReverse="True" RepeatBehavior="Forever" />
                </Storyboard>
            </BeginStoryboard>
        </EventTrigger>
    </TextBlock.Triggers>
</TextBlock>

```

Il diagramma seguente mostra l'effetto del [TextBlock](#) modifica l'opacità viene modificata dal controllo **1.00** a **0.00** durante l'intervallo di 5 secondi definito dal [Duration](#).

Opacity	
This is fading text	1.00
This is fading text	0.80
This is fading text	0.60
This is fading text	0.40
This is fading text	0.20
	0.00

L'esempio seguente usa un [ColorAnimation](#) animare il colore di primo piano del blocco di testo. Il valore del colore di primo piano varia da un colore a un altro per una durata pari a 5 secondi, quindi inverte i valori del colore e continua.

```

<TextBlock
    Name="MyChangingColorText"
    Margin="20"
    Width="640" Height="100" FontSize="48" FontWeight="Bold">
    This is changing color text
<TextBlock.Foreground>
    <SolidColorBrush x:Name="MySolidColorBrush" Color="Maroon" />
</TextBlock.Foreground>

<!-- Animates the text block's color. -->
<TextBlock.Triggers>
    <EventTrigger RoutedEvent="TextBlock.Loaded">
        <BeginStoryboard>
            <Storyboard>
                <ColorAnimation
                    Storyboard.TargetName="MySolidColorBrush"
                    Storyboard.TargetProperty="Color"
                    From="DarkOrange" To="SteelBlue" Duration="0:0:5"
                    AutoReverse="True" RepeatBehavior="Forever" />
            </Storyboard>
        </BeginStoryboard>
    </EventTrigger>
</TextBlock.Triggers>
</TextBlock>

```

L'esempio seguente usa un [DoubleAnimation](#) per ruotare il blocco di testo. Il blocco di testo esegue una rotazione completa per una durata pari a 20 secondi, quindi continua a ripetere la rotazione.

```

<TextBlock
    Name="MyRotatingText"
    Margin="20"
    Width="640" Height="100" FontSize="48" FontWeight="Bold" Foreground="Teal"
    >
    This is rotating text
    <TextBlock.RenderTransform>
        <RotateTransform x:Name="MyRotateTransform" Angle="0" CenterX="230" CenterY="25"/>
    </TextBlock.RenderTransform>

    <!-- Animates the text block's rotation. -->
    <TextBlock.Triggers>
        <EventTrigger RoutedEvent="TextBlock.Loaded">
            <BeginStoryboard>
                <Storyboard>
                    <DoubleAnimation
                        Storyboard.TargetName="MyRotateTransform"
                        Storyboard.TargetProperty="(RotateTransform.Angle)"
                        From="0.0" To="360" Duration="0:0:10"
                        RepeatBehavior="Forever" />
                </Storyboard>
            </BeginStoryboard>
        </EventTrigger>
    </TextBlock.Triggers>
</TextBlock>

```

Vedere anche

- [Cenni preliminari sull'animazione](#)

Procedura: Creare testo con un'ombreggiatura

23/10/2019 • 4 minutes to read • [Edit Online](#)

Gli esempi inclusi in questa sezione mostrano come creare un effetto di ombreggiatura per il testo visualizzato.

Esempio

L' [DropShadowEffect](#) oggetto consente di creare un'ampia gamma di effetti ombreggiatura per Windows Presentation Foundation (WPF) gli oggetti. Nell'esempio seguente viene illustrato un effetto di ombreggiatura applicato al testo. Poiché in questo caso l'ombreggiatura è sfumata, il colore dell'ombreggiatura sarà sfocato.

Shadow Text

È possibile controllare la larghezza di un'ombreggiatura impostando [ShadowDepth](#) la proprietà. Il valore `4.0` indica una larghezza ombreggiata di 4 pixel. È possibile controllare la morbidezza o la sfocatura di un'ombreggiatura modificando [BlurRadius](#) la proprietà. Un valore `0.0` indica che non è presente alcuna sfocatura. Nell'esempio di codice seguente viene illustrato come creare un'ombreggiatura sfumata.

```
<!-- Soft single shadow. -->
<TextBlock
    Text="Shadow Text"
    Foreground="Teal">
    <TextBlock.Effect>
        <DropShadowEffect
            ShadowDepth="4"
            Direction="330"
            Color="Black"
            Opacity="0.5"
            BlurRadius="4"/>
    </TextBlock.Effect>
</TextBlock>
```

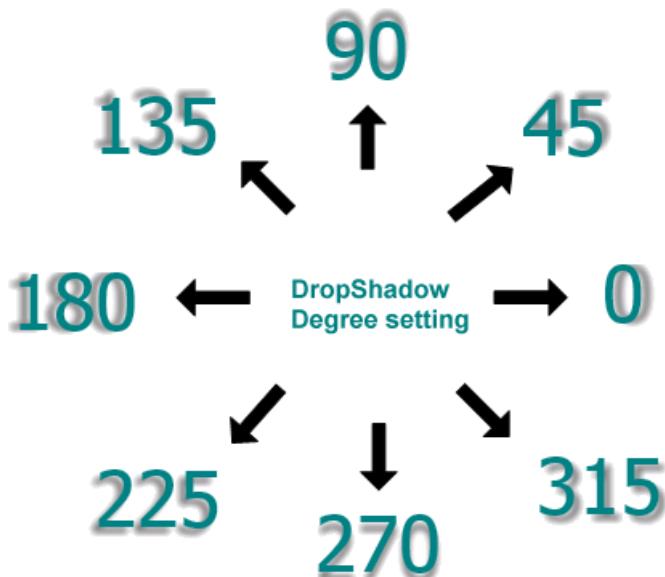
NOTE

Questi effetti di ombreggiatura non passano attraverso Windows Presentation Foundation (WPF) la pipeline di rendering del testo. Di conseguenza, ClearType è disabilitato quando si usano questi effetti.

Nell'esempio seguente viene illustrato un effetto di ombreggiatura piena applicato al testo. In questo caso, l'ombreggiatura non è sfocata.

Shadow Text

È possibile creare un'ombreggiatura rigida impostando [BlurRadius](#) la `0.0` proprietà su, che indica che non viene utilizzata alcuna sfocatura. È possibile controllare la direzione dell'ombreggiatura modificando la [Direction](#) proprietà. Impostare il valore direzionale di questa proprietà su un grado compreso `0` tra `360` e. Nella figura seguente vengono mostrati i valori direzionali [Direction](#) dell'impostazione della proprietà.



Nell'esempio di codice seguente viene illustrato come creare un'ombreggiatura piena.

```
<!-- Hard single shadow. -->
<TextBlock
    Text="Shadow Text"
    Foreground="Maroon">
    <TextBlock.Effect>
        <DropShadowEffect
            ShadowDepth="6"
            Direction="135"
            Color="Maroon"
            Opacity="0.35"
            BlurRadius="0.0" />
    </TextBlock.Effect>
</TextBlock>
```

Uso di un effetto di sfocatura

Un [BlurBitmapEffect](#) oggetto può essere usato per creare un effetto simile all'ombreggiatura che può essere posizionato dietro un oggetto di testo. Un effetto bitmap di sfocatura applicato al testo consente di sfocare il testo in tutte le direzioni in modo uniforme.

L'esempio seguente illustra un effetto di sfocatura applicato al testo.

Shadow Text

Nell'esempio di codice seguente viene illustrato come creare un effetto di sfocatura.

```
<!-- Shadow effect by creating a blur. -->
<TextBlock
    Text="Shadow Text"
    Foreground="Green"
    Grid.Column="0" Grid.Row="0" >
    <TextBlock.Effect>
        <BlurEffect
            Radius="8.0"
            KernelType="Box"/>
    </TextBlock.Effect>
</TextBlock>
<TextBlock
    Text="Shadow Text"
    Foreground="Maroon"
    Grid.Column="0" Grid.Row="0" />
```

Uso di una trasformazione di traslazione

Un [TranslateTransform](#) oggetto può essere usato per creare un effetto simile all'ombreggiatura che può essere posizionato dietro un oggetto di testo.

Nell'esempio di codice seguente viene [TranslateTransform](#) usato un oggetto per eseguire l'offset del testo. In questo esempio una copia del testo con un leggero offset sotto il testo primario crea un effetto di ombreggiatura.

Shadow Text

Nell'esempio di codice seguente viene illustrato come creare una trasformazione per un effetto di ombreggiatura.

```
<!-- Shadow effect by creating a transform. -->
<TextBlock
    Foreground="Black"
    Text="Shadow Text"
    Grid.Column="0" Grid.Row="0">
    <TextBlock.RenderTransform>
        <TranslateTransform X="3" Y="3" />
    </TextBlock.RenderTransform>
</TextBlock>
<TextBlock
    Foreground="Coral"
    Text="Shadow Text"
    Grid.Column="0" Grid.Row="0" >
</TextBlock>
```

Procedura: Creare testo con contorni

23/10/2019 • 5 minutes to read • [Edit Online](#)

Nella maggior parte dei casi, quando si aggiungono ornamento per le stringhe di testo di Windows Presentation Foundation (WPF) un'applicazione, si usa il testo in termini di una raccolta di caratteri discreti o glifi. Ad esempio, è possibile creare un pennello sfumato lineare e metterle in pratica per la [Foreground](#) proprietà di un [TextBox](#) oggetto. Quando si visualizzano o si modifica la casella di testo, il pennello sfumato lineare viene applicato automaticamente al set corrente di caratteri nella stringa di testo.

Spectrum Foreground

Tuttavia, è anche possibile convertire testo in [Geometry](#) oggetti, consentendo di creare altri tipi di testo visivamente accattivanti. Ad esempio, è possibile creare un [Geometry](#) oggetto in base alla struttura di una stringa di testo.

Spectrum Outline

Quando il testo viene convertito in un [Geometry](#) dell'oggetto, non è più una raccolta di caratteri, non è possibile modificare i caratteri nella stringa di testo. Tuttavia, è possibile intervenire sull'aspetto del testo convertito modificandone le proprietà del tratto e del riempimento. Il tratto fa riferimento alla struttura del testo convertito, mentre il riempimento fa riferimento all'area all'interno della struttura del testo convertito.

Gli esempi seguenti illustrano alcuni modi per creare effetti visivi modificando il tratto e riempimento del testo convertito.



È anche possibile modificare il rettangolo delimitatore, o un'evidenziazione, del testo convertito. L'esempio seguente illustra un modo per creare effetti visivi modificando il tratto ed evidenziazione del testo convertito.



Esempio

La chiave per la conversione del testo a un [Geometry](#) oggetto consiste nell'usare il [FormattedText](#) oggetto. Dopo aver creato questo oggetto, è possibile usare la [BuildGeometry](#) e [BuildHighlightGeometry](#) metodi per convertire il testo da [Geometry](#) oggetti. Il primo metodo restituisce la geometria del testo formattato; il secondo metodo restituisce che la geometria del testo formattato del rettangolo. Esempio di codice seguente viene illustrato come creare un [FormattedText](#) oggetto e recuperare le geometrie di testo formattato e il relativo riquadro delimitatore.

```

/// <summary>
/// Create the outline geometry based on the formatted text.
/// </summary>
public void CreateText()
{
    System.Windows.FontStyle fontStyle = FontStyles.Normal;
    FontWeight fontWeight = FontWeights.Medium;

    if (Bold == true) fontWeight = FontWeights.Bold;
    if (Italic == true) fontStyle = FontStyles.Italic;

    // Create the formatted text based on the properties set.
    FormattedText formattedText = new FormattedText(
        Text,
        CultureInfo.GetCultureInfo("en-us"),
        FlowDirection.LeftToRight,
        new Typeface(
            Font,
            fontStyle,
            fontWeight,
            FontStretches.Normal),
        FontSize,
        System.Windows.Media.Brushes.Black // This brush does not matter since we use the geometry of the
text.
    );

    // Build the geometry object that represents the text.
    _textGeometry = formattedText.BuildGeometry(new System.Windows.Point(0, 0));

    // Build the geometry object that represents the text highlight.
    if (Highlight == true)
    {
        _textHighLightGeometry = formattedText.BuildHighlightGeometry(new System.Windows.Point(0, 0));
    }
}

```

```

''' <summary>
''' Create the outline geometry based on the formatted text.
''' </summary>
Public Sub CreateText()
    Dim fontStyle As FontStyle = FontStyles.Normal
    Dim fontWeight As FontWeight = FontWeights.Medium

    If Bold = True Then
        fontWeight = FontWeights.Bold
    End If
    If Italic = True Then
        fontStyle = FontStyles.Italic
    End If

    ' Create the formatted text based on the properties set.
    Dim formattedText As New FormattedText(Text, CultureInfo.GetCultureInfo("en-us"),
    FlowDirection.LeftToRight, New Typeface(Font, fontStyle, fontWeight, FontStretches.Normal), FontSize,
    Brushes.Black) ' This brush does not matter since we use the geometry of the text.

    ' Build the geometry object that represents the text.
    _textGeometry = formattedText.BuildGeometry(New Point(0, 0))

    ' Build the geometry object that represents the text highlight.
    If Highlight = True Then
        _textHighLightGeometry = formattedText.BuildHighlightGeometry(New Point(0, 0))
    End If
End Sub

```

Per visualizzare l'oggetto recuperato il [Geometry](#) oggetti, è necessario accedere il [DrawingContext](#) dell'oggetto che viene visualizzato il testo convertito. Questi esempi di codice, questa operazione viene eseguita tramite la creazione di un oggetto di controllo personalizzato che deriva da una classe che supporta il rendering definite dall'utente.

Per visualizzare [Geometry](#) oggetti nel controllo personalizzato, specificare una sostituzione per il [OnRender](#) (metodo). Deve usare il metodo sottoposto a override il [DrawGeometry](#) metodo consente di disegnare la [Geometry](#) oggetti.

```
/// <summary>
/// OnRender override draws the geometry of the text and optional highlight.
/// </summary>
/// <param name="drawingContext">Drawing context of the OutlineText control.</param>
protected override void OnRender(DrawingContext drawingContext)
{
    // Draw the outline based on the properties that are set.
    drawingContext.DrawGeometry(Fill, new System.Windows.Media.Pen(Stroke, StrokeThickness), _textGeometry);

    // Draw the text highlight based on the properties that are set.
    if (Highlight == true)
    {
        drawingContext.DrawGeometry(null, new System.Windows.Media.Pen(Stroke, StrokeThickness),
    _textHighLightGeometry);
    }
}
```

```
''' <summary>
''' OnRender override draws the geometry of the text and optional highlight.
''' </summary>
''' <param name="drawingContext">Drawing context of the OutlineText control.</param>
Protected Overrides Sub OnRender(ByVal drawingContext As DrawingContext)
    ' Draw the outline based on the properties that are set.
    drawingContext.DrawGeometry(Fill, New Pen(Stroke, StrokeThickness), _textGeometry)

    ' Draw the text highlight based on the properties that are set.
    If Highlight = True Then
        drawingContext.DrawGeometry(Nothing, New Pen(Stroke, StrokeThickness), _textHighLightGeometry)
    End If
End Sub
```

Per l'origine dell'oggetto di controllo utente personalizzato di esempio, vedere [OutlineTextControl.cs](#) per C# e [OutlineTextControl.vb](#) per Visual Basic.

Vedere anche

- [Disegno di testo formattato](#)

Procedura: creare testo sullo sfondo di un controllo

04/11/2019 • 2 minutes to read • [Edit Online](#)

È possibile disegnare il testo direttamente sullo sfondo di un controllo convertendo una stringa di testo in un oggetto [FormattedText](#), quindi disegnando l'oggetto nella [DrawingContext](#) del controllo. È anche possibile usare questa tecnica per disegnare sullo sfondo di oggetti derivati da [Panel](#), ad esempio [Canvas](#) e [StackPanel](#).



Esempio di controlli con sfondi di testo personalizzati

Esempio

Per creare uno sfondo di un controllo, creare un nuovo oggetto [DrawingBrush](#) e creare il testo convertito nel [DrawingContext](#) dell'oggetto. Assegnare quindi la nuova [DrawingBrush](#) alla proprietà [background](#) del controllo.

Nell'esempio di codice seguente viene illustrato come creare un oggetto [FormattedText](#) e come eseguire il progetto sullo sfondo di un oggetto [Label](#) e [Button](#).

```

// Handle the WindowLoaded event for the window.
private void WindowLoaded(object sender, EventArgs e)
{
    // Update the background property of the label and button.
    myLabel.Background = new DrawingBrush(DrawMyText("My Custom Label"));
    myButton.Background = new DrawingBrush(DrawMyText("Display Text"));
}

// Convert the text string to a geometry and draw it to the control's DrawingContext.
private Drawing DrawMyText(string textString)
{
    // Create a new DrawingGroup of the control.
    DrawingGroup drawingGroup = new DrawingGroup();

    // Open the DrawingGroup in order to access the DrawingContext.
    using (DrawingContext drawingContext = drawingGroup.Open())
    {
        // Create the formatted text based on the properties set.
        FormattedText formattedText = new FormattedText(
            textString,
            CultureInfo.GetCultureInfo("en-us"),
            FlowDirection.LeftToRight,
            new Typeface("Comic Sans MS Bold"),
            48,
            System.Windows.Media.Brushes.Black // This brush does not matter since we use the geometry of the
text.
        );

        // Build the geometry object that represents the text.
        Geometry textGeometry = formattedText.BuildGeometry(new System.Windows.Point(20, 0));

        // Draw a rounded rectangle under the text that is slightly larger than the text.
        drawingContext.DrawRoundedRectangle(System.Windows.Media.Brushes.PapayaWhip, null, new Rect(new
System.Windows.Size(formattedText.Width + 50, formattedText.Height + 5)), 5.0, 5.0);

        // Draw the outline based on the properties that are set.
        drawingContext.DrawGeometry(System.Windows.Media.Brushes.Gold, new
System.Windows.Media.Pen(System.Windows.Media.Brushes.Maroon, 1.5), textGeometry);

        // Return the updated DrawingGroup content to be used by the control.
        return drawingGroup;
    }
}

```

Vedere anche

- [FormattedText](#)
- [Disegno di testo formattato](#)

Procedura: disegnare testo in un oggetto Visual

10/02/2020 • 2 minutes to read • [Edit Online](#)

Nell'esempio seguente viene illustrato come creare testo in un [DrawingVisual](#) utilizzando un oggetto [DrawingContext](#). Un contesto di disegno viene restituito chiamando il metodo [RenderOpen](#) di un oggetto [DrawingVisual](#). È possibile disegnare grafica e testo in un contesto di disegno.

Per disegnare testo nel contesto di disegno, utilizzare il metodo [DrawText](#) di un oggetto [DrawingContext](#). Al termine della creazione del contenuto nel contesto di disegno, chiamare il metodo [Close](#) per chiudere il contesto di disegno e salvare in modo permanente il contenuto.

Esempio

```
// Create a DrawingVisual that contains text.
private DrawingVisual CreateDrawingVisualText()
{
    // Create an instance of a DrawingVisual.
    DrawingVisual drawingVisual = new DrawingVisual();

    // Retrieve the DrawingContext from the DrawingVisual.
    DrawingContext drawingContext = drawingVisual.RenderOpen();

    // Draw a formatted text string into the DrawingContext.
    drawingContext.DrawText(
        new FormattedText("Click Me!",
            CultureInfo.GetCultureInfo("en-us"),
            FlowDirection.LeftToRight,
            new Typeface("Verdana"),
            36, System.Windows.Media.Brushes.Black),
        new System.Windows.Point(200, 116));

    // Close the DrawingContext to persist changes to the DrawingVisual.
    drawingContext.Close();

    return drawingVisual;
}
```

```
' Create a DrawingVisual that contains text.
Private Function CreateDrawingVisualText() As DrawingVisual
    ' Create an instance of a DrawingVisual.
    Dim drawingVisual As New DrawingVisual()

    ' Retrieve the DrawingContext from the DrawingVisual.
    Dim drawingContext As DrawingContext = drawingVisual.RenderOpen()

    ' Draw a formatted text string into the DrawingContext.
    drawingContext.DrawText(New FormattedText("Click Me!", CultureInfo.GetCultureInfo("en-us"),
        FlowDirection.LeftToRight, New Typeface("Verdana"), 36, Brushes.Black), New Point(200, 116))

    ' Close the DrawingContext to persist changes to the DrawingVisual.
    drawingContext.Close()

    Return drawingVisual
End Function
```

NOTE

Per l'esempio di codice completo dal quale è stato estratto l'esempio di codice precedente, vedere [Hit Test Using DrawingVisuals Sample \(Esempio di Hit Test mediante DrawingVisual\)](#).

Procedura: Usare caratteri speciali in XAML

08/11/2019 • 2 minutes to read • [Edit Online](#)

I file di markup creati in Visual Studio vengono salvati automaticamente nel formato di file UTF-8 Unicode, il che significa che la maggior parte dei caratteri speciali, ad esempio i contrassegni accentati, sono codificati correttamente. Tuttavia, esiste un set di caratteri speciali di uso comune che viene gestito diversamente. Questi caratteri speciali seguono lo standard XML World Wide Web Consortium (W3C) per la codifica.

La tabella seguente illustra la sintassi per la codifica di questo set di caratteri speciali:

CARATTERE	SINTASSI	DESCRIZIONE
<	<	Simbolo minore di.
>	>	Simbolo maggiore di.
&	&	Simbolo e commerciale.
"	"	Simbolo virgoletta doppia.

NOTE

Se si crea un file di markup usando un editor di testo, ad esempio Blocco note di Windows, è necessario salvare il file nel formato di file UTF-8 Unicode per mantenere i caratteri speciali codificati.

L'esempio riportato di seguito mostra come usare i caratteri speciali nel testo durante la creazione del markup.

Esempio

```
<!-- Display special characters that require special encoding: < > & " -->
<TextBlock>
    &lt;    <!-- Less than symbol -->
    &gt;    <!-- Greater than symbol -->
    &amp;   <!-- Ampersand symbol -->
    &quot;  <!-- Double quote symbol -->
</TextBlock>

<!-- Display miscellaneous special characters -->
<TextBlock>
    Cæsar  <!-- AE diphthong symbol -->
    © 2006 <!-- Copyright symbol -->
    Español <!-- Tilde symbol -->
    ¥      <!-- Yen symbol -->
</TextBlock>
```

Stampa e gestione di sistemi di stampa

25/11/2019 • 2 minutes to read • [Edit Online](#)

Windows Vista e Microsoft .NET Framework introducono un nuovo percorso di stampa, ovvero un'alternativa alla stampa di Microsoft Windows Graphics Device Interface (GDI), e un set di API di gestione del sistema di stampa notevolmente ampliato.

Contenuto della sezione

[Panoramica della stampa](#)

Descrizione del nuovo percorso di stampa e delle API.

[Procedure relative alle proprietà](#)

Una serie di articoli che illustrano come usare il nuovo percorso di stampa e le API.

Vedere anche

- [System.Printing](#)
- [System.Printing.IndexedProperties](#)
- [System.Printing.Interop](#)
- [Documenti in WPF](#)
- [Documenti XPS](#)

Cenni preliminari sulla stampa

31/01/2020 • 20 minutes to read • [Edit Online](#)

Con Microsoft .NET Framework, gli sviluppatori di applicazioni che usano Windows Presentation Foundation (WPF) dispongono di un nuovo set completo di API di gestione del sistema di stampa e stampa. Con Windows Vista, alcuni di questi miglioramenti apportati al sistema di stampa sono disponibili anche per gli sviluppatori che creano applicazioni Windows Forms e sviluppatori che usano codice non gestito. Alla base di questa nuova funzionalità si trova il nuovo formato di file XPS (XML Paper Specification) e il percorso di stampa XPS.

In questo argomento sono contenute le seguenti sezioni.

Informazioni su XPS

XPS è un formato di documento elettronico, un formato di file di spooling e un linguaggio di descrizione della pagina. Si tratta di un formato di documento aperto che usa XML, Open Packaging Conventions (OPC) e altri standard del settore per creare documenti multipiattaforma. XPS semplifica il processo di creazione, condivisione, stampa, visualizzazione e archiviazione di documenti digitali. Per ulteriori informazioni su XPS, vedere la pagina relativa ai [documenti XPS](#).

Diverse tecniche per la stampa di contenuto basato su XPS mediante WPF sono illustrate in [stampa di file XPS a livello di codice](#). Può essere utile fare riferimento a tali esempi durante la lettura di questo argomento. Gli sviluppatori di codice non gestito dovrebbero vedere la documentazione per la [funzione MXDC_ESCAPE](#). Windows Forms gli sviluppatori devono usare l'API nello spazio dei nomi [System.Drawing.Printing](#) che non supporta il percorso di stampa XPS completo, ma supporta un percorso di stampa GDI-XPS ibrido. Vedere [Architettura del percorso di stampa](#) più avanti.

Percorso di stampa XPS

Il percorso di stampa XPS (XML Paper Specification) è una nuova funzionalità di Windows che consente di ridefinire il modo in cui la stampa viene gestita nelle applicazioni Windows. Poiché XPS può sostituire un linguaggio di presentazione del documento, ad esempio RTF, un formato dello spooler di stampa, ad esempio WMF, e un linguaggio di descrizione della pagina (ad esempio, PCL o PostScript); il nuovo percorso di stampa mantiene il formato XPS dalla pubblicazione dell'applicazione all'elaborazione finale nel dispositivo o nel driver di stampa.

Il percorso di stampa XPS è basato sul modello di driver della stampante XPS (XPSSDrv), che offre diversi vantaggi per gli sviluppatori, ad esempio la stampa WYSIWYG, il supporto dei colori migliorato e prestazioni di stampa significativamente migliorate. Per ulteriori informazioni su XPSSDrv, vedere la [documentazione di Windows Driver Kit](#).

Il funzionamento dello spooler di stampa per i documenti XPS è sostanzialmente identico a quello delle versioni precedenti di Windows. Tuttavia, è stato migliorato per supportare il percorso di stampa XPS oltre al percorso di stampa GDI esistente. Il nuovo percorso di stampa USA a livello nativo un file di spooling XPS. Mentre i driver della stampante in modalità utente scritti per le versioni precedenti di Windows continueranno a funzionare, è necessario un driver della stampante XPS (XPSSDrv) per usare il percorso di stampa XPS.

I vantaggi del percorso di stampa XPS sono significativi e includono:

- Supporto per la stampa WYSIWYG
- Supporto nativo di profili colori avanzati che includono 32 bit per canale (bpc), CMYK, colori con nome, molteplici inchiostri e supporto nativo di trasparenza e sfumature.

- Prestazioni di stampa migliorate per applicazioni .NET Framework e basate su Win32.
- Formato XPS standard di settore.

Per gli scenari di stampa di base, è disponibile un'API semplice e intuitiva con un unico punto di ingresso per l'interfaccia utente, la configurazione e l'invio di processi. Per scenari avanzati, è disponibile un supporto aggiuntivo per la personalizzazione dell'interfaccia utente (o nessuna Interfaccia utente), la stampa sincrona o asincrona e le funzionalità di stampa in modalità batch. Entrambe le opzioni forniscono supporto di stampa in modalità di attendibilità completa o parziale.

XPS è stato progettato tenendo presente l'estensibilità. Usando il Framework di estendibilità, le funzionalità e le funzionalità possono essere aggiunte a XPS in modo modulare. Le funzionalità di estensibilità includono:

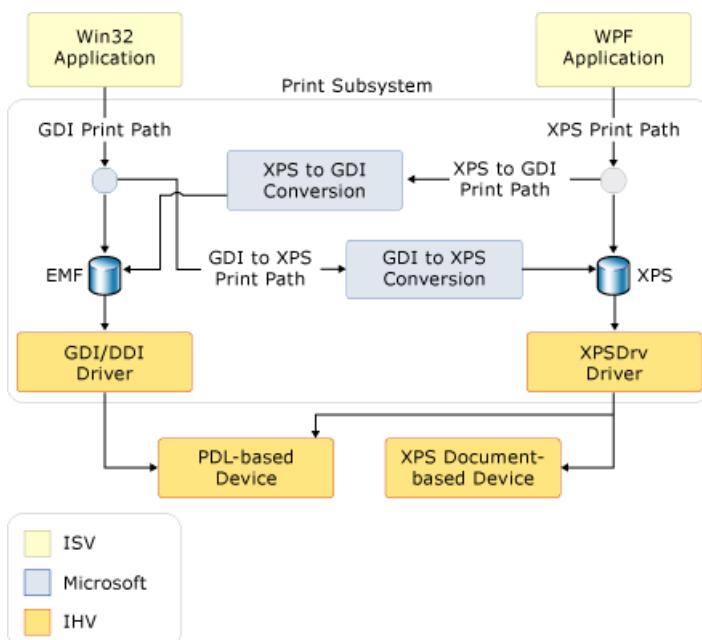
- Schema di stampa. Lo schema pubblico viene aggiornato regolarmente e consente la rapida estensione delle funzionalità del dispositivo Vedere **PrintTicket** e **PrintCapabilities** più avanti.
- Pipeline del filtro estensibile. La pipeline del filtro del driver della stampante XPS (XPSDrv) è stata progettata per consentire la stampa diretta e scalabile di documenti XPS. Per ulteriori informazioni, vedere [driver della stampante XPSDrv](#).

Architettura del percorso di stampa

Anche se le applicazioni Win32 e .NET Framework supportano XPS, le applicazioni Win32 e Windows Forms utilizzano una conversione da GDI a XPS per creare contenuto formattato XPS per XPS Printer driver (XPSDrv). Queste applicazioni non sono necessarie per usare il percorso di stampa XPS e possono continuare a usare la stampa basata su Enhanced Metafile (EMF). Tuttavia, la maggior parte delle funzionalità e dei miglioramenti di XPS è disponibile solo per le applicazioni destinate al percorso di stampa XPS.

Per consentire l'utilizzo di stampanti basate su XPSDrv da parte di applicazioni Win32 e Windows Forms, il driver della stampante XPS (XPSDrv) supporta la conversione del formato GDI in XPS. Il modello XPSDrv fornisce inoltre un convertitore per XPS al formato GDI, in modo che le applicazioni Win32 possano stampare documenti XPS. Per le applicazioni WPF, la conversione di XPS in formato GDI viene eseguita automaticamente dai metodi [Write](#) e [WriteAsync](#) della classe [XpsDocumentWriter](#) ogni volta che la coda di stampa di destinazione dell'operazione di scrittura non dispone di un driver XPSDrv. (Windows Forms le applicazioni non possono stampare documenti XPS).

Nella figura seguente viene illustrato il sottosistema di stampa e vengono definite le parti fornite da Microsoft e le parti definite dai fornitori di software e hardware:



Stampa XPS di base

WPF definisce un'API di base e avanzata. Per le applicazioni che non richiedono una personalizzazione di stampa estesa o l'accesso al set di funzionalità XPS completo, è disponibile il supporto di stampa di base. Il supporto di stampa di base viene esposto tramite un controllo finestra di dialogo di stampa che richiede una configurazione minima e offre un'Interfaccia utente già familiare all'utente. Molte funzionalità XPS sono disponibili tramite questo modello di stampa semplificato.

PrintDialog

Il controllo [System.Windows.Controls.PrintDialog](#) fornisce un singolo punto di ingresso per Interfaccia utente, configurazione e invio di processi XPS. Per informazioni su come creare un'istanza del controllo e come usarlo, vedere [Richiamare una finestra di dialogo di stampa](#).

Stampa XPS avanzata

Per accedere al set completo di funzionalità XPS, è necessario usare l'API di stampa avanzata. Diverse API pertinenti sono descritte in dettaglio più avanti. Per un elenco completo delle API del percorso di stampa XPS, vedere i riferimenti agli spazi dei nomi [System.Windows.Xps](#) e [System.Printing](#).

PrintTicket e PrintCapabilities

Le classi [PrintTicket](#) e [PrintCapabilities](#) sono le fondamenta delle funzionalità avanzate di XPS. Entrambi i tipi di oggetti sono strutture in formato XML di funzionalità orientate alla stampa, ad esempio regole di confronto, stampa su due lati, graffatura e così via. Queste strutture sono definite dallo schema di stampa. Un oggetto [PrintTicket](#) indica a una stampante come elaborare un processo di stampa. La classe [PrintCapabilities](#) consente di definire le funzionalità di una stampante. Eseguendo una query delle funzionalità di una stampante, è possibile creare un oggetto [PrintTicket](#) per usare in modo completo tutte le funzionalità supportate di una stampante. Analogamente, è possibile evitare le funzionalità non supportate.

L'esempio riportato di seguito illustra come eseguire una query della classe [PrintCapabilities](#) di una stampante e creare un oggetto [PrintTicket](#) tramite il codice.

```

// ----- GetPrintTicketFromPrinter -----
/// <summary>
///   Returns a PrintTicket based on the current default printer.</summary>
/// <returns>
///   A PrintTicket for the current local default printer.</returns>
PrintTicket^ GetPrintTicketFromPrinter ()
{
    PrintQueue^ printQueue = nullptr;

    LocalPrintServer^ localPrintServer = gcnew LocalPrintServer();

    // Retrieving collection of local printer on user machine
    PrintQueueCollection^ localPrinterCollection = localPrintServer->GetPrintQueues();

    System::Collections::IEnumerator^ localPrinterEnumerator = localPrinterCollection->GetEnumerator();

    if (localPrinterEnumerator->MoveNext())
    {
        // Get PrintQueue from first available printer
        printQueue = ((PrintQueue^)localPrinterEnumerator->Current);
    }
    else
    {
        return nullptr;
    }

    // Get default PrintTicket from printer
    PrintTicket^ printTicket = printQueue->DefaultPrintTicket;

    PrintCapabilites^ printCapabilites = printQueue->GetPrintCapabilities();

    // Modify PrintTicket
    if (printCapabilites->CollationCapability->Contains(Collation::Collated))
    {
        printTicket->Collation = Collation::Collated;
    }
    if (printCapabilites->DuplexingCapability->Contains(Duplexing::TwoSidedLongEdge))
    {
        printTicket->Duplexing = Duplexing::TwoSidedLongEdge;
    }
    if (printCapabilites->StaplingCapability->Contains(Stapling::StapleDualLeft))
    {
        printTicket->Stapling = Stapling::StapleDualLeft;
    }
    return printTicket;
}// end:GetPrintTicketFromPrinter()

```

```

// ----- GetPrintTicketFromPrinter -----
/// <summary>
/// Returns a PrintTicket based on the current default printer.</summary>
/// <returns>
/// A PrintTicket for the current local default printer.</returns>
private PrintTicket GetPrintTicketFromPrinter()
{
    PrintQueue printQueue = null;

    LocalPrintServer localPrintServer = new LocalPrintServer();

    // Retrieving collection of local printer on user machine
    PrintQueueCollection localPrinterCollection =
        localPrintServer.GetPrintQueues();

    System.Collections.IEnumerator localPrinterEnumerator =
        localPrinterCollection.GetEnumerator();

    if (localPrinterEnumerator.MoveNext())
    {
        // Get PrintQueue from first available printer
        printQueue = (PrintQueue)localPrinterEnumerator.Current;
    }
    else
    {
        // No printer exist, return null PrintTicket
        return null;
    }

    // Get default PrintTicket from printer
    PrintTicket printTicket = printQueue.DefaultPrintTicket;

    PrintCapabilities printCapabilites = printQueue.GetPrintCapabilities();

    // Modify PrintTicket
    if (printCapabilites.CollationCapability.Contains(Collation.Collated))
    {
        printTicket.Collation = Collation.Collated;
    }

    if ( printCapabilites.DuplexingCapability.Contains(
        Duplexing.TwoSidedLongEdge) )
    {
        printTicket.Duplexing = Duplexing.TwoSidedLongEdge;
    }

    if (printCapabilites.StaplingCapability.Contains(Stapling.StapleDualLeft))
    {
        printTicket.Stapling = Stapling.StapleDualLeft;
    }

    return printTicket;
}// end:GetPrintTicketFromPrinter()

```

```

' ----- GetPrintTicketFromPrinter -----
''' <summary>
''' Returns a PrintTicket based on the current default printer.</summary>
''' <returns>
''' A PrintTicket for the current local default printer.</returns>
Private Function GetPrintTicketFromPrinter() As PrintTicket
    Dim printQueue As PrintQueue = Nothing

    Dim localPrintServer As New LocalPrintServer()

    ' Retrieving collection of local printer on user machine
    Dim localPrinterCollection As PrintQueueCollection = localPrintServer.GetPrintQueues()

    Dim localPrinterEnumerator As System.Collections.IEnumerator = localPrinterCollection.GetEnumerator()

    If localPrinterEnumerator.MoveNext() Then
        ' Get PrintQueue from first available printer
        printQueue = CType(localPrinterEnumerator.Current, PrintQueue)
    Else
        ' No printer exist, return null PrintTicket
        Return Nothing
    End If

    ' Get default PrintTicket from printer
    Dim printTicket As PrintTicket = printQueue.DefaultPrintTicket

    Dim printCapabilites As PrintCapabilities = printQueue.GetPrintCapabilities()

    ' Modify PrintTicket
    If printCapabilites.CollationCapability.Contains(Collation.Collated) Then
        printTicket.Collation = Collation.Collated
    End If

    If printCapabilites.DuplexingCapability.Contains(Duplexing.TwoSidedLongEdge) Then
        printTicket.Duplexing = Duplexing.TwoSidedLongEdge
    End If

    If printCapabilites.StaplingCapability.Contains(Stapling.StapleDualLeft) Then
        printTicket.Stapling = Stapling.StapleDualLeft
    End If

    Return printTicket
End Function ' end:GetPrintTicketFromPrinter()

```

PrintServer e PrintQueue

La classe [PrintServer](#) rappresenta un server di stampa di rete, mentre la classe [PrintQueue](#) rappresenta una stampante e la coda del processo di output a essa associata. Insieme, queste API consentono una gestione avanzata dei processi di stampa di un server. La classe [PrintServer](#), o una delle relative classi derivate, consente di gestire la classe [PrintQueue](#). Il metodo [AddJob](#) consente di inserire un nuovo processo di stampa nella coda.

L'esempio di codice riportato di seguito illustra come creare un oggetto [LocalPrintServer](#) e come accedere al relativo oggetto [PrintQueue](#) predefinito tramite il codice.

```

// ----- GetPrintXpsDocumentWriter() -----
/// <summary>
///   Returns an XpsDocumentWriter for the default print queue.</summary>
/// <returns>
///   An XpsDocumentWriter for the default print queue.</returns>
private XpsDocumentWriter GetPrintXpsDocumentWriter()
{
    // Create a local print server
    LocalPrintServer ps = new LocalPrintServer();

    // Get the default print queue
    PrintQueue pq = ps.DefaultPrintQueue;

    // Get an XpsDocumentWriter for the default print queue
    XpsDocumentWriter xpsdw = PrintQueue.CreateXpsDocumentWriter(pq);
    return xpsdw;
}// end:GetPrintXpsDocumentWriter()

```

```

' ----- GetPrintXpsDocumentWriter() -----
''' <summary>
'''   Returns an XpsDocumentWriter for the default print queue.</summary>
''' <returns>
'''   An XpsDocumentWriter for the default print queue.</returns>
Private Function GetPrintXpsDocumentWriter() As XpsDocumentWriter
    ' Create a local print server
    Dim ps As New LocalPrintServer()

    ' Get the default print queue
    Dim pq As PrintQueue = ps.DefaultPrintQueue

    ' Get an XpsDocumentWriter for the default print queue
    Dim xpsdw As XpsDocumentWriter = PrintQueue.CreateXpsDocumentWriter(pq)
    Return xpsdw
End Function ' end:GetPrintXpsDocumentWriter()

```

XpsDocumentWriter

Una [XpsDocumentWriter](#), con molti metodi [Write](#) e [WriteAsync](#), viene utilizzata per scrivere documenti XPS in un [PrintQueue](#). Ad esempio, il metodo [Write\(FixedPage, PrintTicket\)](#) viene utilizzato per restituire un documento XPS e [PrintTicket](#) in modo sincrono. Il metodo [WriteAsync\(FixedDocument, PrintTicket\)](#) viene utilizzato per restituire un documento XPS e [PrintTicket](#) in modo asincrono.

L'esempio riportato di seguito illustra come creare un oggetto [XpsDocumentWriter](#) tramite codice.

```

// ----- GetPrintXpsDocumentWriter() -----
/// <summary>
///   Returns an XpsDocumentWriter for the default print queue.</summary>
/// <returns>
///   An XpsDocumentWriter for the default print queue.</returns>
private XpsDocumentWriter GetPrintXpsDocumentWriter()
{
    // Create a local print server
    LocalPrintServer ps = new LocalPrintServer();

    // Get the default print queue
    PrintQueue pq = ps.DefaultPrintQueue;

    // Get an XpsDocumentWriter for the default print queue
    XpsDocumentWriter xpsdw = PrintQueue.CreateXpsDocumentWriter(pq);
    return xpsdw;
}// end:GetPrintXpsDocumentWriter()

```

```

' ----- GetPrintXpsDocumentWriter() -----
''' <summary>
''' Returns an XpsDocumentWriter for the default print queue.</summary>
''' <returns>
''' An XpsDocumentWriter for the default print queue.</returns>
Private Function GetPrintXpsDocumentWriter() As XpsDocumentWriter
    ' Create a local print server
    Dim ps As New LocalPrintServer()

    ' Get the default print queue
    Dim pq As PrintQueue = ps.DefaultPrintQueue

    ' Get an XpsDocumentWriter for the default print queue
    Dim xpsdw As XpsDocumentWriter = PrintQueue.CreateXpsDocumentWriter(pq)
    Return xpsdw
End Function ' end:GetPrintXpsDocumentWriter()

```

I metodi [AddJob](#) consentono anche di stampare in diversi modi. Vedere [Stampa di file XPS a livello di codice](#). per informazioni dettagliate.

Percorso di stampa GDI

Sebbene le applicazioni WPF supportino in modo nativo il percorso di stampa XPS, le applicazioni Win32 e Windows Forms possono avvalersi anche di alcune funzionalità XPS. Il driver della stampante XPS (XPSDrv) può convertire l'output basato su GDI in formato XPS. Per gli scenari avanzati, la conversione personalizzata del contenuto è supportata tramite [Microsoft XPS Document Converter \(MXDC\)](#). Analogamente, le applicazioni WPF possono anche restituire il percorso di stampa GDI chiamando uno dei metodi [Write](#) o [WriteAsync](#) della classe [XpsDocumentWriter](#) e designando una stampante non XpsDrv come coda di stampa di destinazione.

Per le applicazioni che non richiedono funzionalità o supporto XPS, il percorso di stampa GDI corrente rimane invariato.

- Per materiale di riferimento aggiuntivo sul percorso di stampa GDI e sulle varie opzioni di conversione XPS, vedere [Microsoft XPS Document Converter \(MXDC\)](#) e i [driver della stampante XPSDrv](#).

Modello di driver XPSDrv

Il percorso di stampa XPS migliora l'efficienza dello spooler utilizzando XPS come formato dello spooler di stampa nativo per la stampa su un driver o una stampante abilitata per XPS. Il processo di spooling semplificato elimina la necessità di generare un file di spool intermedio, ad esempio un file di dati EMF, prima dello spooling del documento. Grazie alle dimensioni del file di spool più piccolo, il percorso di stampa XPS può ridurre il traffico di rete e migliorare le prestazioni di stampa.

EMF è un formato chiuso che rappresenta l'output dell'applicazione come una serie di chiamate in GDI per i servizi di rendering. Diversamente da EMF, il formato di spooling XPS rappresenta il documento effettivo senza richiedere un'ulteriore interpretazione quando viene restituito un driver della stampante basato su XPS (XPSDrv). I driver possono agire direttamente sui dati nel formato. Questa funzionalità Elimina le conversioni dei dati e dello spazio colori necessarie quando si usano i file EMF e i driver di stampa basati su GDI.

Le dimensioni dei file di spooling vengono in genere ridotte quando si utilizzano documenti XPS destinati a un driver della stampante XPS (XPSDrv) rispetto ai rispettivi equivalenti EMF; Esistono tuttavia alcune eccezioni:

- Una grafica vettoriale molto complessa, a più livelli o scritta in modo inappropriato può presentare dimensioni maggiori rispetto alla relativa versione bitmap.
- Per scopi di visualizzazione, i file XPS incorporano tipi di carattere del dispositivo, nonché tipi di caratteri basati sul computer. Al contrario, i file di spooling GDI non incorporano alcun tipo di carattere del dispositivo. Tuttavia, entrambi i tipi di carattere presentano sottoinsiemi (vedere di seguito) e i driver della

stampante possono rimuovere i tipi di carattere del dispositivo prima di trasmettere il file alla stampante.

La riduzione delle dimensioni di spooling viene eseguita tramite molteplici meccanismi:

- **Incorporamento di sottoinsiemi di tipi di carattere.** Solo i caratteri usati nel documento effettivo vengono archiviati nel file XPS.
- **Supporto grafico avanzato.** Il supporto nativo per le primitive di trasparenza e sfumatura evita la rasterizzazione del contenuto nel documento XPS.
- **Identificazione di risorse comuni.** Le risorse usate più volte (ad esempio un'immagine che rappresenta un logo aziendale) sono considerate risorse condivise e vengono caricate solo una volta.
- **Compressione ZIP.** Tutti i documenti XPS utilizzano la compressione ZIP.

Vedere anche

- [PrintDialog](#)
- [XpsDocumentWriter](#)
- [XpsDocument](#)
- [PrintTicket](#)
- [PrintCapabilities](#)
- [PrintServer](#)
- [PrintQueue](#)
- [Procedure relative alle proprietà](#)
- [Documenti in WPF](#)
- [Documenti XPS](#)
- [Serializzazione e archiviazione di documenti](#)
- [Microsoft XPS Document Converter \(MXDC\)](#)

Procedure relative alla stampa

30/10/2019 • 2 minutes to read • [Edit Online](#)

Negli argomenti di questa sezione viene illustrato come utilizzare le funzionalità di gestione del sistema di stampa e stampa incluse in Windows Presentation Foundation (WPF), nonché il nuovo percorso di stampa XPS (XML Paper Specification).

Contenuto della sezione

[Richiamare una finestra di dialogo di stampa](#)

Istruzioni per il markup XAML per dichiarare un oggetto finestra di dialogo di stampa di Microsoft Windows e usare il codice per richiamare la finestra di dialogo dall'interno di un'applicazione Windows Presentation Foundation (WPF).

[Duplicare una stampante](#)

Istruzioni per l'installazione di una seconda coda di stampa con esattamente le stesse proprietà di una coda di stampa esistente.

[Diagnosticare processi di stampa problematici](#)

Istruzioni per l'utilizzo delle proprietà delle code di stampa e dei processi di stampa per la diagnosi di un processo di stampa che non viene stampato.

[Verificare l'eventuale possibilità di eseguire un processo di stampa in questo preciso momento](#)

Istruzioni per l'utilizzo delle proprietà delle code di stampa e dei processi di stampa per stabilire a livello di codice le ore del giorno in cui il processo può essere stampato.

[Enumerare un sottoinsieme di code di stampa](#)

Istruzioni per la generazione di un elenco di stampanti con determinate caratteristiche.

[Ottenere le proprietà dell'oggetto del sistema di stampa senza reflection](#)

Istruzioni su come individuare le proprietà dell'oggetto del sistema di stampa di runtime e i relativi tipi.

[Stampa di file XPS a livello di codice](#)

Istruzioni per la stampa rapida dei file XPS (XML Paper Specification) senza la necessità di un interfaccia utente.

[Verificare lo stato delle stampanti da postazione remota](#)

Istruzioni per la creazione di un'utilità che rileverà le stampanti per individuare quelle che riscontrano un inceppamento della carta o altri problemi.

[Convalidare e unire PrintTicket](#)

Istruzioni per verificare che un ticket di stampa sia valido e che non sia richiesta alcuna operazione non supportata dalla stampante.

Vedere anche

- [System.Printing](#)
- [System.Printing.IndexedProperties](#)
- [System.Printing.Interop](#)
- [Panoramica della stampa](#)
- [Documenti in WPF](#)
- [Documenti XPS](#)

Procedura: richiamare una finestra di dialogo di stampa

10/02/2020 • 3 minutes to read • [Edit Online](#)

Per consentire la stampa dall'applicazione, è possibile creare e aprire semplicemente un oggetto [PrintDialog](#).

Esempio

Il controllo [PrintDialog](#) fornisce un singolo punto di ingresso per Interfaccia utente, configurazione e invio di processi XPS. Il controllo è facile da usare ed è possibile crearne un'istanza usando Extensible Application Markup Language (XAML) markup o codice. Nell'esempio seguente viene illustrato come creare un'istanza di e come aprire il controllo nel codice e come stamparlo. Viene inoltre illustrato come assicurarsi che la finestra di dialogo fornisca all'utente la possibilità di impostare un intervallo specifico di pagine. Nel codice di esempio si presuppone che esista un file FixedDocumentSequence.XPS nella radice dell'unità C:.

```
private void InvokePrint(object sender, RoutedEventArgs e)
{
    // Create the print dialog object and set options
    PrintDialog pDialog = new PrintDialog();
    pDialog.PageRangeSelection = PageRangeSelection.AllPages;
    pDialog.UserPageRangeEnabled = true;

    // Display the dialog. This returns true if the user presses the Print button.
    Nullable<Boolean> print = pDialog.ShowDialog();
    if (print == true)
    {
        XpsDocument xpsDocument = new XpsDocument("C:\\\\FixedDocumentSequence.xps", FileAccess.ReadWrite);
        FixedDocumentSequence fixedDocSeq = xpsDocument.GetFixedDocumentSequence();
        pDialog.PrintDocument(fixedDocSeq.DocumentPaginator, "Test print job");
    }
}
```

```
Private Sub InvokePrint(ByVal sender As Object, ByVal e As RoutedEventArgs)
    ' Create the print dialog object and set options
    Dim pDialog As New PrintDialog()
    pDialog.PageRangeSelection = PageRangeSelection.AllPages
    pDialog.UserPageRangeEnabled = True

    ' Display the dialog. This returns true if the user presses the Print button.
    Dim print? As Boolean = pDialog.ShowDialog()
    If print = True Then
        Dim xpsDocument As New XpsDocument("C:\\\\FixedDocumentSequence.xps", FileAccess.ReadWrite)
        Dim fixedDocSeq As FixedDocumentSequence = xpsDocument.GetFixedDocumentSequence()
        pDialog.PrintDocument(fixedDocSeq.DocumentPaginator, "Test print job")
    End If
End Sub
```

Una volta aperta la finestra di dialogo, gli utenti saranno in grado di effettuare una selezione tra le stampanti installate nel computer. Hanno inoltre la possibilità di selezionare il writer di [documenti XPS Microsoft](#) per creare un file XPS (XML Paper Specification) anziché stampare.

NOTE

Il controllo [System.Windows.Controls.PrintDialog](#) di WPF, descritto in questo argomento, non dovrebbe essere confuso con il componente [System.Windows.Forms.PrintDialog](#) di Windows Forms.

In modo esplicito, è possibile usare il metodo [PrintDocument](#) senza mai aprire la finestra di dialogo. In questo senso, il controllo può essere usato come componente di stampa non visibile. Per motivi di prestazioni, tuttavia, sarebbe preferibile usare il metodo [AddJob](#) o uno dei molti metodi [Write](#) e [WriteAsync](#) della [XpsDocumentWriter](#). Per ulteriori informazioni, vedere la pagina relativa alla [stampa di file XPS](#) e a livello di codice.

Vedere anche

- [PrintDialog](#)
- [Documenti in WPF](#)
- [Panoramica della stampa](#)
- [Microsoft XPS Document Writer](#)

Procedura: Clonare una stampante

23/10/2019 • 4 minutes to read • [Edit Online](#)

La maggior parte delle aziende, a un certo punto acquisti più stampanti dello stesso modello. In genere, queste vengono installate con le impostazioni di configurazione praticamente identici. L'installazione di ogni stampante può richiedere molto tempo e tendente all'errore. Il [System.Printing.IndexedProperties](#) dello spazio dei nomi e il [InstallPrintQueue](#) classe esposte con Microsoft .NET Framework consente di installare immediatamente un numero qualsiasi di code di stampa aggiuntive che sono stati clonati da una coda di stampa esistente.

Esempio

Nell'esempio seguente, una coda di stampa secondo viene clonata da una coda di stampa esistente. Il secondo è diverso dal primo solo nel relativo nome, posizione, porta e lo stato condiviso. I passaggi principali per eseguire questa operazione sono come indicato di seguito.

1. Creare un [PrintQueue](#) oggetto per la stampante esistente che sta per essere clonata.
2. Creare un [PrintPropertyDictionary](#) dal [PropertiesCollection](#) del [PrintQueue](#). Il [Value](#) proprietà di ogni voce in questo dizionario è un oggetto di uno dei tipi derivati da [PrintProperty](#). Esistono due modi per impostare il valore di una voce in questo dizionario.
 - Usare il dizionario [rimuovere](#) e [Add](#) metodi per rimuovere la voce e quindi aggiungerlo nuovamente con il valore desiderato.
 - Usare il dizionario [SetProperty](#) (metodo).

L'esempio seguente illustra entrambe le direzioni.

3. Creare un [PrintBooleanProperty](#) dell'oggetto e impostare relativi [Name](#) su "IsShared" e la relativa [Value](#) a `true`.
4. Usare il [PrintBooleanProperty](#) oggetto come valore del [PrintPropertyDictionary](#) della voce "IsShared".
5. Creare un [PrintStringProperty](#) dell'oggetto e impostare relativi [Name](#) a "Nomecondivisione" e la relativa [Value](#) al relativo [String](#).
6. Usare la [PrintStringProperty](#) oggetto come valore del [PrintPropertyDictionary](#) della voce "Nomecondivisione".
7. Creare un'altra [PrintStringProperty](#) dell'oggetto e impostare relativi [Name](#) "Posizione" e la relativa [Value](#) al relativo [String](#).
8. Utilizzare la seconda [PrintStringProperty](#) come valore dell'oggetto di [PrintPropertyDictionary](#) dell'ingresso "Percorso".
9. Creare una matrice di [Strings](#). Ogni elemento è il nome di una porta nel server.
10. Usare [InstallPrintQueue](#) per installare la nuova stampante con i nuovi valori.

Seguito è riportato un esempio.

```

LocalPrintServer myLocalPrintServer = new LocalPrintServer(PrintSystemDesiredAccess.AdministrateServer);
PrintQueue sourcePrintQueue = myLocalPrintServer.DefaultPrintQueue;
PrintPropertyDictionary myPrintProperties = sourcePrintQueue.PropertiesCollection;

// Share the new printer using Remove/Add methods
PrintBooleanProperty shared = new PrintBooleanProperty("IsShared", true);
myPrintProperties.Remove("IsShared");
myPrintProperties.Add("IsShared", shared);

// Give the new printer its share name using SetProperty method
PrintStringProperty theShareName = new PrintStringProperty("ShareName", "\"Son of " + sourcePrintQueue.Name
+ "\"");
myPrintProperties SetProperty("ShareName", theShareName);

// Specify the physical location of the new printer using Remove/Add methods
PrintStringProperty theLocation = new PrintStringProperty("Location", "the supply room");
myPrintProperties.Remove("Location");
myPrintProperties.Add("Location", theLocation);

// Specify the port for the new printer
String[] port = new String[] { "COM1:" };

// Install the new printer on the local print server
PrintQueue clonedPrinter = myLocalPrintServer.InstallPrintQueue("My clone of " + sourcePrintQueue.Name, "Xerox
WCP 35 PS", port, "WinPrint", myPrintProperties);
myLocalPrintServer.Commit();

// Report outcome
Console.WriteLine("{0} in {1} has been installed and shared as {2}", clonedPrinter.Name,
clonedPrinter.Location, clonedPrinter.ShareName);
Console.WriteLine("Press Return to continue ...");
Console.ReadLine();

```

```

Dim myLocalPrintServer As New LocalPrintServer(PrintSystemDesiredAccess.AdministrateServer)
Dim sourcePrintQueue As PrintQueue = myLocalPrintServer.DefaultPrintQueue
Dim myPrintProperties As PrintPropertyDictionary = sourcePrintQueue.PropertiesCollection

' Share the new printer using Remove/Add methods
Dim [shared] As New PrintBooleanProperty("IsShared", True)
myPrintProperties.Remove("IsShared")
myPrintProperties.Add("IsShared", [shared])

' Give the new printer its share name using SetProperty method
Dim theShareName As New PrintStringProperty("ShareName", """Son of " & sourcePrintQueue.Name & """")
myPrintProperties SetProperty("ShareName", theShareName)

' Specify the physical location of the new printer using Remove/Add methods
Dim theLocation As New PrintStringProperty("Location", "the supply room")
myPrintProperties.Remove("Location")
myPrintProperties.Add("Location", theLocation)

' Specify the port for the new printer
Dim port() As String = { "COM1:" }

' Install the new printer on the local print server
Dim clonedPrinter As PrintQueue = myLocalPrintServer.InstallPrintQueue("My clone of " & sourcePrintQueue.Name,
"Xerox WCP 35 PS", port, "WinPrint", myPrintProperties)
myLocalPrintServer.Commit()

' Report outcome
Console.WriteLine("{0} in {1} has been installed and shared as {2}", clonedPrinter.Name,
clonedPrinter.Location, clonedPrinter.ShareName)
Console.WriteLine("Press Return to continue ...")
Console.ReadLine()

```

Vedere anche

- [System.Printing.IndexedProperties](#)
- [PrintPropertyDictionary](#)
- [LocalPrintServer](#)
- [PrintQueue](#)
- [DictionaryEntry](#)
- [Documenti in WPF](#)
- [Panoramica della stampa](#)

Procedura: Diagnosticare un processo di stampa problematico

23/10/2019 • 19 minutes to read • [Edit Online](#)

Gli amministratori di rete fanno spesso fronte ai reclami degli utenti su processi di stampa lenti o che non vengono eseguiti affatto. Il set completo di proprietà del processo di stampa esposto nelle API di Microsoft .NET Framework fornisce un mezzo per eseguire una diagnosi remota rapida dei processi di stampa.

Esempio

Di seguito sono indicati i passaggi principali per la creazione di questo tipo di utilità.

1. Identificare il processo di stampa oggetto del reclamo dell'utente. Gli utenti spesso non sono in grado di eseguire questa verifica con precisione. Non conoscono i nomi dei server di stampa o delle stampanti. Possono descrivere il percorso della stampante con una terminologia diversa da quella usata nell'impostazione della [Location](#) relativa proprietà. È consigliabile quindi generare un elenco dei processi inviati dall'utente. Nel caso ci siano più processi, la comunicazione tra l'utente e l'amministratore del sistema di stampa può essere quindi usata per individuare il processo problematico. Di seguito sono indicati i passaggi secondari.
 - a. Ottenere un elenco di tutti i server di stampa.
 - b. Riprodurre il ciclo dei server per eseguire una query sulle code di stampa.
 - c. In ogni passaggio del ciclo del server, riprodurre il ciclo di tutte le code del server per eseguire query sui processi.
 - d. In ogni passaggio del ciclo di code, eseguire il ciclo dei processi e raccogliere informazioni di identificazione su quelli che sono stati inviati dall'utente del reclamo.
2. Quando viene individuato il processo di stampa problematico, esaminare le proprietà rilevanti per vedere quale potrebbe essere la causa del problema. Ad esempio, se si tratta di un errore di stato del processo oppure la stampante che gestisce la coda è passata alla modalità offline prima della stampa.

Il codice riportato di seguito include una serie di esempi di codice. Il primo esempio di codice contiene il ciclo tra le code di stampa. Vedere il passaggio 1c precedente. La variabile `myPrintQueues` è l'[PrintQueueCollection](#) oggetto per il server di stampa corrente.

L'esempio di codice inizia con l'aggiornamento dell'oggetto coda di [PrintQueue.Refresh](#) stampa corrente con. Ciò garantisce che le proprietà dell'oggetto rappresentano accuratamente lo stato della stampante fisica rappresentata. L'applicazione ottiene quindi la raccolta di processi di stampa attualmente nella coda di stampa tramite [GetPrintJobInfoCollection](#).

Successivamente, l'applicazione esegue il [PrintSystemJobInfo](#) ciclo della raccolta e confronta [Submitter](#) ogni proprietà con l'alias dell'utente che si lamenta. Se ci sono corrispondenze, l'applicazione aggiunge le informazioni di identificazione sul processo alla stringa che verrà visualizzata. Le variabili `userName` e `jobList` vengono inizializzate prima nell'applicazione.

```

for each (PrintQueue^ pq in myPrintQueues)
{
    pq->Refresh();
    PrintJobInfoCollection^ jobs = pq->GetPrintJobInfoCollection();
    for each (PrintSystemJobInfo^ job in jobs)
    {
        // Since the user may not be able to articulate which job is problematic,
        // present information about each job the user has submitted.
        if (job->Submitter == userName)
        {
            atLeastOne = true;
            jobList = jobList + "\nServer:" + line;
            jobList = jobList + "\n\tQueue:" + pq->Name;
            jobList = jobList + "\n\tLocation:" + pq->Location;
            jobList = jobList + "\n\t\tJob: " + job->JobName + " ID: " + job->JobIdentifier;
        }
    }
}

```

```

foreach (PrintQueue pq in myPrintQueues)
{
    pq.Refresh();
    PrintJobInfoCollection jobs = pq.GetPrintJobInfoCollection();
    foreach (PrintSystemJobInfo job in jobs)
    {
        // Since the user may not be able to articulate which job is problematic,
        // present information about each job the user has submitted.
        if (job.Submitter == userName)
        {
            atLeastOne = true;
            jobList = jobList + "\nServer:" + line;
            jobList = jobList + "\n\tQueue:" + pq.Name;
            jobList = jobList + "\n\tLocation:" + pq.Location;
            jobList = jobList + "\n\t\tJob: " + job.JobName + " ID: " + job.JobIdentifier;
        }
    }// end for each print job
}// end for each print queue

```

```

For Each pq As PrintQueue In myPrintQueues
    pq.Refresh()
    Dim jobs As PrintJobInfoCollection = pq.GetPrintJobInfoCollection()
    For Each job As PrintSystemJobInfo In jobs
        ' Since the user may not be able to articulate which job is problematic,
        ' present information about each job the user has submitted.
        If job.Submitter = userName Then
            atLeastOne = True
            jobList = jobList & vbCrLf & "Server:" & line
            jobList = jobList & vbCrLf & vbTab & "Queue:" & pq.Name
            jobList = jobList & vbCrLf & vbTab & "Location:" & pq.Location
            jobList = jobList & vbCrLf & vbTab & "Job: " & job.JobName & " ID: " & job.JobIdentifier
        End If
    Next job ' end for each print job

    Next pq ' end for each print queue

```

L'esempio di codice successivo preleva l'applicazione al passaggio 2. Vedere sopra. Il processo problematico è stato identificato e l'applicazione richiede le informazioni che lo identificheranno. Da queste informazioni vengono creati [PrintServer](#) oggetti [PrintQueue](#), e [PrintSystemJobInfo](#).

A questo punto l'applicazione contiene una struttura ramificata corrispondente ai due modi di controllo dello stato del processo di stampa:

- È possibile leggere i flag della [JobStatus](#) proprietà che è di tipo [PrintJobStatus](#)
- È possibile leggere tutte [IsBlocked](#) le proprietà rilevanti, ad esempio e. [IsInError](#)

Questo esempio illustra entrambi i metodi, quindi all'utente è stato chiesto in precedenza come metodo da usare e ha risposto con "Y" se volesse usare i flag della [JobStatus](#) proprietà. Per i dettagli dei due metodi, vedere di seguito. Infine, l'applicazione usa un metodo denominato **ReportQueueAndJobAvailability** per segnalare se il processo può essere stampato all'ora indicata. Questo metodo viene illustrato in [Individuare se è possibile eseguire o meno un processo di stampa all'orario indicato](#).

```
// When the problematic print job has been identified, enter information about it.
Console::Write("\nEnter the print server hosting the job (including leading slashes \\\\): " + "\n(press
Return for the current computer \\\{0\}): ", Environment::MachineName);
String^ pServer = Console::ReadLine();
if (pServer == "")
{
    pServer = "\\\\" + Environment::MachineName;
}
Console::Write("\nEnter the print queue hosting the job: ");
String^ pQueue = Console::ReadLine();
Console::Write("\nEnter the job ID: ");
Int16 jobID = Convert::ToInt16(Console::ReadLine());

// Create objects to represent the server, queue, and print job.
PrintServer^ hostingServer = gcnew PrintServer(pServer, PrintSystemDesiredAccess::AdministrateServer);
PrintQueue^ hostingQueue = gcnew PrintQueue(hostingServer, pQueue,
PrintSystemDesiredAccess::AdministratePrinter);
PrintSystemJobInfo^ theJob = hostingQueue->GetJob(jobID);

if (useAttributesResponse == "Y")
{
    TroubleSpotter::SpotTroubleUsingJobAttributes(theJob);
    // TroubleSpotter class is defined in the complete example.
} else
{
    TroubleSpotter::SpotTroubleUsingProperties(theJob);
}

TroubleSpotter::ReportQueueAndJobAvailability(theJob);
```

```

// When the problematic print job has been identified, enter information about it.
Console.WriteLine("\nEnter the print server hosting the job (including leading slashes \\\\"": " +
"\n(press Return for the current computer \\{0}): ", Environment.MachineName);
String pServer = Console.ReadLine();
if (pServer == "")
{
    pServer = "\\\\" + Environment.MachineName;
}
Console.WriteLine("\nEnter the print queue hosting the job: ");
String pQueue = Console.ReadLine();
Console.WriteLine("\nEnter the job ID: ");
Int16 jobID = Convert.ToInt16(Console.ReadLine());

// Create objects to represent the server, queue, and print job.
PrintServer hostingServer = new PrintServer(pServer, PrintSystemDesiredAccess.AdministrateServer);
PrintQueue hostingQueue = new PrintQueue(hostingServer, pQueue, PrintSystemDesiredAccess.AdministratePrinter);
PrintSystemJobInfo theJob = hostingQueue.GetJob(jobID);

if (useAttributesResponse == "Y")
{
    TroubleSpotter.SpotTroubleUsingJobAttributes(theJob);
    // TroubleSpotter class is defined in the complete example.
}
else
{
    TroubleSpotter.SpotTroubleUsingProperties(theJob);
}

TroubleSpotter.ReportQueueAndJobAvailability(theJob);

```

```

' When the problematic print job has been identified, enter information about it.
Console.WriteLine(vbLf & "Enter the print server hosting the job (including leading slashes \\): " & vbCrLf & "
(press Return for the current computer \\{0}): ", Environment.MachineName)
Dim pServer As String = Console.ReadLine()
If pServer = "" Then
    pServer = "\\" & Environment.MachineName
End If
Console.WriteLine(vbLf & "Enter the print queue hosting the job: ")
Dim pQueue As String = Console.ReadLine()
Console.WriteLine(vbLf & "Enter the job ID: ")
Dim jobID As Int16 = Convert.ToInt16(Console.ReadLine())

' Create objects to represent the server, queue, and print job.
Dim hostingServer As New PrintServer(pServer, PrintSystemDesiredAccess.AdministrateServer)
Dim hostingQueue As New PrintQueue(hostingServer, pQueue, PrintSystemDesiredAccess.AdministratePrinter)
Dim theJob As PrintSystemJobInfo = hostingQueue.GetJob(jobID)

If useAttributesResponse = "Y" Then
    TroubleSpotter.SpotTroubleUsingJobAttributes(theJob)
    ' TroubleSpotter class is defined in the complete example.
Else
    TroubleSpotter.SpotTroubleUsingProperties(theJob)
End If

TroubleSpotter.ReportQueueAndJobAvailability(theJob)

```

Per controllare lo stato del processo di stampa usando i [JobStatus](#) flag della proprietà, controllare ogni flag pertinente per verificare se è impostato. Il metodo standard per verificare se un bit è impostato in un set di flag di bit consiste nell'eseguire un'operazione di AND logico con il set di flag come uno degli operandi e il flag stesso come altro operando. Poiché il flag stesso ha un solo bit impostato, il risultato dell'AND logico è che, al massimo, è impostato quello stesso bit. Per verificare se lo è o meno, confrontare il risultato dell'AND logico con il flag stesso. Per ulteriori informazioni, vedere [PrintJobStatus!` operatore & \(C# riferimento\)e. FlagsAttribute](#)

Per ogni attributo il cui bit è impostato, il codice lo riporta allo schermo della console e talvolta suggerisce un modo per rispondere. Il metodo **HandlePausedJob** che viene chiamato se il processo o la coda è in pausa è illustrato di seguito.

```
// Check for possible trouble states of a print job using the flags of the JobStatus property
static void SpotTroubleUsingJobAttributes (PrintSystemJobInfo^ theJob)
{
    if (((theJob->JobStatus & PrintJobStatus::Blocked) == PrintJobStatus::Blocked)
    {
        Console::WriteLine("The job is blocked.");
    }
    if (((theJob->JobStatus & PrintJobStatus::Completed) == PrintJobStatus::Completed)
        ||
        ((theJob->JobStatus & PrintJobStatus::Printed) == PrintJobStatus::Printed))
    {
        Console::WriteLine("The job has finished. Have user recheck all output bins and be sure the correct
printer is being checked.");
    }
    if (((theJob->JobStatus & PrintJobStatus::Deleted) == PrintJobStatus::Deleted)
        ||
        ((theJob->JobStatus & PrintJobStatus::Deleting) == PrintJobStatus::Deleting))
    {
        Console::WriteLine("The user or someone with administration rights to the queue has deleted the job. It
must be resubmitted.");
    }
    if ((theJob->JobStatus & PrintJobStatus::Error) == PrintJobStatus::Error)
    {
        Console::WriteLine("The job has errored.");
    }
    if ((theJob->JobStatus & PrintJobStatus::Offline) == PrintJobStatus::Offline)
    {
        Console::WriteLine("The printer is offline. Have user put it online with printer front panel.");
    }
    if ((theJob->JobStatus & PrintJobStatus::PaperOut) == PrintJobStatus::PaperOut)
    {
        Console::WriteLine("The printer is out of paper of the size required by the job. Have user add paper.");
    }
    if (((theJob->JobStatus & PrintJobStatus::Paused) == PrintJobStatus::Paused)
        ||
        ((theJob->HostingPrintQueue->QueueStatus & PrintQueueStatus::Paused) == PrintQueueStatus::Paused))
    {
        HandlePausedJob(theJob);
        //HandlePausedJob is defined in the complete example.
    }

    if ((theJob->JobStatus & PrintJobStatus::Printing) == PrintJobStatus::Printing)
    {
        Console::WriteLine("The job is printing now.");
    }
    if ((theJob->JobStatus & PrintJobStatus::Spooling) == PrintJobStatus::Spooling)
    {
        Console::WriteLine("The job is spooling now.");
    }
    if ((theJob->JobStatus & PrintJobStatus::UserIntervention) == PrintJobStatus::UserIntervention)
    {
        Console::WriteLine("The printer needs human intervention.");
    }
};

};
```

```

// Check for possible trouble states of a print job using the flags of the JobStatus property
internal static void SpotTroubleUsingJobAttributes(PrintSystemJobInfo theJob)
{
    if (((theJob.JobStatus & PrintJobStatus.Blocked) == PrintJobStatus.Blocked)
    {
        Console.WriteLine("The job is blocked.");
    }
    if (((theJob.JobStatus & PrintJobStatus.Completed) == PrintJobStatus.Completed)
        ||
        ((theJob.JobStatus & PrintJobStatus.Printed) == PrintJobStatus.Printed))
    {
        Console.WriteLine("The job has finished. Have user recheck all output bins and be sure the correct
printer is being checked.");
    }
    if (((theJob.JobStatus & PrintJobStatus.Deleted) == PrintJobStatus.Deleted)
        ||
        ((theJob.JobStatus & PrintJobStatus.Deleting) == PrintJobStatus.Deleting))
    {
        Console.WriteLine("The user or someone with administration rights to the queue has deleted the job. It
must be resubmitted.");
    }
    if ((theJob.JobStatus & PrintJobStatus.Error) == PrintJobStatus.Error)
    {
        Console.WriteLine("The job has errored.");
    }
    if ((theJob.JobStatus & PrintJobStatus.Offline) == PrintJobStatus.Offline)
    {
        Console.WriteLine("The printer is offline. Have user put it online with printer front panel.");
    }
    if ((theJob.JobStatus & PrintJobStatus.PaperOut) == PrintJobStatus.PaperOut)
    {
        Console.WriteLine("The printer is out of paper of the size required by the job. Have user add
paper.");
    }

    if (((theJob.JobStatus & PrintJobStatus.Paused) == PrintJobStatus.Paused)
        ||
        ((theJob.HostingPrintQueue.QueueStatus & PrintQueueStatus.Paused) == PrintQueueStatus.Paused))
    {
        HandlePausedJob(theJob);
        //HandlePausedJob is defined in the complete example.
    }

    if ((theJob.JobStatus & PrintJobStatus.Printing) == PrintJobStatus.Printing)
    {
        Console.WriteLine("The job is printing now.");
    }
    if ((theJob.JobStatus & PrintJobStatus.Spooling) == PrintJobStatus.Spooling)
    {
        Console.WriteLine("The job is spooling now.");
    }
    if ((theJob.JobStatus & PrintJobStatus.UserIntervention) == PrintJobStatus.UserIntervention)
    {
        Console.WriteLine("The printer needs human intervention.");
    }
}
//end SpotTroubleUsingJobAttributes

```

```

' Check for possible trouble states of a print job using the flags of the JobStatus property
Friend Shared Sub SpotTroubleUsingJobAttributes(ByVal theJob As PrintSystemJobInfo)
    If (theJob.JobStatus And PrintJobStatus.Blocked) = PrintJobStatus.Blocked Then
        Console.WriteLine("The job is blocked.")
    End If
    If ((theJob.JobStatus And PrintJobStatus.Completed) = PrintJobStatus.Completed) OrElse ((theJob.JobStatus
And PrintJobStatus.Printed) = PrintJobStatus.Printed) Then
        Console.WriteLine("The job has finished. Have user recheck all output bins and be sure the correct
printer is being checked.")
    End If
    If ((theJob.JobStatus And PrintJobStatus.Deleted) = PrintJobStatus.Deleted) OrElse ((theJob.JobStatus And
PrintJobStatus.Deleting) = PrintJobStatus.Deleting) Then
        Console.WriteLine("The user or someone with administration rights to the queue has deleted the job. It
must be resubmitted.")
    End If
    If (theJob.JobStatus And PrintJobStatus.Error) = PrintJobStatus.Error Then
        Console.WriteLine("The job has errored.")
    End If
    If (theJob.JobStatus And PrintJobStatus.Offline) = PrintJobStatus.Offline Then
        Console.WriteLine("The printer is offline. Have user put it online with printer front panel.")
    End If
    If (theJob.JobStatus And PrintJobStatus.PaperOut) = PrintJobStatus.PaperOut Then
        Console.WriteLine("The printer is out of paper or the size required by the job. Have user add paper.")
    End If

    If ((theJob.JobStatus And PrintJobStatus.Paused) = PrintJobStatus.Paused) OrElse
((theJob.HostingPrintQueue.QueueStatus And PrintQueueStatus.Paused) = PrintQueueStatus.Paused) Then
        HandlePausedJob(theJob)
        'HandlePausedJob is defined in the complete example.
    End If

    If (theJob.JobStatus And PrintJobStatus.Printing) = PrintJobStatus.Printing Then
        Console.WriteLine("The job is printing now.")
    End If
    If (theJob.JobStatus And PrintJobStatus.Spooling) = PrintJobStatus.Spooling Then
        Console.WriteLine("The job is spooling now.")
    End If
    If (theJob.JobStatus And PrintJobStatus.UserIntervention) = PrintJobStatus.UserIntervention Then
        Console.WriteLine("The printer needs human intervention.")
    End If

End Sub

```

Per controllare lo stato del processo di stampa usando proprietà separate, è sufficiente leggere tutte le proprietà e, se la proprietà è `true`, visualizzarla nello schermo della console e suggerire possibilmente una modalità di risposta. Il metodo **HandlePausedJob** che viene chiamato se il processo o la coda è in pausa è illustrato di seguito.

```

// Check for possible trouble states of a print job using its properties
static void SpotTroubleUsingProperties (PrintSystemJobInfo^ theJob)
{
    if (theJob->IsBlocked)
    {
        Console::WriteLine("The job is blocked.");
    }
    if (theJob->IsCompleted || theJob->IsPrinted)
    {
        Console::WriteLine("The job has finished. Have user recheck all output bins and be sure the correct
printer is being checked.");
    }
    if (theJob->IsDeleted || theJob->IsDeleting)
    {
        Console::WriteLine("The user or someone with administration rights to the queue has deleted the job. It
must be resubmitted.");
    }
    if (theJob->IsInError)
    {
        Console::WriteLine("The job has errored.");
    }
    if (theJob->IsOffline)
    {
        Console::WriteLine("The printer is offline. Have user put it online with printer front panel.");
    }
    if (theJob->IsPaperOut)
    {
        Console::WriteLine("The printer is out of paper of the size required by the job. Have user add paper.");
    }

    if (theJob->IsPaused || theJob->HostingPrintQueue->IsPaused)
    {
        HandlePausedJob(theJob);
        //HandlePausedJob is defined in the complete example.
    }

    if (theJob->IsPrinting)
    {
        Console::WriteLine("The job is printing now.");
    }
    if (theJob->IsSpooling)
    {
        Console::WriteLine("The job is spooling now.");
    }
    if (theJob->IsUserInterventionRequired)
    {
        Console::WriteLine("The printer needs human intervention.");
    }
};


```

```

// Check for possible trouble states of a print job using its properties
internal static void SpotTroubleUsingProperties(PrintSystemJobInfo theJob)
{
    if (theJob.IsBlocked)
    {
        Console.WriteLine("The job is blocked.");
    }
    if (theJob.IsCompleted || theJob.IsPrinted)
    {
        Console.WriteLine("The job has finished. Have user recheck all output bins and be sure the correct
printer is being checked.");
    }
    if (theJob.IsDeleted || theJob.IsDeleting)
    {
        Console.WriteLine("The user or someone with administration rights to the queue has deleted the job. It
must be resubmitted.");
    }
    if (theJob.IsInError)
    {
        Console.WriteLine("The job has errored.");
    }
    if (theJob.IsOffline)
    {
        Console.WriteLine("The printer is offline. Have user put it online with printer front panel.");
    }
    if (theJob.IsPaperOut)
    {
        Console.WriteLine("The printer is out of paper of the size required by the job. Have user add
paper.");
    }

    if (theJob.IsPaused || theJob.HostingPrintQueue.IsPaused)
    {
        HandlePausedJob(theJob);
        //HandlePausedJob is defined in the complete example.
    }

    if (theJob.IsPrinting)
    {
        Console.WriteLine("The job is printing now.");
    }
    if (theJob.IsSpooling)
    {
        Console.WriteLine("The job is spooling now.");
    }
    if (theJob.IsUserInterventionRequired)
    {
        Console.WriteLine("The printer needs human intervention.");
    }
}//end SpotTroubleUsingProperties

```

```

' Check for possible trouble states of a print job using its properties
Friend Shared Sub SpotTroubleUsingProperties(ByVal theJob As PrintSystemJobInfo)
    If theJob.IsBlocked Then
        Console.WriteLine("The job is blocked.")
    End If
    If theJob.IsCompleted OrElse theJob.IsPrinted Then
        Console.WriteLine("The job has finished. Have user recheck all output bins and be sure the correct
printer is being checked.")
    End If
    If theJob.IsDeleted OrElse theJob.IsDeleting Then
        Console.WriteLine("The user or someone with administration rights to the queue has deleted the job. It
must be resubmitted.")
    End If
    If theJob.IsInError Then
        Console.WriteLine("The job has errored.")
    End If
    If theJob.IsOffline Then
        Console.WriteLine("The printer is offline. Have user put it online with printer front panel.")
    End If
    If theJob.IsPaperOut Then
        Console.WriteLine("The printer is out of paper of the size required by the job. Have user add paper.")
    End If

    If theJob.IsPaused OrElse theJob.HostingPrintQueue.IsPaused Then
        HandlePausedJob(theJob)
        'HandlePausedJob is defined in the complete example.
    End If

    If theJob.IsPrinting Then
        Console.WriteLine("The job is printing now.")
    End If
    If theJob.IsSpooling Then
        Console.WriteLine("The job is spooling now.")
    End If
    If theJob.isUserInterventionRequired Then
        Console.WriteLine("The printer needs human intervention.")
    End If

End Sub

```

Il metodo **HandlePausedJob** consente all'utente dell'applicazione di riprendere i processi sospesi in modalità remota. Poiché potrebbe esserci un motivo valido per cui è stata sospesa la coda di stampa, il metodo inizia richiedendo all'utente conferma sul riavvio. Se la risposta è "Y", viene chiamato [PrintQueue.Resume](#) il metodo.

Successivamente, verrà richiesto all'utente di decidere se deve essere ripreso il processo stesso, nel caso sia stato sospeso indipendentemente dalla coda di stampa. (Confrontare [PrintQueue.IsPaused](#) e [PrintSystemJobInfo.IsPaused](#)) Se la risposta è "Y", [PrintSystemJobInfo.Resume](#) viene chiamato il metodo; in caso contrario [Cancel](#), viene chiamato.

```

static void HandlePausedJob (PrintSystemJobInfo^ theJob)
{
    // If there's no good reason for the queue to be paused, resume it and
    // give user choice to resume or cancel the job.
    Console::WriteLine("The user or someone with administrative rights to the queue" + "\nhas paused the job or
queue." + "\nResume the queue? (Has no effect if queue is not paused.)" + "\nEnter \"Y\" to resume, otherwise
press return: ");
    String^ resume = Console::ReadLine();
    if (resume == "Y")
    {
        theJob->HostingPrintQueue->Resume();

        // It is possible the job is also paused. Find out how the user wants to handle that.
        Console::WriteLine("Does user want to resume print job or cancel it?" + "\nEnter \"Y\" to resume (any
other key cancels the print job): ");
        String^ userDecision = Console::ReadLine();
        if (userDecision == "Y")
        {
            theJob->Resume();
        } else
        {
            theJob->Cancel();
        }
    }
};


```

```

internal static void HandlePausedJob(PrintSystemJobInfo theJob)
{
    // If there's no good reason for the queue to be paused, resume it and
    // give user choice to resume or cancel the job.
    Console.WriteLine("The user or someone with administrative rights to the queue" +
        "\nhas paused the job or queue." +
        "\nResume the queue? (Has no effect if queue is not paused.)" +
        "\nEnter \"Y\" to resume, otherwise press return: ");
    String resume = Console.ReadLine();
    if (resume == "Y")
    {
        theJob.HostingPrintQueue.Resume();

        // It is possible the job is also paused. Find out how the user wants to handle that.
        Console.WriteLine("Does user want to resume print job or cancel it?" +
            "\nEnter \"Y\" to resume (any other key cancels the print job): ");
        String userDecision = Console.ReadLine();
        if (userDecision == "Y")
        {
            theJob.Resume();
        } else
        {
            theJob.Cancel();
        }
    }
//end if the queue should be resumed
}//end HandlePausedJob

```

```

Friend Shared Sub HandlePausedJob(ByVal theJob As PrintSystemJobInfo)
    ' If there's no good reason for the queue to be paused, resume it and
    ' give user choice to resume or cancel the job.
    Console.WriteLine("The user or someone with administrative rights to the queue" & vbCrLf & "has paused the
job or queue." & vbCrLf & "Resume the queue? (Has no effect if queue is not paused.)" & vbCrLf & "Enter ""Y"" to
resume, otherwise press return: ")
    Dim [resume] As String = Console.ReadLine()
    If [resume] = "Y" Then
        theJob.HostingPrintQueue.Resume()

        ' It is possible the job is also paused. Find out how the user wants to handle that.
        Console.WriteLine("Does user want to resume print job or cancel it?" & vbCrLf & "Enter ""Y"" to resume
(any other key cancels the print job): ")
        Dim userDecision As String = Console.ReadLine()
        If userDecision = "Y" Then
            theJob.Resume()
        Else
            theJob.Cancel()
        End If
    End If 'end if the queue should be resumed

End Sub

```

Vedere anche

- [PrintJobStatus](#)
- [PrintSystemJobInfo](#)
- [FlagsAttribute](#)
- [PrintQueue](#)
- [Operatore & \(C# riferimento\)](#)
- [Documenti in WPF](#)
- [Panoramica della stampa](#)

Procedura: Scoprire se è possibile eseguire un processo di stampa a quest'ora del giorno

23/10/2019 • 12 minutes to read • [Edit Online](#)

Le code di stampa non sono sempre disponibili per 24 ore al giorno. Hanno le proprietà dell'ora di inizio e di fine che possono essere impostate in modo da renderle non disponibili in determinati orari del giorno. Questa funzionalità può essere usata, ad esempio, per riservare una stampante per l'uso esclusivo di un determinato reparto dopo le 17:00. Tale reparto avrebbe una coda diversa per la manutenzione della stampante rispetto ad altri reparti. La coda per gli altri reparti verrà impostata in modo da non essere disponibile dopo le 17:00, mentre la coda per il reparto favorito potrebbe essere impostata in modo che sia sempre disponibile.

Inoltre, i processi di stampa possono essere impostati in modo da essere stampabili solo entro un intervallo di tempo specificato.

Le [PrintQueue](#) classi [PrintSystemJobInfo](#) e esposte nelle API di Microsoft .NET Framework forniscono un mezzo per verificare in remoto se un determinato processo di stampa è in grado di stampare in una determinata coda all'ora corrente.

Esempio

Nell'esempio seguente è riportato un esempio in grado di diagnosticare i problemi relativi a un processo di stampa.

Esistono due passaggi principali per questo tipo di funzione, come indicato di seguito.

1. Leggere le [StartTimeOfDay](#) proprietà [UntilTimeOfDay](#) e dell'oggetto [PrintQueue](#) per determinare se l'ora corrente è compresa tra di esse.
2. Leggere le [StartTimeOfDay](#) proprietà [UntilTimeOfDay](#) e dell'oggetto [PrintSystemJobInfo](#) per determinare se l'ora corrente è compresa tra di esse.

Tuttavia, le complicazioni derivano dal fatto che [DateTime](#) queste proprietà non sono oggetti. Sono [Int32](#) invece oggetti che esprimono l'ora del giorno come numero di minuti dalla mezzanotte. Inoltre, questa non è la mezzanotte nel fuso orario corrente, ma la mezzanotte UTC (Coordinated Universal Time).

Il primo esempio di codice presenta il metodo statico [ReportQueueAndJobAvailability](#), a cui viene [PrintSystemJobInfo](#) passato un oggetto e chiama metodi helper per determinare se il processo è in grado di stampare nell'ora corrente e, in caso contrario, quando è in grado di stampare. Si noti che [PrintQueue](#) un oggetto non viene passato al metodo. Questo è dovuto al [PrintSystemJobInfo](#) fatto che include un riferimento alla coda nella [HostingPrintQueue](#) relativa proprietà.

I metodi subordinati includono il metodo [ReportAvailabilityAtThisTime](#) di overload che può assumere [PrintQueue](#) o [PrintSystemJobInfo](#) come parametro. Esiste anche un [TimeConverter](#).

[ConvertToLocalHumanReadableTime](#). Tutti questi metodi sono descritti di seguito.

Il metodo [ReportQueueAndJobAvailability](#) inizia con il controllo per verificare se la coda o il processo di stampa non è disponibile in questo momento. Se una di esse non è disponibile, verifica se la coda non è disponibile. Se non è disponibile, il metodo segnala questo fatto e l'ora in cui la coda diventerà nuovamente disponibile. Quindi controlla il processo e, se non è disponibile, indica l'intervallo di tempo successivo quando è in grado di stampare. Infine, il metodo segnala la prima volta che il processo è in grado di stampare. Questo è il successivo di due volte.

- Ora di disponibilità successiva della coda di stampa.
- Ora successiva disponibile per il processo di stampa.

Quando si segnalano ora del giorno `ToShortTimeString`, viene chiamato anche il metodo poiché questo metodo evita gli anni, i mesi e i giorni dall'output. Non è possibile limitare la disponibilità di una coda di stampa o di un processo di stampa a determinati anni, mesi o giorni.

```
static void ReportQueueAndJobAvailability (PrintSystemJobInfo^ theJob)
{
    if (!(ReportAvailabilityAtThisTime(theJob->HostingPrintQueue) && ReportAvailabilityAtThisTime(theJob)))
    {
        if (!ReportAvailabilityAtThisTime(theJob->HostingPrintQueue))
        {
            Console::WriteLine("\nThat queue is not available at this time of day." + "\nJobs in the queue will
start printing again at {0}", TimeConverter::ConvertToLocalHumanReadableTime(theJob->HostingPrintQueue-
>StartTimeOfDay).ToString());
            // TimeConverter class is defined in the complete sample
        }
        if (!ReportAvailabilityAtThisTime(theJob))
        {
            Console::WriteLine("\nThat job is set to print only between {0} and {1}",
TimeConverter::ConvertToLocalHumanReadableTime(theJob->StartTimeOfDay).ToString(),
TimeConverter::ConvertToLocalHumanReadableTime(theJob->UntilTimeOfDay).ToString());
        }
        Console::WriteLine("\nThe job will begin printing as soon as it reaches the top of the queue after:");
        if (theJob->StartTimeOfDay > theJob->HostingPrintQueue->StartTimeOfDay)
        {
            Console::WriteLine(TimeConverter::ConvertToLocalHumanReadableTime(theJob-
>StartTimeOfDay).ToString());
        } else
        {
            Console::WriteLine(TimeConverter::ConvertToLocalHumanReadableTime(theJob->HostingPrintQueue-
>StartTimeOfDay).ToString());
        }
    }
};
```

```

internal static void ReportQueueAndJobAvailability(PrintSystemJobInfo theJob)
{
    if (!(ReportAvailabilityAtThisTime(theJob.HostingPrintQueue) && ReportAvailabilityAtThisTime(theJob)))
    {
        if (!ReportAvailabilityAtThisTime(theJob.HostingPrintQueue))
        {
            Console.WriteLine("\nThat queue is not available at this time of day." +
                "\nJobs in the queue will start printing again at {0}",

TimeConverter.ConvertToLocalHumanReadableTime(theJob.HostingPrintQueue.StartTimeOfDay).ToShortTimeString());
            // TimeConverter class is defined in the complete sample
        }

        if (!ReportAvailabilityAtThisTime(theJob))
        {
            Console.WriteLine("\nThat job is set to print only between {0} and {1}",
                TimeConverter.ConvertToLocalHumanReadableTime(theJob.StartTimeOfDay).ToShortTimeString(),
                TimeConverter.ConvertToLocalHumanReadableTime(theJob.UntilTimeOfDay).ToShortTimeString());
        }
        Console.WriteLine("\nThe job will begin printing as soon as it reaches the top of the queue after:");
        if (theJob.StartTimeOfDay > theJob.HostingPrintQueue.StartTimeOfDay)
        {

Console.WriteLine(TimeConverter.ConvertToLocalHumanReadableTime(theJob.StartTimeOfDay).ToShortTimeString());
        }
        else
        {

Console.WriteLine(TimeConverter.ConvertToLocalHumanReadableTime(theJob.HostingPrintQueue.StartTimeOfDay).ToShortTimeString());
        }
    }
}

}//end if at least one is not available
}//end ReportQueueAndJobAvailability

```

```

Friend Shared Sub ReportQueueAndJobAvailability(ByVal theJob As PrintSystemJobInfo)
    If Not(ReportAvailabilityAtThisTime(theJob.HostingPrintQueue) AndAlso
ReportAvailabilityAtThisTime(theJob)) Then
        If Not ReportAvailabilityAtThisTime(theJob.HostingPrintQueue) Then
            Console.WriteLine(vbLf & "That queue is not available at this time of day." & vbCrLf & "Jobs in the
queue will start printing again at {0}",
TimeConverter.ConvertToLocalHumanReadableTime(theJob.HostingPrintQueue.StartTimeOfDay).ToShortTimeString())
            ' TimeConverter class is defined in the complete sample
        End If

        If Not ReportAvailabilityAtThisTime(theJob) Then
            Console.WriteLine(vbLf & "That job is set to print only between {0} and {1}",
TimeConverter.ConvertToLocalHumanReadableTime(theJob.StartTimeOfDay).ToShortTimeString(),
TimeConverter.ConvertToLocalHumanReadableTime(theJob.UntilTimeOfDay).ToShortTimeString())
            End If
        Console.WriteLine(vbLf & "The job will begin printing as soon as it reaches the top of the queue
after:")
        If theJob.StartTimeOfDay > theJob.HostingPrintQueue.StartTimeOfDay Then

Console.WriteLine(TimeConverter.ConvertToLocalHumanReadableTime(theJob.StartTimeOfDay).ToShortTimeString())
        Else

Console.WriteLine(TimeConverter.ConvertToLocalHumanReadableTime(theJob.HostingPrintQueue.StartTimeOfDay).ToShortTimeString())
        End If
    End If 'end if at least one is not available
End Sub

```

I due overload del metodo **ReportAvailabilityAtThisTime** sono identici, ad eccezione del tipo passato, quindi

solo la [PrintQueue](#) versione è riportata di seguito.

NOTE

Il fatto che i metodi siano identici, ad eccezione del tipo, genera la domanda sul motivo per cui l'esempio non crea un metodo generico `<ReportAvailabilityAtThisTime T >`. Il motivo è che un metodo di questo tipo deve essere limitato a una classe con le proprietà **StartTimeOfDay** e **UntilTimeOfDay** chiamate dal metodo, ma un metodo generico può essere limitato solo a una singola classe e l'unica classe comune a entrambi e nell'albero di ereditarietà è [PrintSystemObject](#) che non dispone di tali proprietà. [PrintSystemJobInfo PrintQueue](#)

Il metodo **ReportAvailabilityAtThisTime** (presentato nell'esempio di codice riportato di seguito) inizia con **Boolean** l'inizializzazione `true` di una variabile **Sentinel** in. Se la coda non è `false` disponibile, verrà reimpostata su.

Successivamente, il metodo verifica se i tempi di inizio e di fine "fino a" sono identici. In caso affermativo, la coda è sempre disponibile, quindi il metodo `true` restituisce.

Se la coda non è sempre disponibile, il metodo usa la proprietà statica [UtcNow](#) per ottenere l'ora corrente [DateTime](#) come oggetto. Non è necessaria l'ora locale perché le [StartTimeOfDay](#) proprietà e [UntilTimeOfDay](#) si trovano in ora UTC.

Tuttavia, queste due proprietà non [DateTime](#) sono oggetti. Sono [Int32](#) espresse l'ora come numero di minuti-dopo-UTC-mezzanotte. Quindi, dobbiamo convertire l' [DateTime](#) oggetto in minuti, dopo la mezzanotte. Al termine, il metodo controlla semplicemente se "Now" è compreso tra l'inizio della coda e il valore "until", imposta la sentinella su false se "Now" non è compresa tra le due volte e restituisce **Sentinel**.

```
static Boolean ReportAvailabilityAtThisTime (PrintQueue^ pq)
{
    Boolean available = true;
    if (pq->StartTimeOfDay != pq->UntilTimeOfDay)
    {
        DateTime utcNow = DateTime::UtcNow;
        Int32 utcNowAsMinutesAfterMidnight = (utcNow.TimeOfDay.Hours * 60) + utcNow.TimeOfDay.Minutes;

        // If now is not within the range of available times . . .
        if (!((pq->StartTimeOfDay < utcNowAsMinutesAfterMidnight) && (utcNowAsMinutesAfterMidnight < pq->UntilTimeOfDay)))
        {
            available = false;
        }
    }
    return available;
};
```

```

private static Boolean ReportAvailabilityAtThisTime(PrintQueue pq)
{
    Boolean available = true;
    if (pq.StartTimeOfDay != pq.UntilTimeOfDay) // If the printer is not available 24 hours a day
    {
        DateTime utcNow = DateTime.UtcNow;
        Int32 utcNowAsMinutesAfterMidnight = (utcNow.TimeOfDay.Hours * 60) + utcNow.TimeOfDay.Minutes;

        // If now is not within the range of available times . . .
        if (!(pq.StartTimeOfDay < utcNowAsMinutesAfterMidnight)
            && (utcNowAsMinutesAfterMidnight < pq.UntilTimeOfDay))
        {
            available = false;
        }
    }
    return available;
}//end ReportAvailabilityAtThisTime

```

```

Private Shared Function ReportAvailabilityAtThisTime(ByVal pq As PrintQueue) As Boolean
    Dim available As Boolean = True
    If pq.StartTimeOfDay <> pq.UntilTimeOfDay Then ' If the printer is not available 24 hours a day
        Dim utcNow As Date = Date.UtcNow
        Dim utcNowAsMinutesAfterMidnight As Int32 = (utcNow.TimeOfDay.Hours * 60) + utcNow.TimeOfDay.Minutes

        ' If now is not within the range of available times . . .
        If Not((pq.StartTimeOfDay < utcNowAsMinutesAfterMidnight) AndAlso (utcNowAsMinutesAfterMidnight <
pq.UntilTimeOfDay)) Then
            available = False
        End If
    End If
    Return available
End Function 'end ReportAvailabilityAtThisTime

```

Il metodo **TimeConverter. ConvertToLocalHumanReadableTime** (presentato nell'esempio di codice seguente) non usa metodi introdotti con Microsoft .NET Framework, quindi la discussione è breve. Il metodo dispone di un'attività di conversione doppia: deve prendere un numero intero che esprime minuti-dopo-mezzanotte e convertirlo in un tempo leggibile ed è necessario convertirlo nell'ora locale. Questa operazione viene eseguita creando innanzitutto un [DateTime](#) oggetto impostato sulla mezzanotte UTC, quindi viene utilizzato il [AddMinutes](#) metodo per aggiungere i minuti passati al metodo. Viene restituito un nuovo [DateTime](#) oggetto che esprime l'ora originale passata al metodo. Il [ToLocalTime](#) metodo converte quindi questo oggetto nell'ora locale.

```

private ref class TimeConverter {

internal:
    static DateTime ConvertToLocalHumanReadableTime (Int32 timeInMinutesAfterUTCMidnight)
    {
        // Construct a UTC midnight object.
        // Must start with current date so that the local Daylight Savings system, if any, will be taken into
account.

        DateTime utcNow = DateTime::UtcNow;
        DateTime utcMidnight = DateTime(utcNow.Year, utcNow.Month, utcNow.Day, 0, 0, 0, DateTimeKind::Utc);

        // Add the minutes passed into the method in order to get the intended UTC time.
        Double minutesAfterUTCMidnight = ((Double)timeInMinutesAfterUTCMidnight);
        DateTime utcTime = utcMidnight.AddMinutes(minutesAfterUTCMidnight);

        // Convert to local time.
        DateTime localTime = utcTime.ToLocalTime();

        return localTime;
    };
}

```

```

class TimeConverter
{
    // Convert time as minutes past UTC midnight into human readable time in local time zone.
    internal static DateTime ConvertToLocalHumanReadableTime(Int32 timeInMinutesAfterUTCMidnight)
    {
        // Construct a UTC midnight object.
        // Must start with current date so that the local Daylight Savings system, if any, will be taken into
account.

        DateTime utcNow = DateTime.UtcNow;
        DateTime utcMidnight = new DateTime(utcNow.Year, utcNow.Month, utcNow.Day, 0, 0, 0, DateTimeKind.Utc);

        // Add the minutes passed into the method in order to get the intended UTC time.
        Double minutesAfterUTCMidnight = (Double)timeInMinutesAfterUTCMidnight;
        DateTime utcTime = utcMidnight.AddMinutes(minutesAfterUTCMidnight);

        // Convert to local time.
        DateTime localTime = utcTime.ToLocalTime();

        return localTime;
    } // end ConvertToLocalHumanReadableTime
} //end TimeConverter class

```

```

Friend Class TimeConverter
    ' Convert time as minutes past UTC midnight into human readable time in local time zone.
    Friend Shared Function ConvertToLocalHumanReadableTime(ByVal timeInMinutesAfterUTCMidnight As Int32) As
Date
        ' Construct a UTC midnight object.
        ' Must start with current date so that the local Daylight Savings system, if any, will be taken into
account.
        Dim utcNow As Date = Date.UtcNow
        Dim utcMidnight As New Date(utcNow.Year, utcNow.Month, utcNow.Day, 0, 0, 0, DateTimeKind.Utc)

        ' Add the minutes passed into the method in order to get the intended UTC time.
        Dim minutesAfterUTCMidnight As Double = CType(timeInMinutesAfterUTCMidnight, Double)
        Dim utcTime As Date = utcMidnight.AddMinutes(minutesAfterUTCMidnight)

        ' Convert to local time.
        Dim localTime As Date = utcTime.ToLocalTime()

        Return localTime

    End Function ' end ConvertToLocalHumanReadableTime

End Class

```

Vedere anche

- [DateTime](#)
- [PrintSystemJobInfo](#)
- [PrintQueue](#)
- [Documenti in WPF](#)
- [Panoramica della stampa](#)

Procedura: enumerare un sottoinsieme di code di stampa

10/02/2020 • 3 minutes to read • [Edit Online](#)

Una situazione comune affrontata dai professionisti IT (Information Technology) che gestiscono un set di stampanti a livello aziendale consiste nel generare un elenco di stampanti con determinate caratteristiche. Questa funzionalità viene fornita dal metodo [GetPrintQueues](#) di un oggetto [PrintServer](#) e dall'enumerazione [EnumeratedPrintQueueTypes](#).

Esempio

Nell'esempio seguente il codice inizia creando una matrice di flag che specificano le caratteristiche delle code di stampa che si desidera elencare. In questo esempio, si stanno cercando le code di stampa installate localmente nel server di stampa e condivise. L'enumerazione [EnumeratedPrintQueueTypes](#) offre molte altre possibilità.

Il codice crea quindi un oggetto [LocalPrintServer](#), una classe derivata da [PrintServer](#). Il server di stampa locale è il computer in cui è in esecuzione l'applicazione.

L'ultimo passaggio significativo consiste nel passare la matrice al metodo [GetPrintQueues](#).

Infine, i risultati vengono presentati all'utente.

```
// Specify that the list will contain only the print queues that are installed as local and are shared
array<System::Printing::EnumeratedPrintQueueTypes>^ enumerationFlags =
{EnumeratedPrintQueueTypes::Local, EnumeratedPrintQueueTypes::Shared};

LocalPrintServer^ printServer = gcnew LocalPrintServer();

//Use the enumerationFlags to filter out unwanted print queues
PrintQueueCollection^ printQueuesOnLocalServer = printServer->GetPrintQueues(enumerationFlags);

Console::WriteLine("These are your shared, local print queues:\n\n");

for each (PrintQueue^ printer in printQueuesOnLocalServer)
{
    Console::WriteLine("\tThe shared printer " + printer->Name + " is located at " + printer->Location + "\n");
}
Console::WriteLine("Press enter to continue.");
Console::ReadLine();
```

```

// Specify that the list will contain only the print queues that are installed as local and are shared
EnumeratedPrintQueueTypes[] enumerationFlags = {EnumeratedPrintQueueTypes.Local,
                                                EnumeratedPrintQueueTypes.Shared};

LocalPrintServer printServer = new LocalPrintServer();

//Use the enumerationFlags to filter out unwanted print queues
PrintQueueCollection printQueuesOnLocalServer = printServer.GetPrintQueues(enumerationFlags);

Console.WriteLine("These are your shared, local print queues:\n\n");

foreach (PrintQueue printer in printQueuesOnLocalServer)
{
    Console.WriteLine("\tThe shared printer " + printer.Name + " is located at " + printer.Location + "\n");
}
Console.WriteLine("Press enter to continue.");
Console.ReadLine();

```

```

' Specify that the list will contain only the print queues that are installed as local and are shared
Dim enumerationFlags() As EnumeratedPrintQueueTypes = {EnumeratedPrintQueueTypes.Local,
                                                       EnumeratedPrintQueueTypes.Shared}

Dim printServer As New LocalPrintServer()

'Use the enumerationFlags to filter out unwanted print queues
Dim printQueuesOnLocalServer As PrintQueueCollection = printServer.GetPrintQueues(enumerationFlags)

Console.WriteLine("These are your shared, local print queues:" & vbCrLf & vbCrLf)

For Each printer As PrintQueue In printQueuesOnLocalServer
    Console.WriteLine(vbTab & "The shared printer " & printer.Name & " is located at " & printer.Location &
vbLf)
Next printer
Console.WriteLine("Press enter to continue.")
Console.ReadLine()

```

Questo esempio può essere esteso con il ciclo di `foreach` che esegue l'analisi di ogni coda di stampa. Ad esempio, è possibile escludere le stampanti che non supportano la stampa su due lati, facendo in modo che il ciclo chiama il metodo [GetPrintCapabilities](#) della coda di stampa e testando il valore restituito per la presenza di duplexing.

Vedere anche

- [GetPrintQueues](#)
- [PrintServer](#)
- [LocalPrintServer](#)
- [EnumeratedPrintQueueTypes](#)
- [PrintQueue](#)
- [GetPrintCapabilities](#)
- [Documenti in WPF](#)
- [Panoramica della stampa](#)
- [Microsoft XPS Document Writer](#)

Procedura: Ottenere le proprietà dell'oggetto del sistema di stampa senza reflection

23/10/2019 • 2 minutes to read • [Edit Online](#)

Uso della reflection per specificare i dettagli delle proprietà (e i tipi di tali proprietà) su un oggetto può rallentare le prestazioni dell'applicazione. Il [System.Printing.IndexedProperties](#) dello spazio dei nomi fornisce un mezzo per ottenere queste informazioni con l'uso della reflection.

Esempio

I passaggi per eseguire questa operazione sono i seguenti.

1. Creare un'istanza del tipo. Nell'esempio seguente, il tipo è il [PrintQueue](#) tipo fornito con Microsoft .NET Framework, ma quasi identico codice dovrebbe funzionare per i tipi che derivano dal [PrintSystemObject](#).
2. Creare un [PrintPropertyDictionary](#) dalla proprietà del tipo [PropertiesCollection](#). Il [Value](#) proprietà di ogni voce in questo dizionario è un oggetto di uno dei tipi derivati da [PrintProperty](#).
3. Enumerare i membri del dizionario. Per ognuno di essi, procedere come segue.
4. Il valore di ogni voce per l'upcast [PrintProperty](#) e usarlo per creare un [PrintProperty](#) oggetto.
5. Ottenerne il tipo dei [Value](#) di ogni il [PrintProperty](#) oggetto.

```
// Enumerate the properties, and their types, of a queue without using Reflection
LocalPrintServer localPrintServer = new LocalPrintServer();
PrintQueue defaultPrintQueue = LocalPrintServer.GetDefaultPrintQueue();

PrintPropertyDictionary printQueueProperties = defaultPrintQueue.PropertiesCollection;

Console.WriteLine("These are the properties, and their types, of {0}, a {1}", defaultPrintQueue.Name,
defaultPrintQueue.GetType().ToString() +"\n");

foreach (DictionaryEntry entry in printQueueProperties)
{
    PrintProperty property = (PrintProperty)entry.Value;

    if (property.Value != null)
    {
        Console.WriteLine(property.Name + "\t(Type: {0})", property.Value.GetType().ToString());
    }
}
Console.WriteLine("\n\nPress Return to continue...");
Console.ReadLine();
```

```

' Enumerate the properties, and their types, of a queue without using Reflection
Dim localPrintServer As New LocalPrintServer()
Dim defaultPrintQueue As PrintQueue = LocalPrintServer.GetDefaultPrintQueue()

Dim printQueueProperties As PrintPropertyDictionary = defaultPrintQueue.PropertiesCollection

Console.WriteLine("These are the properties, and their types, of {0}, a {1}", defaultPrintQueue.Name,
defaultPrintQueue.GetType().ToString() + vbCrLf)

For Each entry As DictionaryEntry In printQueueProperties
    Dim [property] As PrintProperty = CType(entry.Value, PrintProperty)

    If [property].Value IsNot Nothing Then
        Console.WriteLine([property].Name & vbTab & "(Type: {0})", [property].Value.GetType().ToString())
    End If
Next entry
Console.WriteLine(vbLf & vbCrLf & "Press Return to continue...")
Console.ReadLine()

```

Vedere anche

- [PrintProperty](#)
- [PrintSystemObject](#)
- [System.Printing.IndexedProperties](#)
- [PrintPropertyDictionary](#)
- [LocalPrintServer](#)
- [PrintQueue](#)
- [DictionaryEntry](#)
- [Documenti in WPF](#)
- [Panoramica della stampa](#)

Procedura: stampa di file XPS a livello di codice

25/11/2019 • 11 minutes to read • [Edit Online](#)

È possibile utilizzare un overload del metodo [AddJob](#) per stampare i file XPS (XML Paper Specification) senza aprire un [PrintDialog](#) o, in generale, qualsiasi interfaccia utente.

È anche possibile stampare file XPS usando i molti metodi [XpsDocumentWriter.Write](#) e [XpsDocumentWriter.WriteAsync](#). Per ulteriori informazioni, vedere la pagina relativa alla [stampa di un documento XPS](#).

Un altro modo per stampare XPS consiste nell'usare i metodi [PrintDialog.PrintDocument](#) o [PrintDialog.PrintVisual](#). Vedere [Richiamare una finestra di dialogo Stampa](#).

Esempio

I passaggi principali per l'uso del metodo di [AddJob\(String, String, Boolean\)](#) a tre parametri sono i seguenti. L'esempio riportato di seguito offre delle informazioni dettagliate.

1. Determinare se la stampante è una stampante XPSDrv. Per ulteriori informazioni su XPSDrv, vedere [Cenni preliminari sulla stampa](#).
2. Se la stampante non è una stampante XPSDrv, impostare l'apartment del thread a thread singolo.
3. Creare un'istanza di un server di stampa e un oggetto coda di stampa.
4. Chiamare il metodo, specificando il nome di un processo, il file da stampare e un flag di [Boolean](#) che indica se la stampante è una stampante XPSDrv.

Nell'esempio seguente viene illustrato come stampare in batch tutti i file XPS in una directory. Anche se l'applicazione richiede all'utente di specificare la directory, il metodo a tre parametri [AddJob\(String, String, Boolean\)](#) non richiede una interfaccia utente. Può essere usato in qualsiasi percorso di codice in cui si hanno un nome file e un percorso XPS che è possibile passare ad esso.

L'overload [AddJob\(String, String, Boolean\)](#) a tre parametri di [AddJob](#) deve essere eseguito in un Apartment a thread singolo ogni volta che viene `false` il parametro [Boolean](#), che deve essere quando viene utilizzata una stampante non XPSDrv. Tuttavia, lo stato dell'Apartment predefinito per .NET è più thread. Questa impostazione predefinita deve essere annullata poiché nell'esempio si presuppone una stampante non XPSDrv.

Ci sono due modi per modificare il valore predefinito. Un modo consiste nel aggiungere semplicemente il [STAThreadAttribute](#) (ovvero "`[System.STAThreadAttribute()]`") appena sopra la prima riga del metodo di [Main](#) dell'applicazione (in genere "`static void Main(string[] args)`"). Tuttavia, molte applicazioni richiedono che il metodo [Main](#) disponga di uno stato Apartment a thread multipli, quindi è disponibile un secondo metodo: inserire la chiamata a [AddJob\(String, String, Boolean\)](#) in un thread separato il cui stato Apartment è impostato su [STA](#) con [SetApartmentState](#). Nell'esempio seguente si usa la seconda tecnica.

Di conseguenza, l'esempio inizia creando un'istanza di un oggetto [Thread](#) e passandogli un metodo [PrintXPS](#) come parametro di [ThreadStart](#). Il metodo [PrintXPS](#) viene definito più avanti nell'esempio. Il thread viene quindi impostato su un Apartment a thread singolo. Il solo codice rimanente del metodo [Main](#) avvia il nuovo thread.

La sostanza dell'esempio è nel metodo `static BatchXPSPrinter`. Dopo aver creato un server di stampa e una coda, il metodo richiede all'utente una directory contenente i file XPS. Dopo aver convalidato l'esistenza della directory e la presenza di *file con estensione XPS, il metodo aggiunge ogni file di questo tipo alla coda di stampa. Nell'esempio si presuppone che la stampante non sia XPSDrv, quindi si passa `false` all'ultimo

parametro del metodo `AddJob(String, String, Boolean)`. Per questo motivo, il metodo convaliderà il markup XPS nel file prima di tentare di convertirlo nel linguaggio di descrizione della pagina della stampante. Se la convalida non riesce, viene generata un'eccezione. Il codice di esempio rileverà l'eccezione, invierà una notifica all'utente e quindi procederà per elaborare il file XPS successivo.

```
class Program
{
    [System.MTAThreadAttribute()] // Added for clarity, but this line is redundant because MTA is the default.
    static void Main(string[] args)
    {
        // Create the secondary thread and pass the printing method for
        // the constructor's ThreadStart delegate parameter. The BatchXPSPrinter
        // class is defined below.
        Thread printingThread = new Thread(BatchXPSPrinter.PrintXPS);

        // Set the thread that will use PrintQueue.AddJob to single threading.
        printingThread.SetApartmentState(ApartmentState.STA);

        // Start the printing thread. The method passed to the Thread
        // constructor will execute.
        printingThread.Start();
    }//end Main
}//end Program class

public class BatchXPSPrinter
{
    public static void PrintXPS()
    {
        // Create print server and print queue.
        LocalPrintServer localPrintServer = new LocalPrintServer();
        PrintQueue defaultPrintQueue = LocalPrintServer.GetDefaultPrintQueue();

        // Prompt user to identify the directory, and then create the directory object.
        Console.Write("Enter the directory containing the XPS files: ");
        String directoryPath = Console.ReadLine();
        DirectoryInfo dir = new DirectoryInfo(directoryPath);

        // If the user mistyped, end the thread and return to the Main thread.
        if (!dir.Exists)
        {
            Console.WriteLine("There is no such directory.");
        }
        else
        {
            // If there are no XPS files in the directory, end the thread
            // and return to the Main thread.
            if (dir.GetFiles("*.xps").Length == 0)
            {
                Console.WriteLine("There are no XPS files in the directory.");
            }
            else
            {
                Console.WriteLine("\nJobs will now be added to the print queue.");
                Console.WriteLine("If the queue is not paused and the printer is working, jobs will begin printing.");

                // Batch process all XPS files in the directory.
                foreach (FileInfo f in dir.GetFiles("*.xps"))
                {
                    String nextFile = directoryPath + "\\\" + f.Name;
                    Console.WriteLine("Adding {0} to queue.", nextFile);

                    try
                    {
                        // Print the Xps file while providing XPS validation and progress notifications.
                        PrintSystemJobInfo xpsPrintJob = defaultPrintQueue.AddJob(f.Name, nextFile, false);
                    }
                    catch (Exception ex)
                    {
                        Console.WriteLine(ex.Message);
                    }
                }
            }
        }
    }
}
```

```

        }
        catch (PrintJobException e)
        {
            Console.WriteLine("\n\t{0} could not be added to the print queue.", f.Name);
            if (e.InnerException.Message == "File contains corrupted data.")
            {
                Console.WriteLine("\tIt is not a valid XPS file. Use the isXPS Conformance Tool
to debug it.");
            }
            Console.WriteLine("\tContinuing with next XPS file.\n");
        }
    }// end for each XPS file
}//end if there are no XPS files in the directory
}//end if the directory does not exist

Console.WriteLine("Press Enter to end program.");
Console.ReadLine();
}// end PrintXPS method
}// end BatchXPSPrinter class

```

```

Friend Class Program
<System.MTAThreadAttribute()>
Shared Sub Main(ByVal args() As String) ' Added for clarity, but this line is redundant because MTA is
the default.
    ' Create the secondary thread and pass the printing method for
    ' the constructor's ThreadStart delegate parameter. The BatchXPSPrinter
    ' class is defined below.
    Dim printingThread As New Thread(AddressOf BatchXPSPrinter.PrintXPS)

    ' Set the thread that will use PrintQueue.AddJob to single threading.
    printingThread.SetApartmentState(ApartmentState.STA)

    ' Start the printing thread. The method passed to the Thread
    ' constructor will execute.
    printingThread.Start()

End Sub

End Class

Public Class BatchXPSPrinter
Public Shared Sub PrintXPS()
    ' Create print server and print queue.
    Dim localPrintServer As New LocalPrintServer()
    Dim defaultPrintQueue As PrintQueue = LocalPrintServer.GetDefaultPrintQueue()

    ' Prompt user to identify the directory, and then create the directory object.
    Console.Write("Enter the directory containing the XPS files: ")
    Dim directoryPath As String = Console.ReadLine()
    Dim dir As New DirectoryInfo(directoryPath)

    ' If the user mistyped, end the thread and return to the Main thread.
    If Not dir.Exists Then
        Console.WriteLine("There is no such directory.")
    Else
        ' If there are no XPS files in the directory, end the thread
        ' and return to the Main thread.
        If dir.GetFiles("*.xps").Length = 0 Then
            Console.WriteLine("There are no XPS files in the directory.")
        Else
            Console.WriteLine(vbLf & "Jobs will now be added to the print queue.")
            Console.WriteLine("If the queue is not paused and the printer is working, jobs will begin
printing.")

            ' Batch process all XPS files in the directory.
            For Each f As FileInfo In dir.GetFiles("*.xps")
                Dim nextFile As String = directoryPath & "\" & f.Name

```

```

        Console.WriteLine("Adding {0} to queue.", nextFile)

    Try
        ' Print the Xps file while providing XPS validation and progress notifications.
        Dim xpsPrintJob As PrintSystemJobInfo = defaultPrintQueue.AddJob(f.Name, nextFile,
    False)
    Catch e As PrintJobException
        Console.WriteLine(vbLf & vbTab & "{0} could not be added to the print queue.",
    f.Name)
        If e.InnerException.Message = "File contains corrupted data." Then
            Console.WriteLine(vbTab & "It is not a valid XPS file. Use the isXPS Conformance
Tool to debug it.")
        End If
        Console.WriteLine(vbTab & "Continuing with next XPS file." & vbCrLf)
    End Try

    Next f ' end for each XPS file

    End If 'end if there are no XPS files in the directory

    End If 'end if the directory does not exist

    Console.WriteLine("Press Enter to end program.")
    Console.ReadLine()

End Sub

End Class

```

Se si usa una stampante XPSDrv, allora è possibile impostare il parametro finale su `true`. In tal caso, poiché XPS è il linguaggio di descrizione della pagina della stampante, il metodo invierà il file alla stampante senza convalidarlo né convertirlo in un altro linguaggio di descrizione della pagina. Se si è incerti in fase di progettazione se l'applicazione utilizzerà una stampante XPSDrv, è possibile modificare l'applicazione in modo che legga la `IsXpsDevice` proprietà e il ramo in base a quanto rilevato.

Poiché inizialmente saranno disponibili alcune stampanti XPSDrv immediatamente dopo il rilascio di Windows Vista e Microsoft .NET Framework, potrebbe essere necessario mascherare una stampante non XPSDrv come stampante XPSDrv. A tale scopo, aggiungere Pipelineconfig.xml all'elenco dei file nella seguente chiave del Registro di sistema del computer che esegue l'applicazione:

`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Print\Environments\Windows NT x86\Drivers\Version-3\ <PrinterName> \DependentFiles`

dove `<PrinterName>` è qualsiasi coda di stampa. Il computer deve quindi essere riavviato.

Questo travestimento consentirà di passare `true` come parametro finale di `AddJob(String, String, Boolean)` senza generare un'eccezione, ma dal momento che `<PrinterName>` non è una stampante XPSDrv, verrà stampata solo l'operazione di Garbage Collection.

NOTE

Per semplicità, nell'esempio precedente viene utilizzata la presenza di un'estensione *.XPS come test di un file XPS. Tuttavia, i file XPS non devono avere questa estensione. [IsXPS.exe \(strumento di conformità isXPS\)](#) è un modo per verificare la validità di un file per XPS.

Vedere anche

- [PrintQueue](#)
- [AddJob](#)
- [ApartmentState](#)

- [STAThreadAttribute](#)
- [Documenti XPS](#)
- [Stampa di un documento XPS](#)
- [Threading gestito e non gestito](#)
- [isXPS.exe \(strumento di conformità isXPS\)](#)
- [Documenti in WPF](#)
- [Panoramica della stampa](#)

Procedura: Sorvegliare da remoto lo stato delle stampanti

23/10/2019 • 16 minutes to read • [Edit Online](#)

In qualsiasi momento in aziende di medie e grandi dimensioni potrebbero essere presenti più stampanti non funzionanti a causa di fogli bloccati o carta esaurita o un'altra situazione problematica. Il set completo di proprietà della stampante esposte nelle API di Microsoft .NET Framework fornisce un mezzo per eseguire un rapido sondaggio degli Stati delle stampanti.

Esempio

Di seguito sono indicati i passaggi principali per la creazione di questo tipo di utilità.

1. Ottenere un elenco di tutti i server di stampa.
2. Scorrere i server per eseguire una query sulle code di stampa.
3. In ogni passaggio del ciclo del server, scorrere tutte le code del server e leggere tutte le proprietà che potrebbero indicare che la coda attualmente non funziona.

Il codice seguente è una serie di frammenti di codice. Per semplicità, in questo esempio si presuppone che esista un elenco delimitato da CRLF di server di stampa. La variabile `fileOfPrintServers` è un `StreamReader` oggetto per il file. Poiché ogni nome di server si trova su una riga a se stante `ReadLine`, qualsiasi chiamata di ottiene il nome del server successivo `StreamReader` sposta il cursore all'inizio della riga successiva.

All'interno del ciclo esterno, il codice crea `PrintServer` un oggetto per il server di stampa più recente e specifica che l'applicazione deve disporre di diritti amministrativi per il server.

NOTE

Se sono presenti numerosi server, è possibile migliorare le prestazioni usando i `PrintServer(String, String[], PrintSystemDesiredAccess)` costruttori che inizializzano solo le proprietà necessarie.

Nell'esempio viene quindi `GetPrintQueues` usato per creare una raccolta di tutte le code del server e iniziare a scorrerle. Il ciclo interno include una struttura ramificata corrispondente ai due metodi di verifica dello stato della stampante:

- È possibile leggere i flag della `QueueStatus` proprietà che è di tipo `PrintQueueStatus`
- È possibile leggere tutte le proprietà rilevanti `IsOutOfPaper`, ad esempio, e. `IsPaperJammed`

Questo esempio illustra entrambi i metodi, quindi all'utente è stato chiesto in precedenza come metodo da usare e ha risposto con "y" se volesse usare i flag della `QueueStatus` proprietà. Per i dettagli dei due metodi, vedere di seguito.

Infine, i risultati vengono presentati all'utente.

```

// Survey queue status for every queue on every print server
System::String^ line;
System::String^ statusReport = "\n\nAny problem states are indicated below:\n\n";
while ((line = fileOfPrintServers->ReadLine()) != nullptr)
{
    System::Printing::PrintServer^ myPS = gcnew System::Printing::PrintServer(line,
PrintSystemDesiredAccess::AdministrateServer);
    System::Printing::PrintQueueCollection^ myPrintQueues = myPS->GetPrintQueues();
    statusReport = statusReport + "\n" + line;
    for each (System::Printing::PrintQueue^ pq in myPrintQueues)
    {
        pq->Refresh();
        statusReport = statusReport + "\n\t" + pq->Name + ":";

        if (useAttributesResponse == "y")
        {
            TroubleSpotter::SpotTroubleUsingQueueAttributes(statusReport, pq);
            // TroubleSpotter class is defined in the complete example.
        } else
        {
            TroubleSpotter::SpotTroubleUsingProperties(statusReport, pq);
        }
    }
}
fileOfPrintServers->Close();
Console::WriteLine(statusReport);
Console::WriteLine("\nPress Return to continue.");
Console::ReadLine();

```

```

// Survey queue status for every queue on every print server
String line;
String statusReport = "\n\nAny problem states are indicated below:\n\n";
while ((line = fileOfPrintServers.ReadLine()) != null)
{
    PrintServer myPS = new PrintServer(line, PrintSystemDesiredAccess.AdministrateServer);
    PrintQueueCollection myPrintQueues = myPS.GetPrintQueues();
    statusReport = statusReport + "\n" + line;
    foreach (PrintQueue pq in myPrintQueues)
    {
        pq.Refresh();
        statusReport = statusReport + "\n\t" + pq.Name + ":";

        if (useAttributesResponse == "y")
        {
            TroubleSpotter.SpotTroubleUsingQueueAttributes(ref statusReport, pq);
            // TroubleSpotter class is defined in the complete example.
        }
        else
        {
            TroubleSpotter.SpotTroubleUsingProperties(ref statusReport, pq);
        }
    }
    // end for each print queue
}

// end while list of print servers is not yet exhausted

fileOfPrintServers.Close();
Console.WriteLine(statusReport);
Console.WriteLine("\nPress Return to continue.");
Console.ReadLine();

```

```

' Survey queue status for every queue on every print server
Dim line As String
Dim statusReport As String = vbLf & vbLf & "Any problem states are indicated below:" & vbLf & vbLf
line = fileOfPrintServers.ReadLine()
Do While line IsNot Nothing
    Dim myPS As New PrintServer(line, PrintSystemDesiredAccess.AdministrateServer)
    Dim myPrintQueues As PrintQueueCollection = myPS.GetPrintQueues()
    statusReport = statusReport & vbLf & line
    For Each pq As PrintQueue In myPrintQueues
        pq.Refresh()
        statusReport = statusReport & vbLf & vbTab & pq.Name & ":" 
        If useAttributesResponse = "y" Then
            TroubleSpotter.SpotTroubleUsingQueueAttributes(statusReport, pq)
            ' TroubleSpotter class is defined in the complete example.
        Else
            TroubleSpotter.SpotTroubleUsingProperties(statusReport, pq)
        End If
    Next pq ' end for each print queue
    line = fileOfPrintServers.ReadLine()
Loop ' end while list of print servers is not yet exhausted

fileOfPrintServers.Close()
Console.WriteLine(statusReport)
Console.WriteLine(vbLf & "Press Return to continue.")
Console.ReadLine()

```

Per controllare lo stato della stampante usando i flag [QueueStatus](#) della proprietà, controllare ogni flag pertinente per verificare se è impostato. Il metodo standard per verificare se un bit è impostato in un set di flag di bit consiste nell'eseguire un'operazione di AND logico con il set di flag come uno degli operandi e il flag stesso come altro operando. Poiché il flag stesso ha un solo bit impostato, il risultato dell'AND logico è che, al massimo, è impostato quello stesso bit. Per verificare se lo è o meno, confrontare il risultato dell'AND logico con il flag stesso. Per ulteriori informazioni, vedere [PrintQueueStatus! operatore & \(C# riferimento\)](#)e. [FlagsAttribute](#)

Per ogni attributo il cui bit è impostato, il codice aggiunge un avviso al report finale che verrà presentato all'utente. (Il metodo **ReportAvailabilityAtThisTime** che viene chiamato alla fine del codice è illustrato di seguito.)

```

internal:
    // Check for possible trouble states of a printer using the flags of the QueueStatus property
    static void SpotTroubleUsingQueueAttributes (System::String^% statusReport, System::Printing::PrintQueue^
pq)
    {
        if ((pq->QueueStatus & PrintQueueStatus::PaperProblem) == PrintQueueStatus::PaperProblem)
        {
            statusReport = statusReport + "Has a paper problem. ";
        }
        if ((pq->QueueStatus & PrintQueueStatus::NoToner) == PrintQueueStatus::NoToner)
        {
            statusReport = statusReport + "Is out of toner. ";
        }
        if ((pq->QueueStatus & PrintQueueStatus::DoorOpen) == PrintQueueStatus::DoorOpen)
        {
            statusReport = statusReport + "Has an open door. ";
        }
        if ((pq->QueueStatus & PrintQueueStatus::Error) == PrintQueueStatus::Error)
        {
            statusReport = statusReport + "Is in an error state. ";
        }
        if ((pq->QueueStatus & PrintQueueStatus::NotAvailable) == PrintQueueStatus::NotAvailable)
        {
            statusReport = statusReport + "Is not available. ";
        }
        if ((pq->QueueStatus & PrintQueueStatus::Offline) == PrintQueueStatus::Offline)
        {
            statusReport = statusReport + "Is off line. ";
        }
        if ((pq->QueueStatus & PrintQueueStatus::OutOfMemory) == PrintQueueStatus::OutOfMemory)
        {
            statusReport = statusReport + "Is out of memory. ";
        }
        if ((pq->QueueStatus & PrintQueueStatus::PaperOut) == PrintQueueStatus::PaperOut)
        {
            statusReport = statusReport + "Is out of paper. ";
        }
        if ((pq->QueueStatus & PrintQueueStatus::OutputBinFull) == PrintQueueStatus::OutputBinFull)
        {
            statusReport = statusReport + "Has a full output bin. ";
        }
        if ((pq->QueueStatus & PrintQueueStatus::PaperJam) == PrintQueueStatus::PaperJam)
        {
            statusReport = statusReport + "Has a paper jam. ";
        }
        if ((pq->QueueStatus & PrintQueueStatus::Paused) == PrintQueueStatus::Paused)
        {
            statusReport = statusReport + "Is paused. ";
        }
        if ((pq->QueueStatus & PrintQueueStatus::TonerLow) == PrintQueueStatus::TonerLow)
        {
            statusReport = statusReport + "Is low on toner. ";
        }
        if ((pq->QueueStatus & PrintQueueStatus::UserIntervention) == PrintQueueStatus::UserIntervention)
        {
            statusReport = statusReport + "Needs user intervention. ";
        }

        // Check if queue is even available at this time of day
        // The method below is defined in the complete example.
        ReportAvailabilityAtThisTime(statusReport, pq);
    };

```

```

// Check for possible trouble states of a printer using the flags of the QueueStatus property
internal static void SpotTroubleUsingQueueAttributes(ref String statusReport, PrintQueue pq)
{
    if ((pq.QueueStatus & PrintQueueStatus.PaperProblem) == PrintQueueStatus.PaperProblem)
    {
        statusReport = statusReport + "Has a paper problem. ";
    }
    if ((pq.QueueStatus & PrintQueueStatus.NoToner) == PrintQueueStatus.NoToner)
    {
        statusReport = statusReport + "Is out of toner. ";
    }
    if ((pq.QueueStatus & PrintQueueStatus.DoorOpen) == PrintQueueStatus.DoorOpen)
    {
        statusReport = statusReport + "Has an open door. ";
    }
    if ((pq.QueueStatus & PrintQueueStatus.Error) == PrintQueueStatus.Error)
    {
        statusReport = statusReport + "Is in an error state. ";
    }
    if ((pq.QueueStatus & PrintQueueStatus.NotAvailable) == PrintQueueStatus.NotAvailable)
    {
        statusReport = statusReport + "Is not available. ";
    }
    if ((pq.QueueStatus & PrintQueueStatus.Offline) == PrintQueueStatus.Offline)
    {
        statusReport = statusReport + "Is off line. ";
    }
    if ((pq.QueueStatus & PrintQueueStatus.OutOfMemory) == PrintQueueStatus.OutOfMemory)
    {
        statusReport = statusReport + "Is out of memory. ";
    }
    if ((pq.QueueStatus & PrintQueueStatus.PaperOut) == PrintQueueStatus.PaperOut)
    {
        statusReport = statusReport + "Is out of paper. ";
    }
    if ((pq.QueueStatus & PrintQueueStatus.OutputBinFull) == PrintQueueStatus.OutputBinFull)
    {
        statusReport = statusReport + "Has a full output bin. ";
    }
    if ((pq.QueueStatus & PrintQueueStatus.PaperJam) == PrintQueueStatus.PaperJam)
    {
        statusReport = statusReport + "Has a paper jam. ";
    }
    if ((pq.QueueStatus & PrintQueueStatus.Paused) == PrintQueueStatus.Paused)
    {
        statusReport = statusReport + "Is paused. ";
    }
    if ((pq.QueueStatus & PrintQueueStatus.TonerLow) == PrintQueueStatus.TonerLow)
    {
        statusReport = statusReport + "Is low on toner. ";
    }
    if ((pq.QueueStatus & PrintQueueStatus.UserIntervention) == PrintQueueStatus.UserIntervention)
    {
        statusReport = statusReport + "Needs user intervention. ";
    }

    // Check if queue is even available at this time of day
    // The method below is defined in the complete example.
    ReportAvailabilityAtThisTime(ref statusReport, pq);
}

```

```

' Check for possible trouble states of a printer using the flags of the QueueStatus property
Friend Shared Sub SpotTroubleUsingQueueAttributes(ByRef statusReport As String, ByVal pq As PrintQueue)
    If (pq.QueueStatus And PrintQueueStatus.PaperProblem) = PrintQueueStatus.PaperProblem Then
        statusReport = statusReport & "Has a paper problem. "
    End If
    If (pq.QueueStatus And PrintQueueStatus.NoToner) = PrintQueueStatus.NoToner Then
        statusReport = statusReport & "Is out of toner. "
    End If
    If (pq.QueueStatus And PrintQueueStatus.DoorOpen) = PrintQueueStatus.DoorOpen Then
        statusReport = statusReport & "Has an open door. "
    End If
    If (pq.QueueStatus And PrintQueueStatus.Error) = PrintQueueStatus.Error Then
        statusReport = statusReport & "Is in an error state. "
    End If
    If (pq.QueueStatus And PrintQueueStatus.NotAvailable) = PrintQueueStatus.NotAvailable Then
        statusReport = statusReport & "Is not available. "
    End If
    If (pq.QueueStatus And PrintQueueStatus.Offline) = PrintQueueStatus.Offline Then
        statusReport = statusReport & "Is off line. "
    End If
    If (pq.QueueStatus And PrintQueueStatus.OutOfMemory) = PrintQueueStatus.OutOfMemory Then
        statusReport = statusReport & "Is out of memory. "
    End If
    If (pq.QueueStatus And PrintQueueStatus.PaperOut) = PrintQueueStatus.PaperOut Then
        statusReport = statusReport & "Is out of paper. "
    End If
    If (pq.QueueStatus And PrintQueueStatus.OutputBinFull) = PrintQueueStatus.OutputBinFull Then
        statusReport = statusReport & "Has a full output bin. "
    End If
    If (pq.QueueStatus And PrintQueueStatus.PaperJam) = PrintQueueStatus.PaperJam Then
        statusReport = statusReport & "Has a paper jam. "
    End If
    If (pq.QueueStatus And PrintQueueStatus.Paused) = PrintQueueStatus.Paused Then
        statusReport = statusReport & "Is paused. "
    End If
    If (pq.QueueStatus And PrintQueueStatus.TonerLow) = PrintQueueStatus.TonerLow Then
        statusReport = statusReport & "Is low on toner. "
    End If
    If (pq.QueueStatus And PrintQueueStatus.UserIntervention) = PrintQueueStatus.UserIntervention Then
        statusReport = statusReport & "Needs user intervention. "
    End If

    ' Check if queue is even available at this time of day
    ' The method below is defined in the complete example.
    ReportAvailabilityAtThisTime(statusReport, pq)
End Sub

```

Per controllare lo stato delle stampanti usando tutte le proprietà, basta leggere tutte le proprietà e aggiungere una nota alla relazione finale che sarà presentata all'utente se la proprietà è `true`. (Il metodo **ReportAvailabilityAtThisTime** che viene chiamato alla fine del codice è illustrato di seguito.)

```

internal:
    // Check for possible trouble states of a printer using its properties
    static void SpotTroubleUsingProperties (System::String^% statusReport, System::Printing::PrintQueue^ pq)
    {
        if (pq->HasPaperProblem)
        {
            statusReport = statusReport + "Has a paper problem. ";
        }
        if (!(pq->HasToner))
        {
            statusReport = statusReport + "Is out of toner. ";
        }
        if (pq->IsDoorOpened)
        {
            statusReport = statusReport + "Has an open door. ";
        }
        if (pq->IsInError)
        {
            statusReport = statusReport + "Is in an error state. ";
        }
        if (pq-> IsNotAvailable)
        {
            statusReport = statusReport + "Is not available. ";
        }
        if (pq->IsOffline)
        {
            statusReport = statusReport + "Is off line. ";
        }
        if (pq->IsOutOfMemory)
        {
            statusReport = statusReport + "Is out of memory. ";
        }
        if (pq->IsOutOfPaper)
        {
            statusReport = statusReport + "Is out of paper. ";
        }
        if (pq->IsOutputBinFull)
        {
            statusReport = statusReport + "Has a full output bin. ";
        }
        if (pq->IsPaperJammed)
        {
            statusReport = statusReport + "Has a paper jam. ";
        }
        if (pq->IsPaused)
        {
            statusReport = statusReport + "Is paused. ";
        }
        if (pq->IsTonerLow)
        {
            statusReport = statusReport + "Is low on toner. ";
        }
        if (pq->NeedUserIntervention)
        {
            statusReport = statusReport + "Needs user intervention. ";
        }

        // Check if queue is even available at this time of day
        // The following method is defined in the complete example.
        ReportAvailabilityAtThisTime(statusReport, pq);
    };

```

```

// Check for possible trouble states of a printer using its properties
internal static void SpotTroubleUsingProperties(ref String statusReport, PrintQueue pq)
{
    if (pq.HasPaperProblem)
    {
        statusReport = statusReport + "Has a paper problem. ";
    }
    if (!(pq.HasToner))
    {
        statusReport = statusReport + "Is out of toner. ";
    }
    if (pq.IsDoorOpened)
    {
        statusReport = statusReport + "Has an open door. ";
    }
    if (pq.IsInError)
    {
        statusReport = statusReport + "Is in an error state. ";
    }
    if (pq.IsNotNullAvailable)
    {
        statusReport = statusReport + "Is not available. ";
    }
    if (pq.IsNotNullOffline)
    {
        statusReport = statusReport + "Is off line. ";
    }
    if (pq.IsNotNullOfMemory)
    {
        statusReport = statusReport + "Is out of memory. ";
    }
    if (pq.IsNotNullOfPaper)
    {
        statusReport = statusReport + "Is out of paper. ";
    }
    if (pq.IsNotNullBinFull)
    {
        statusReport = statusReport + "Has a full output bin. ";
    }
    if (pq.IsPaperJammed)
    {
        statusReport = statusReport + "Has a paper jam. ";
    }
    if (pq.IsPaused)
    {
        statusReport = statusReport + "Is paused. ";
    }
    if (pq.IsTonerLow)
    {
        statusReport = statusReport + "Is low on toner. ";
    }
    if (pq.NeedUserIntervention)
    {
        statusReport = statusReport + "Needs user intervention. ";
    }

    // Check if queue is even available at this time of day
    // The following method is defined in the complete example.
    ReportAvailabilityAtThisTime(ref statusReport, pq);
}

//end SpotTroubleUsingProperties

```

```

' Check for possible trouble states of a printer using its properties
Friend Shared Sub SpotTroubleUsingProperties(ByRef statusReport As String, ByVal pq As PrintQueue)
    If pq.HasPaperProblem Then
        statusReport = statusReport & "Has a paper problem. "
    End If
    If Not(pq.HasToner) Then
        statusReport = statusReport & "Is out of toner. "
    End If
    If pq.IsDoorOpened Then
        statusReport = statusReport & "Has an open door. "
    End If
    If pq.IsInError Then
        statusReport = statusReport & "Is in an error state. "
    End If
    If pq.IsNotNullAvailable Then
        statusReport = statusReport & "Is not available. "
    End If
    If pq.IsOffline Then
        statusReport = statusReport & "Is off line. "
    End If
    If pq.IsOutOfMemory Then
        statusReport = statusReport & "Is out of memory. "
    End If
    If pq.IsOutOfPaper Then
        statusReport = statusReport & "Is out of paper. "
    End If
    If pq.IsOutputBinFull Then
        statusReport = statusReport & "Has a full output bin. "
    End If
    If pq.IsPaperJammed Then
        statusReport = statusReport & "Has a paper jam. "
    End If
    If pq.IsPaused Then
        statusReport = statusReport & "Is paused. "
    End If
    If pq.IsTonerLow Then
        statusReport = statusReport & "Is low on toner. "
    End If
    If pq.NeedUserIntervention Then
        statusReport = statusReport & "Needs user intervention. "
    End If

    ' Check if queue is even available at this time of day
    ' The following method is defined in the complete example.
    ReportAvailabilityAtThisTime(statusReport, pq)

End Sub

```

Il metodo **ReportAvailabilityAtThisTime** è stato creato nel caso in cui sia necessario determinare se la coda è disponibile in quel momento preciso del giorno.

Il metodo non eseguirà alcuna operazione [StartTimeOfDay](#) se [UntilTimeOfDay](#) le proprietà e sono uguali; in tal caso, la stampante è sempre disponibile. Se sono diversi, il metodo ottiene l'ora corrente che deve quindi essere convertita in minuti totali dopo la mezzanotte perché le [StartTimeOfDay](#) proprietà [UntilTimeOfDay](#) e sono [Int32](#) che rappresentano i minuti, dopo la mezzanotte, [DateTime](#) non oggetti. Infine, il metodo controlla se l'ora corrente è compresa tra l'inizio e il lasso di tempo "fino a".

```

private:
    static void ReportAvailabilityAtThisTime (System::String^% statusReport, System::Printing::PrintQueue^ pq)
    {
        if (pq->StartTimeOfDay != pq->UntilTimeOfDay)
        {
            System::DateTime utcNow = DateTime::UtcNow;
            System::Int32 utcNowAsMinutesAfterMidnight = (utcNow.TimeOfDay.Hours * 60) +
            utcNow.TimeOfDay.Minutes;

            // If now is not within the range of available times . . .
            if (!((pq->StartTimeOfDay < utcNowAsMinutesAfterMidnight) && (utcNowAsMinutesAfterMidnight < pq-
            >UntilTimeOfDay)))
            {
                statusReport = statusReport + " Is not available at this time of day. ";
            }
        }
    };

```

```

private static void ReportAvailabilityAtThisTime(ref String statusReport, PrintQueue pq)
{
    if (pq.StartTimeOfDay != pq.UntilTimeOfDay) // If the printer is not available 24 hours a day
    {
        DateTime utcNow = DateTime.UtcNow;
        Int32 utcNowAsMinutesAfterMidnight = (utcNow.TimeOfDay.Hours * 60) + utcNow.TimeOfDay.Minutes;

        // If now is not within the range of available times . . .
        if (!((pq.StartTimeOfDay < utcNowAsMinutesAfterMidnight)
              &&
              (utcNowAsMinutesAfterMidnight < pq.UntilTimeOfDay)))
        {
            statusReport = statusReport + " Is not available at this time of day. ";
        }
    }
}

```

```

Private Shared Sub ReportAvailabilityAtThisTime(ByRef statusReport As String, ByVal pq As PrintQueue)
    If pq.StartTimeOfDay <> pq.UntilTimeOfDay Then ' If the printer is not available 24 hours a day
        Dim utcNow As Date = Date.UtcNow
        Dim utcNowAsMinutesAfterMidnight As Int32 = (utcNow.TimeOfDay.Hours * 60) + utcNow.TimeOfDay.Minutes

        ' If now is not within the range of available times . . .
        If Not((pq.StartTimeOfDay < utcNowAsMinutesAfterMidnight) AndAlso (utcNowAsMinutesAfterMidnight <
        pq.UntilTimeOfDay)) Then
            statusReport = statusReport & " Is not available at this time of day. "
        End If
    End If
End Sub

```

Vedere anche

- [StartTimeOfDay](#)
- [UntilTimeOfDay](#)
- [DateTime](#)
- [PrintQueueStatus](#)
- [FlagsAttribute](#)
- [GetPrintQueues](#)
- [PrintServer](#)
- [LocalPrintServer](#)

- [EnumeratedPrintQueueTypes](#)
- [PrintQueue](#)
- [Operatore & \(C# riferimento\)](#)
- [Documenti in WPF](#)
- [Panoramica della stampa](#)

Procedura: convalidare e unire PrintTicket

10/02/2020 • 10 minutes to read • [Edit Online](#)

Lo schema di [stampa](#) di Microsoft Windows include gli elementi [PrintCapabilities](#) e [PrintTicket](#) flessibili ed estendibili. Il primo rileva le funzionalità di un dispositivo di stampa e il secondo specifica il modo in cui il dispositivo deve usare tali funzionalità rispetto a una particolare sequenza di documenti, documento singolo o singola pagina.

Di seguito è riportata una sequenza di attività tipica per un'applicazione che supporta la stampa.

1. Determinare le funzionalità di una stampante.
2. Configurare un [PrintTicket](#) per usare tali funzionalità.
3. Convalidare il [PrintTicket](#).

Questo articolo illustra come eseguire questa operazione.

Esempio

Nell'esempio semplice riportato di seguito, si è interessati solo a se una stampante può supportare il duplexing, ovvero la stampa su due lati. I passaggi principali sono i seguenti.

1. Ottenere un oggetto [PrintCapabilities](#) con il metodo [GetPrintCapabilities](#).
2. Verificare la presenza delle funzionalità desiderate. Nell'esempio seguente viene testato il [DuplexingCapability](#) proprietà dell'oggetto [PrintCapabilities](#) per la presenza della funzionalità di stampa su entrambi i lati di un foglio di carta con la "trasformazione pagina" lungo il lato lungo del foglio. Poiché [DuplexingCapability](#) è una raccolta, viene usato il metodo [Contains](#) di [ReadOnlyCollection<T>](#).

NOTE

Questo passaggio non è strettamente necessario. Il metodo [MergeAndValidatePrintTicket](#) usato di seguito consente di controllare ogni richiesta nell'[PrintTicket](#) con le funzionalità della stampante. Se la funzionalità richiesta non è supportata dalla stampante, il driver della stampante sostituirà una richiesta alternativa nel [PrintTicket](#) restituito dal metodo.

3. Se la stampante supporta la duplexing, il codice di esempio crea un [PrintTicket](#) che richiede il duplexing. Tuttavia, l'applicazione non specifica ogni possibile impostazione della stampante disponibile nell'elemento [PrintTicket](#). Questo sarebbe uno spreco di tempo per programmatore e programma. Al contrario, il codice impone solo la richiesta di duplexing, quindi unisce questo [PrintTicket](#) a una [PrintTicket](#) esistente, completamente configurata e convalidata, in questo caso, il [PrintTicket](#) predefinito dell'utente.
4. Di conseguenza, l'esempio chiama il metodo [MergeAndValidatePrintTicket](#) per unire il nuovo [PrintTicket](#) minimo con il [PrintTicket](#) predefinito dell'utente. Viene restituito un [ValidationResult](#) che include il nuovo [PrintTicket](#) come una delle relative proprietà.
5. Nell'esempio viene quindi verificato che il nuovo [PrintTicket](#) richiede il duplexing. In caso contrario, l'esempio lo rende il nuovo ticket di stampa predefinito per l'utente. Se il passaggio 2 precedente era stato interrotto e la stampante non supportava il duplex lungo il lato lungo, il test avrebbe avuto come risultato [false](#). (Vedere la nota precedente).
6. L'ultimo passaggio significativo consiste nel eseguire il commit della modifica alla proprietà [UserPrintTicket](#)

della [PrintQueue](#) con il metodo [Commit](#).

```

/// <summary>
/// Changes the user-default PrintTicket setting of the specified print queue.
/// </summary>
/// <param name="queue">the printer whose user-default PrintTicket setting needs to be changed</param>
static private void ChangePrintTicketSetting(PrintQueue queue)
{
    //
    // Obtain the printer's PrintCapabilities so we can determine whether or not
    // duplexing printing is supported by the printer.
    //
    PrintCapabilities printcap = queue.GetPrintCapabilities();

    //
    // The printer's duplexing capability is returned as a read-only collection of duplexing options
    // that can be supported by the printer. If the collection returned contains the duplexing
    // option we want to set, it means the duplexing option we want to set is supported by the printer,
    // so we can make the user-default PrintTicket setting change.
    //
    if (printcap.DuplexingCapability.Contains(Duplexing.TwoSidedLongEdge))
    {
        //
        // To change the user-default PrintTicket, we can first create a delta PrintTicket with
        // the new duplexing setting.
        //
        PrintTicket deltaTicket = new PrintTicket();
        deltaTicket.Duplexing = Duplexing.TwoSidedLongEdge;

        //
        // Then merge the delta PrintTicket onto the printer's current user-default PrintTicket,
        // and validate the merged PrintTicket to get the new PrintTicket we want to set as the
        // printer's new user-default PrintTicket.
        //
        ValidationResult result = queue.MergeAndValidatePrintTicket(queue.UserPrintTicket, deltaTicket);

        //
        // The duplexing option we want to set could be constrained by other PrintTicket settings
        // or device settings. We can check the validated merged PrintTicket to see whether the
        // the validation process has kept the duplexing option we want to set unchanged.
        //
        if (result.ValidatedPrintTicket.Duplexing == Duplexing.TwoSidedLongEdge)
        {
            //
            // Set the printer's user-default PrintTicket and commit the set operation.
            //
            queue.UserPrintTicket = result.ValidatedPrintTicket;
            queue.Commit();
            Console.WriteLine("PrintTicket new duplexing setting is set on '{0}'.", queue.FullName);
        }
        else
        {
            //
            // The duplexing option we want to set has been changed by the validation process
            // when it was resolving setting constraints.
            //
            Console.WriteLine("PrintTicket new duplexing setting is constrained on '{0}'.", queue.FullName);
        }
    }
    else
    {
        //
        // If the printer doesn't support the duplexing option we want to set, skip it.
        //
        Console.WriteLine("PrintTicket new duplexing setting is not supported on '{0}'.", queue.FullName);
    }
}

```

```

''' <summary>
''' Changes the user-default PrintTicket setting of the specified print queue.
''' </summary>
''' <param name="queue">the printer whose user-default PrintTicket setting needs to be changed</param>
Private Shared Sub ChangePrintTicketSetting(ByVal queue As PrintQueue)

    ' Obtain the printer's PrintCapabilities so we can determine whether or not
    ' duplexing printing is supported by the printer.
    '

    Dim printcap As PrintCapabilities = queue.GetPrintCapabilities()

    '

    ' The printer's duplexing capability is returned as a read-only collection of duplexing options
    ' that can be supported by the printer. If the collection returned contains the duplexing
    ' option we want to set, it means the duplexing option we want to set is supported by the printer,
    ' so we can make the user-default PrintTicket setting change.
    '

    If printcap.DuplexingCapability.Contains(Duplexing.TwoSidedLongEdge) Then
        '

        ' To change the user-default PrintTicket, we can first create a delta PrintTicket with
        ' the new duplexing setting.
        '

        Dim deltaTicket As New PrintTicket()
        deltaTicket.Duplexing = Duplexing.TwoSidedLongEdge

        '

        ' Then merge the delta PrintTicket onto the printer's current user-default PrintTicket,
        ' and validate the merged PrintTicket to get the new PrintTicket we want to set as the
        ' printer's new user-default PrintTicket.
        '

        Dim result As ValidationResult = queue.MergeAndValidatePrintTicket(queue.UserPrintTicket, deltaTicket)

        '

        ' The duplexing option we want to set could be constrained by other PrintTicket settings
        ' or device settings. We can check the validated merged PrintTicket to see whether the
        ' validation process has kept the duplexing option we want to set unchanged.
        '

        If result.ValidatedPrintTicket.Duplexing = Duplexing.TwoSidedLongEdge Then
            '

            ' Set the printer's user-default PrintTicket and commit the set operation.
            '

            queue.UserPrintTicket = result.ValidatedPrintTicket
            queue.Commit()
            Console.WriteLine("PrintTicket new duplexing setting is set on '{0}'.", queue.FullName)
        Else
            '

            ' The duplexing option we want to set has been changed by the validation process
            ' when it was resolving setting constraints.
            '

            Console.WriteLine("PrintTicket new duplexing setting is constrained on '{0}'.", queue.FullName)
        End If
    Else
        '

        ' If the printer doesn't support the duplexing option we want to set, skip it.
        '

        Console.WriteLine("PrintTicket new duplexing setting is not supported on '{0}'.", queue.FullName)
    End If
End Sub

```

Per poter testare rapidamente questo esempio, il resto viene presentato di seguito. Creare un progetto e uno spazio dei nomi e quindi incollare entrambi i frammenti di codice di questo articolo nel blocco dello spazio dei nomi.

```

/// <summary>
/// Displays the correct command line syntax to run this sample program.
/// ...

```

```

/// </summary>
static private void DisplayUsage()
{
    Console.WriteLine();
    Console.WriteLine("Usage #1: printticket.exe -l \"<printer_name>\\"");
    Console.WriteLine("      Run program on the specified local printer");
    Console.WriteLine();
    Console.WriteLine("      Quotation marks may be omitted if there are no spaces in printer_name.");
    Console.WriteLine();
    Console.WriteLine("Usage #2: printticket.exe -r \"\\\\\\<server_name>\\<printer_name>\\"");
    Console.WriteLine("      Run program on the specified network printer");
    Console.WriteLine();
    Console.WriteLine("      Quotation marks may be omitted if there are no spaces in server_name or
printer_name.");
    Console.WriteLine();
    Console.WriteLine("Usage #3: printticket.exe -a");
    Console.WriteLine("      Run program on all installed printers");
    Console.WriteLine();
}

[STAThread]
static public void Main(string[] args)
{
    try
    {
        if ((args.Length == 1) && (args[0] == "-a"))
        {
            //
            // Change PrintTicket setting for all local and network printer connections.
            //
            LocalPrintServer server = new LocalPrintServer();

            EnumeratedPrintQueueTypes[] queue_types = {EnumeratedPrintQueueTypes.Local,
                                                       EnumeratedPrintQueueTypes.Connections};

            //
            // Enumerate through all the printers.
            //
            foreach (PrintQueue queue in server.GetPrintQueues(queue_types))
            {
                //
                // Change the PrintTicket setting queue by queue.
                //
                ChangePrintTicketSetting(queue);
            }
        }//end if -a

        else if ((args.Length == 2) && (args[0] == "-l"))
        {
            //
            // Change PrintTicket setting only for the specified local printer.
            //
            LocalPrintServer server = new LocalPrintServer();
            PrintQueue queue = new PrintQueue(server, args[1]);
            ChangePrintTicketSetting(queue);
        }//end if -l

        else if ((args.Length == 2) && (args[0] == "-r"))
        {
            //
            // Change PrintTicket setting only for the specified remote printer.
            //
            String serverName = args[1].Remove(args[1].LastIndexOf(@"\""));
            String printerName = args[1].Remove(0, args[1].LastIndexOf(@"\") + 1);
            PrintServer ps = new PrintServer(serverName);
            PrintQueue queue = new PrintQueue(ps, printerName);
            ChangePrintTicketSetting(queue);
        }//end if -r
    }
}

```

```

        else
        {
            //
            // Unrecognized command line.
            // Show user the correct command line syntax to run this sample program.
            //
            DisplayUsage();
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
        Console.WriteLine(e.StackTrace);

        //
        // Show inner exception information if it's provided.
        //
        if (e.InnerException != null)
        {
            Console.WriteLine("--- Inner Exception ---");
            Console.WriteLine(e.InnerException.Message);
            Console.WriteLine(e.InnerException.StackTrace);
        }
    }
    finally
    {
        Console.WriteLine("Press Return to continue...");
        Console.ReadLine();
    }
}//end Main

```

```

''' <summary>
''' Displays the correct command line syntax to run this sample program.
''' </summary>
Private Shared Sub DisplayUsage()
    Console.WriteLine()
    Console.WriteLine("Usage #1: printticket.exe -l ""<printer_name>""")
    Console.WriteLine("      Run program on the specified local printer")
    Console.WriteLine()
    Console.WriteLine("      Quotation marks may be omitted if there are no spaces in printer_name.")
    Console.WriteLine()
    Console.WriteLine("Usage #2: printticket.exe -r ""\\<server_name>\<printer_name>""")
    Console.WriteLine("      Run program on the specified network printer")
    Console.WriteLine()
    Console.WriteLine("      Quotation marks may be omitted if there are no spaces in server_name or
printer_name")
    Console.WriteLine()
    Console.WriteLine("Usage #3: printticket.exe -a")
    Console.WriteLine("      Run program on all installed printers")
    Console.WriteLine()
End Sub

<STAThread>
Public Shared Sub Main(ByVal args() As String)
    Try
        If (args.Length = 1) AndAlso (args(0) = "-a") Then
            '
            ' Change PrintTicket setting for all local and network printer connections.
            '
            Dim server As New LocalPrintServer()

            Dim queue_types() As EnumeratedPrintQueueTypes = {EnumeratedPrintQueueTypes.Local,
EnumeratedPrintQueueTypes.Connections}

            '
            ' Enumerate through all the printers.
            '

```

```

For Each queue As PrintQueue In server.GetPrintQueues(queue_types)
    '
    ' Change the PrintTicket setting queue by queue.
    '

    ChangePrintTicketSetting(queue)
Next queue 'end if -a

ElseIf (args.Length = 2) AndAlso (args(0) = "-l") Then
    '
    ' Change PrintTicket setting only for the specified local printer.
    '

    Dim server As New LocalPrintServer()
    Dim queue As New PrintQueue(server, args(1))
    ChangePrintTicketSetting(queue) 'end if -l

ElseIf (args.Length = 2) AndAlso (args(0) = "-r") Then
    '
    ' Change PrintTicket setting only for the specified remote printer.
    '

    Dim serverName As String = args(1).Remove(args(1).LastIndexOf("\"))
    Dim printerName As String = args(1).Remove(0, args(1).LastIndexOf("\") + 1)
    Dim ps As New PrintServer(serverName)
    Dim queue As New PrintQueue(ps, printerName)
    ChangePrintTicketSetting(queue) 'end if -r

Else
    '
    ' Unrecognized command line.
    ' Show user the correct command line syntax to run this sample program.
    '

    DisplayUsage()
End If

Catch e As Exception
    Console.WriteLine(e.Message)
    Console.WriteLine(e.StackTrace)

    '
    ' Show inner exception information if it's provided.
    '

    If e.InnerException IsNot Nothing Then
        Console.WriteLine("--- Inner Exception ---")
        Console.WriteLine(e.InnerException.Message)
        Console.WriteLine(e.InnerException.StackTrace)
    End If
Finally
    Console.WriteLine("Press Return to continue...")
    Console.ReadLine()
End Try
End Sub

```

Vedere anche

- [PrintCapabilities](#)
- [PrintTicket](#)
- [GetPrintQueues](#)
- [PrintServer](#)
- [EnumeratedPrintQueueTypes](#)
- [PrintQueue](#)
- [GetPrintCapabilities](#)
- [Documenti in WPF](#)
- [Panoramica della stampa](#)

- Stampa schema

Globalizzazione e localizzazione

08/01/2020 • 2 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) offre un supporto completo per lo sviluppo di applicazioni internazionali.

In questa sezione

[Panoramica della globalizzazione e localizzazione WPF](#)

[Globalizzazione per WPF](#)

[Cenni preliminari sull'utilizzo del layout automatico](#)

[Attributi e commenti di localizzazione](#)

[Panoramica sulle funzionalità bidirezionali di WPF](#)

[Procedure relative alle proprietà](#)

Riferimenti

[System.Globalization](#)

[FlowDirection](#)

[NeutralResourcesLanguageAttribute](#)

[Gestione di xml:lang in XAML](#)

Sezioni correlate

Panoramica della globalizzazione e localizzazione WPF

19/02/2020 • 31 minutes to read • [Edit Online](#)

Quando si limita la disponibilità del prodotto a una sola lingua, si limitano le potenziali basi clienti a una frazione della popolazione 7,5 miliardi del mondo. Se si vuole che le applicazioni raggiungano un pubblico globale, la localizzazione economica del prodotto è uno dei modi migliori e più economici per raggiungere un ampio pubblico di clienti.

Questa panoramica introduce la globalizzazione e la localizzazione in Windows Presentation Foundation (WPF). La globalizzazione si basa sulla progettazione e sviluppo di applicazioni eseguibili in più posizioni. Ad esempio, la globalizzazione supporta interfacce utente e dati internazionali localizzati per utenti in varie impostazioni cultura. WPF offre funzionalità di progettazione globalizzate, tra cui layout automatico, assembly satellite e attributi localizzati e commenti.

La localizzazione consiste nella conversione delle risorse dell'applicazione in versioni localizzate per specifiche impostazioni cultura supportate dall'applicazione. Quando si esegue la localizzazione in WPF, si usano le API nello spazio dei nomi [System.Windows.Markup.Localizer](#). Queste API potenziano lo strumento da riga di comando di [esempio per lo strumento LocBaml](#). Per informazioni su come compilare e usare LocBaml, vedere [localizzare un'applicazione](#).

Procedure consigliate per la globalizzazione e la localizzazione in WPF

Per sfruttare al meglio le funzionalità di globalizzazione e localizzazione integrate in WPF, è possibile seguire i suggerimenti relativi alla progettazione dell'interfaccia utente e alla localizzazione forniti da questa sezione.

Procedure consigliate per la progettazione dell'interfaccia utente WPF

Quando si progetta un Interfaccia utente basato su WPF, è consigliabile implementare le procedure consigliate seguenti:

- Scrivere il Interfaccia utente in XAML; evitare di creare Interfaccia utente nel codice. Quando si crea il Interfaccia utente usando XAML, questo viene esposto tramite le API di localizzazione predefinite.
- Evitare di usare posizioni assolute e dimensioni fisse per il layout del contenuto; usare invece il ridimensionamento relativo o automatico.
 - Usare [SizeToContent](#) e Mantieni le larghezze e le altezze impostate su `Auto`.
 - Evitare di usare [Canvas](#) per il layout di Interfaccia utente.
 - Usare [Grid](#) e la relativa funzionalità di condivisione delle dimensioni.
- Lasciare uno spazio aggiuntivo ai margini perché il testo localizzato spesso richiede più spazio. Lo spazio aggiuntivo servirà per eventuali caratteri sporgenti.
- Abilitare [TextWrapping](#) su [TextBlock](#) per evitare il ritaglio.
- Impostare l'attributo `xml:lang`. Questo attributo descrive le impostazioni cultura di un elemento specifico e dei relativi elementi figlio. Il valore di questa proprietà modifica il comportamento di diverse funzionalità in WPF. Ad esempio, cambia il comportamento della sillabazione, del controllo ortografico, della sostituzione dei numeri, della definizione di lingue con alfabeto non latino e del fallback dei tipi di carattere. Per ulteriori informazioni sull'impostazione della [gestione di XML: lang in XAML](#), vedere [globalizzazione](#)

per WPF .

- Creare un tipo di carattere composito personalizzato per ottenere un controllo migliore sui tipi di carattere usati per lingue diverse. Per impostazione predefinita, WPF usa il tipo di carattere carattere GlobalUserInterface. composite nella directory Windows\Fonts.
- Quando si creano applicazioni di navigazione che possono essere localizzate in impostazioni cultura che presentano testo in un formato da destra a sinistra, impostare in modo esplicito il [FlowDirection](#) di ogni pagina per assicurarsi che la pagina non erediti [FlowDirection](#) dal [NavigationWindow](#).
- Quando si creano applicazioni di navigazione autonome ospitate all'esterno di un browser, impostare l'[StartupUri](#) per l'applicazione iniziale su un [NavigationWindow](#) invece che su una pagina, ad esempio `<Application StartupUri="NavigationWindow.xaml">`. Questa progettazione consente di modificare il [FlowDirection](#) della finestra e della barra di spostamento. Per ulteriori informazioni e un esempio, vedere [esempio di globalizzazione della Home page](#).

Procedure consigliate per la localizzazione di WPF

Quando si localizzano applicazioni basate su WPF, è consigliabile implementare le procedure consigliate seguenti:

- Usare i commenti di localizzazione per fornire un contesto aggiuntivo per i localizzatori.
- Utilizzare gli attributi di localizzazione per controllare la localizzazione anziché omettere selettivamente [Uid](#) proprietà sugli elementi. Per ulteriori informazioni, vedere [attributi e commenti di localizzazione](#) .
- Usare `msbuild -t:updateuid` e `-t:checkuid` per aggiungere e controllare [Uid](#) proprietà nel XAML. Usare [Uid](#) proprietà per tenere traccia delle modifiche tra lo sviluppo e la localizzazione. [Uid](#) proprietà consentono di localizzare nuove modifiche di sviluppo. Se si aggiungono manualmente [Uid](#) proprietà a una Interfaccia utente, l'attività è in genere noiosa e meno accurata.
 - Non modificare o modificare [Uid](#) proprietà dopo l'avvio della localizzazione.
 - Non usare proprietà di [Uid](#) duplicate (ricordare questo suggerimento quando si usa il comando copy-and-paste).
 - Impostare il percorso di `UltimateResourceFallback` in AssemblyInfo.* specificare la lingua appropriata per il fallback, ad esempio
`[assembly: NeutralResourcesLanguage("en-US", UltimateResourceFallbackLocation.Satellite)]`.

Se si decide di includere la lingua di origine nell'assembly principale omettendo il tag `<UICulture>` nel file di progetto, impostare il percorso della `ultimateResourceFallback` come assembly principale anziché il satellite, ad esempio

```
[assembly: NeutralResourcesLanguage("en-US", UltimateResourceFallbackLocation.MainAssembly)]
```

Localizzare un'applicazione WPF

Quando si localizza un'applicazione WPF, sono disponibili diverse opzioni. Ad esempio, è possibile associare le risorse localizzabili nell'applicazione a un file XML, archiviare il testo localizzabile nelle tabelle RESX o fare in modo che il localizzatore usi Extensible Application Markup Language (XAML) file. Questa sezione descrive un flusso di lavoro di localizzazione che usa il modulo BAML di XAML, che offre diversi vantaggi:

- È possibile localizzare dopo la compilazione.
- È possibile eseguire l'aggiornamento a una versione più recente del modulo BAML di XAML con le localizzazioni da una versione precedente del modulo BAML di XAML in modo che sia possibile localizzare nello stesso momento in cui si sviluppa.
- È possibile convalidare gli elementi di origine e la semantica in fase di compilazione perché il modulo BAML di XAML è la forma compilata di XAML.

Processo di compilazione per la localizzazione

Quando si sviluppa un'applicazione WPF, il processo di compilazione per la localizzazione è il seguente:

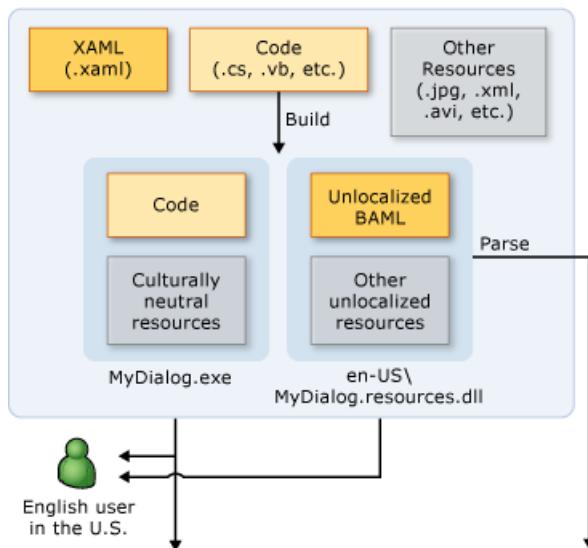
- Lo sviluppatore crea e globalizza l'applicazione WPF. Nel file di progetto lo sviluppatore imposta `<UICulture>en-US</UICulture>` in modo che, quando l'applicazione viene compilata, venga generato un assembly principale indipendente dalla lingua. Questo assembly dispone di un file satellite con estensione resources.dll che contiene tutte le risorse localizzabili. Facoltativamente, è possibile mantenere la lingua di origine nell'assembly principale perché le API di localizzazione supportano l'estrazione dall'assembly principale.
- Quando il file viene compilato nella compilazione, il XAML viene convertito nel formato BAML di XAML. Il `MyDialog.exe` culturalmente neutro e i file di `MyDialog.resources.dll` dipendenti dalla cultura (Inglese) vengono rilasciati ai clienti di lingua inglese.

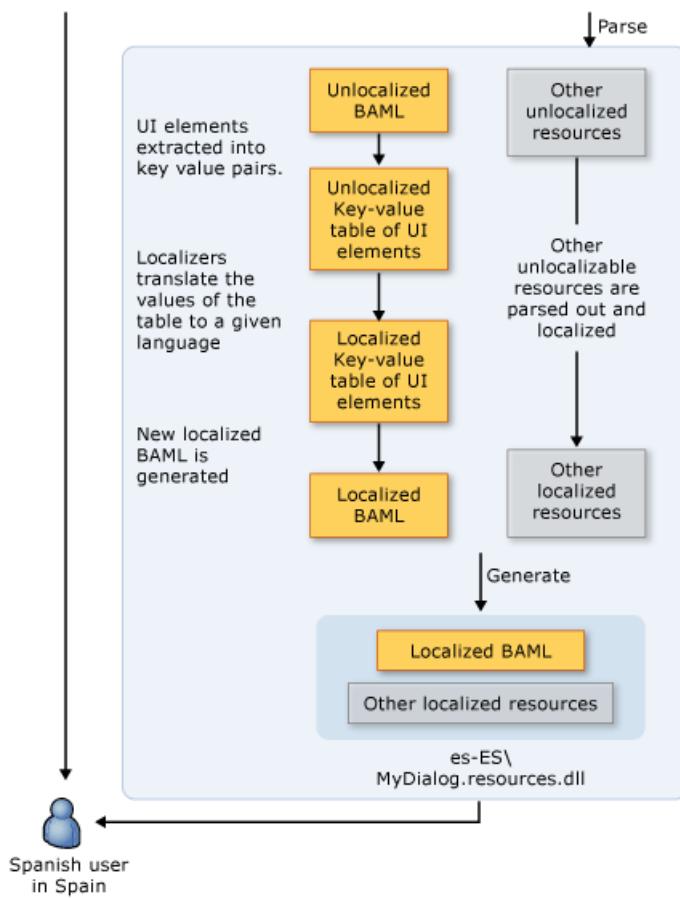
Flusso di lavoro della localizzazione

Il processo di localizzazione inizia dopo la compilazione del file di `MyDialog.resources.dll` non localizzato. Gli elementi e le proprietà Interfaccia utente nel XAML originale vengono estratti dal formato BAML di XAML in coppie chiave-valore usando le API in `System.Windows.Markup.Localizer`. I localizzatori usano le coppie chiave-valore per localizzare l'applicazione. È possibile generare un nuovo file con estensione resource.dll a partire dai nuovi valori dopo che la localizzazione è stata completata.

Le chiavi delle coppie chiave-valore sono `x:Uid` valori posizionati dallo sviluppatore nella XAML originale. Questi valori `x:Uid` consentono all'API di rilevare e unire le modifiche che si verificano tra lo sviluppatore e il localizzatore durante la localizzazione. Se, ad esempio, lo sviluppatore modifica il Interfaccia utente dopo che il localizzatore inizia a localizzare, è possibile unire la modifica allo sviluppo con il lavoro di localizzazione già completato in modo da perdere il lavoro di traduzione minima.

La figura seguente mostra un tipico flusso di lavoro di localizzazione basato sul modulo BAML di XAML. Questo diagramma presuppone che lo sviluppatore scriva l'applicazione in lingua inglese. Lo sviluppatore crea e globalizza l'applicazione WPF. Nel file di progetto lo sviluppatore imposta `<UICulture>en-US</UICulture>` in modo che, durante la compilazione, venga generato un assembly principale indipendente dalla lingua con un file satellite resources.dll contenente tutte le risorse localizzabili. In alternativa, è possibile mantenere la lingua di origine nell'assembly principale poiché le API di localizzazione WPF supportano l'estrazione dall'assembly principale. Dopo il processo di compilazione, il codice XAML viene compilato in BAML. Il file `MyDialog.exe.resources.dll` indipendente dalla lingua viene distribuito ai clienti di lingua inglese.





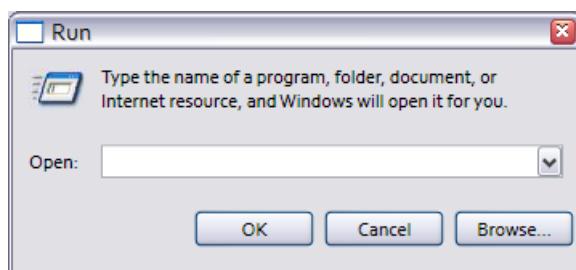
Esempi di localizzazione WPF

Questa sezione contiene esempi di applicazioni localizzate che consentono di comprendere come compilare e localizzare WPF applicazioni.

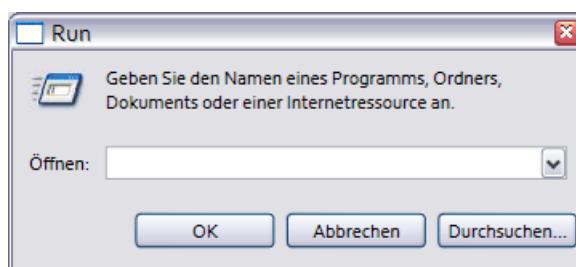
Esempio di finestra di dialogo Esegui

La grafica seguente mostra l'output dell'esempio di finestra di dialogo **Esegui**.

Inglese:



Tedesco:



Progettazione di una finestra di dialogo Esegui globale

Questo esempio genera una finestra di dialogo **Esegui** utilizzando WPF e XAML. Questa finestra di dialogo è equivalente alla finestra di dialogo **Esegui** disponibile nel menu Start di Microsoft Windows.

Alcune delle caratteristiche principali delle finestre di dialogo globali sono:

Layout automatico

In Window1.xaml:

```
<Window SizeToContent="WidthAndHeight">
```

La proprietà Window precedente ridimensiona automaticamente la finestra in base alle dimensioni del contenuto. Questa proprietà impedisce alla finestra di troncare il contenuto che aumenta di dimensioni dopo la localizzazione. Rimuove anche lo spazio superfluo quando le dimensioni del contenuto si riducono dopo la localizzazione.

```
<Grid x:Uid="Grid_1">
```

per il corretto funzionamento delle API di localizzazione WPF sono necessarie **Uid** proprietà.

Vengono usati dalle API di localizzazione WPF per tenere traccia delle modifiche tra lo sviluppo e la localizzazione del interfaccia utente. **Uid** proprietà consentono di unire una versione più recente del Interfaccia utente con una localizzazione precedente del Interfaccia utente. Per aggiungere una proprietà di **Uid**, è possibile eseguire `msbuild -t:updateuid RunDialog.csproj` in una shell dei comandi. Questo è il metodo consigliato per aggiungere **Uid** proprietà perché l'aggiunta manuale è in genere molto dispendiosa in termini di tempo e meno accurata. È possibile verificare che le proprietà **Uid** siano impostate correttamente eseguendo `msbuild -t:checkuid RunDialog.csproj`.

Il Interfaccia utente è strutturato usando il controllo **Grid**, che è un controllo utile per sfruttare il layout automatico in WPF. Si noti che la finestra di dialogo è suddivisa in tre righe e cinque colonne. Una delle definizioni di riga e colonna ha dimensioni fisse; di conseguenza, gli elementi di Interfaccia utente posizionati in ogni cella possono adattarsi ad aumenti e riduzioni delle dimensioni durante la localizzazione.

```
<Grid.ColumnDefinitions>
  <ColumnDefinition x:Uid="ColumnDefinition_1" />
  <ColumnDefinition x:Uid="ColumnDefinition_2" />
```

Le prime due colonne in cui vengono posizionati i **ComboBox** **Open:** label e usano il 10% della larghezza totale Interfaccia utente.

```
<ColumnDefinition x:Uid="ColumnDefinition_3" SharedSizeGroup="Buttons" />
<ColumnDefinition x:Uid="ColumnDefinition_4" SharedSizeGroup="Buttons" />
<ColumnDefinition x:Uid="ColumnDefinition_5" SharedSizeGroup="Buttons" />
</Grid.ColumnDefinitions>
```

Si noti che nell'esempio viene utilizzata la funzionalità di ridimensionamento condiviso di **Grid**. Le ultime tre colonne sfruttano i vantaggi di questa operazione posizionandosi nello stesso **SharedSizeGroup**. Come si evince dal nome della proprietà, in questo modo le colonne possono condividere le stesse dimensioni. Quindi, quando "Sfoglia..." viene localizzato nella stringa più lunga "durchsuchen...", tutti i pulsanti crescono in larghezza anziché avere un piccolo pulsante "OK" e un "durchsuchen..." molto grande pulsante.

XML: lang

```
xml:lang="en-US"
```

Si noti la **gestione di XML: lang in XAML** posizionata in corrispondenza dell'elemento radice del Interfaccia utente. Questa proprietà descrive le impostazioni cultura di un determinato elemento e dei relativi elementi figlio. Questo valore viene utilizzato da diverse funzionalità di WPF e deve essere modificato in modo appropriato durante la localizzazione. Questo valore cambia il dizionario in lingua usato per la sillabazione e il controllo ortografico delle parole. Influisce anche sulla visualizzazione delle cifre e sulla selezione del tipo di carattere da usare da parte del sistema di fallback dei tipi di carattere. Infine, la proprietà influisce sulla visualizzazione dei numeri e sulla forma

dei testi scritti in lingue con alfabeti non latini. Il valore predefinito è "en-US".

Creazione di un assembly di risorse satellite

Nel file con estensione csproj:

Modificare il file di `.csproj` e aggiungere il tag seguente a una `<PropertyGroup>` non condizionale:

```
<UICulture>en-US</UICulture>
```

Si noti l'aggiunta di un valore `UICulture`. Quando questa impostazione è impostata su un valore `CultureInfo` valido, ad esempio en-US, la compilazione del progetto genererà un assembly satellite con tutte le risorse localizzabili al suo interno.

```
<Resource Include="RunIcon.JPG">  
  <Localizable>False</Localizable>  
</Resource>
```

Non è necessario localizzare il `RunIcon.JPG` perché dovrebbe apparire lo stesso per tutte le impostazioni cultura. `Localizable` è impostato su `false` in modo che rimanga nell'assembly principale indipendente dalla lingua anziché nell'assembly satellite. Il valore predefinito di tutte le risorse non compilabili è `Localizable` impostato su `true`.

Localizzazione della finestra di dialogo Eseguì

Analisi

Dopo aver compilato l'applicazione, il primo passaggio della localizzazione consiste nell'analizzare le risorse localizzabili nell'assembly satellite. Ai fini di questo argomento, usare lo strumento LocBaml di esempio, disponibile nell'esempio di [strumento LocBaml](#). Si noti che LocBaml è solo uno strumento di esempio che permette di iniziare a creare uno strumento di localizzazione adatto al processo di localizzazione. Usando LocBaml, eseguire il comando seguente per analizzare: **LocBaml/parse RunDialog.resources.dll/out:** per generare un file "RunDialog.resources.dll.csv".

Localizzazione

Usare un editor CSV che supporta la codifica Unicode per modificare il file. Escludere tutte le voci con categoria di localizzazione "Nessuna". Dovrebbero essere visualizzate le voci seguenti:

CHIAVE DI RISORSA	CATEGORIA DI LOCALIZZAZIONE	VALORE
Button_1:System.Windows.Controls.Button.\$Content	Pulsante	OK
Button_2:System.Windows.Controls.Button.\$Content	Pulsante	Annulla
Button_3:System.Windows.Controls.Button.\$Content	Pulsante	Sfoglia...
ComboBox_1:System.Windows.Controls.ComboBox.\$Content	ComboBox	
TextBlock_1:System.Windows.Controls.TextBlock.\$Content	Text	Digitare il nome del programma, della cartella, del documento o della risorsa Internet da aprire.

CHIAVE DI RISORSA	CATEGORIA DI LOCALIZZAZIONE	VALORE
TextBlock_2:System.Windows.Controls.TextBlock.\$Content	Text	Apri:
Window_1:System.Windows.Window.Title	Titolo	Esegui

La localizzazione dell'applicazione in tedesco richiede le seguenti traduzioni:

CHIAVE DI RISORSA	CATEGORIA DI LOCALIZZAZIONE	VALORE
Button_1:System.Windows.Controls.Button.\$Content	Pulsante	OK
Button_2:System.Windows.Controls.Button.\$Content	Pulsante	Abbrechen
Button_3:System.Windows.Controls.Button.\$Content	Pulsante	Durchsuchen...
ComboBox_1:System.Windows.Controls.ComboBox.ComboBox.\$Content	ComboBox	
TextBlock_1:System.Windows.Controls.TextBlock.\$Content	Text	Geben Sie den Namen eines programmes, Ordners, Dokuments oder einer Internetresource an.
TextBlock_2:System.Windows.Controls.TextBlock.\$Content	Text	Apri:
Window_1:System.Windows.Window.Title	Titolo	Esegui

Generazione

L'ultimo passaggio della localizzazione implica la creazione dell'assembly satellite appena localizzato. Per questa operazione usare il comando LocBaml seguente:

LocBaml.exe /generate RunDialog.resources.dll /trans:RunDialog.resources.dll.CSV /out: ./cul:de-DE

Nelle finestre tedesche, se il file resources.dll viene inserito in una cartella de-DE accanto all'assembly principale, questa risorsa viene caricata automaticamente al posto di quella nella cartella en-US. Se non si dispone di una versione tedesca di Windows per testarla, impostare le impostazioni cultura per le impostazioni cultura di Windows in uso (ad esempio, `en-US`) e sostituire la DLL di risorse originali.

Caricamento di risorse satellite

MYDIALOG.EXE	EN-US\MYDIALOG.RESOURCES.DLL	DE-DE\MYDIALOG.RESOURCES.DLL
Codice	BAML inglese originale	BAML localizzato
Risorse indipendenti dalle impostazioni cultura	Altre risorse in inglese	Altre risorse localizzate in tedesco

.NET sceglie automaticamente l'assembly di risorse satellite da caricare in base

all'[Thread.CurrentCulture](#)dell'applicazione. Per impostazione predefinita, vengono utilizzate le impostazioni cultura del sistema operativo Windows. Se si usano le finestre tedesche, il file *de-DE\MyDialog.resources.dll* viene caricato. Se si usa Windows in lingua inglese, il file *en-US\MyDialog.resources.dll* viene caricato. È possibile impostare la risorsa di fallback finale per l'applicazione specificando l'attributo [NeutralResourcesLanguage](#) nel file *AssemblyInfo* del progetto. Ad esempio, se si specifica:

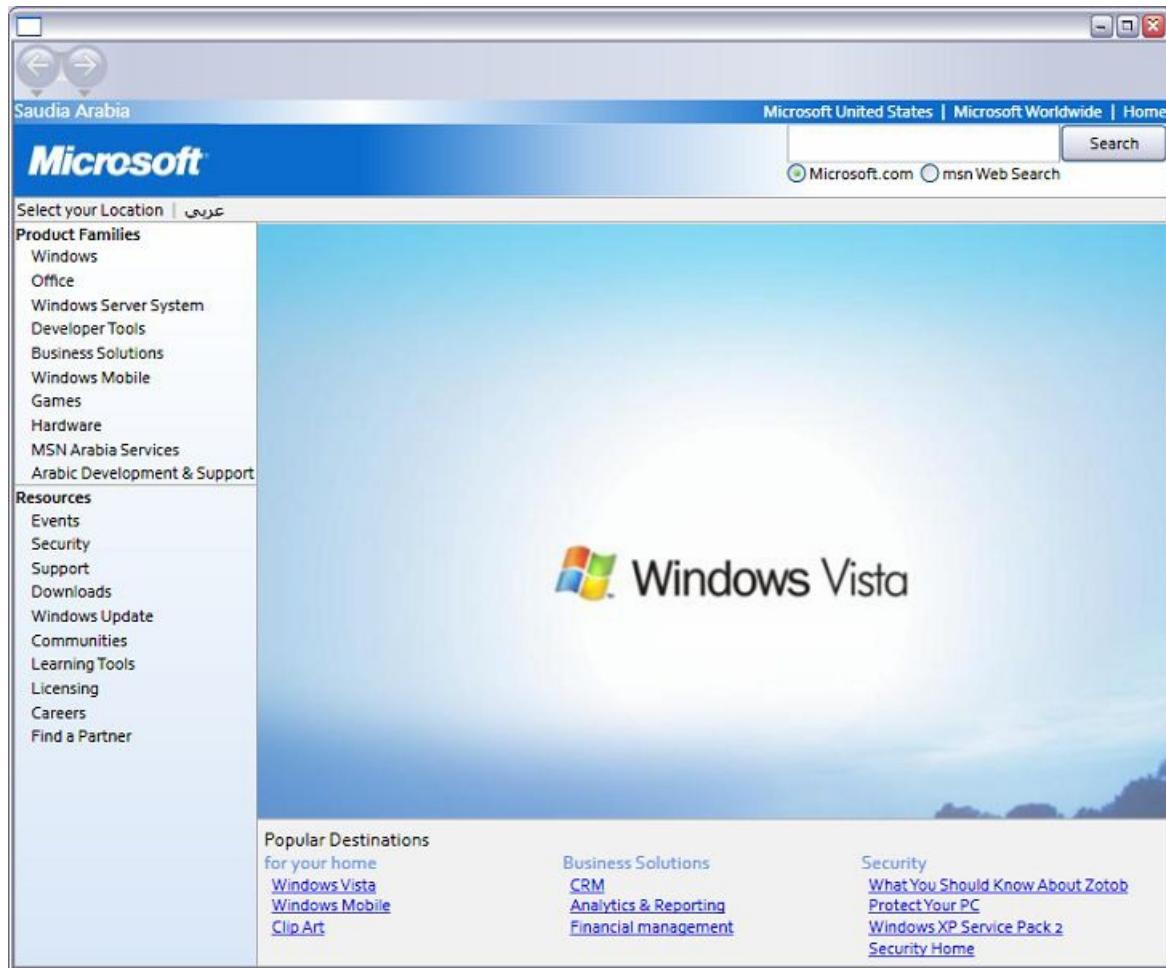
```
[assembly: NeutralResourcesLanguage("en-US", UltimateResourceFallbackLocation.Satellite)]
```

il file *en-US\MyDialog.resources.dll* viene quindi utilizzato con le finestre tedesche se non sono disponibili nessuno dei seguenti file: *de-DE\MyDialog.resources.dll* o *DE\MyDialog.resources.dll*.

Home page Microsoft per l'Arabia Saudita

Le immagini seguenti mostrano una home page in inglese e in arabo. Per l'esempio completo che produce questi elementi grafici, vedere [esempio di globalizzazione della Home page](#).

Inglese:



Arabo:



Progettazione di un home page Microsoft globale

Questo modello di sito Web Microsoft per l'Arabia Saudita illustra le funzionalità di globalizzazione disponibili per le lingue da destra a sinistra. Lingue quali l'ebraico e l'arabo hanno un ordine di lettura da destra a sinistra, quindi il layout di Interfaccia utente deve essere spesso disposto in modo molto diverso rispetto a quello delle lingue da sinistra a destra, ad esempio l'inglese. La localizzazione da una lingua da sinistra a destra in una lingua da destra a sinistra o viceversa può risultare piuttosto complessa. WPF è stato progettato per semplificare tali processi di localizzazione.

FlowDirection

Homepage.xaml:

```
<Page x:Uid="Page_1" x:Class="MicrosoftSaudiArabiaHomepage.Homepage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      FlowDirection="LeftToRight"
      Localization.Comments="FlowDirection(This FlowDirection controls the actual content of the homepage)"
      xml:lang="en-US">
```

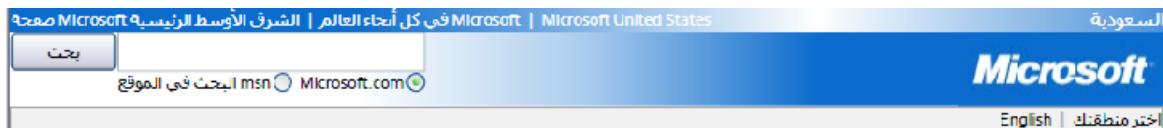
Si noti la proprietà **FlowDirection** in **Page**. Se si modifica questa proprietà in **RightToLeft**, il **FlowDirection** del **Page** e dei relativi elementi figlio verrà modificato in modo che il layout di questo Interfaccia utente venga capovolto per diventare da destra a sinistra come previsto da un utente arabo. È possibile eseguire l'override del comportamento di ereditarietà specificando un **FlowDirection** esplicito su qualsiasi elemento. La proprietà **FlowDirection** è disponibile in qualsiasi elemento correlato **FrameworkElement** o documento e ha un valore implicito di **LeftToRight**.

Osservare che anche i pennelli sfumatura di sfondo vengono capovolti correttamente quando viene modificata la **FlowDirection** radice:

FlowDirection="LeftToRight"



FlowDirection="RightToLeft"



Evitare l'utilizzo di dimensioni fisse per riquadri e controlli

Esaminare Homepage XAML. si noti che, a parte la larghezza e l'altezza fisse specificate per l'intero Interfaccia utente nella parte superiore **DockPanel**, non sono presenti altre dimensioni fisse. Se si usano dimensioni fisse, è possibile che il testo localizzato risulti più lungo rispetto al testo di origine e debba essere ritagliato. I pannelli e i controlli WPF verranno automaticamente ridimensionati in base al contenuto. La maggior parte dei controlli dispone anche di dimensioni minime e massime che è possibile impostare per un maggiore controllo (ad esempio, `MinWidth = "20"`). Con **Grid**, è anche possibile impostare larghezze e altezze relative usando '*' (ad esempio, `Width="0.25*"`) o usare la funzionalità di condivisione delle dimensioni della cella.

Commenti di localizzazione

In molti casi il contenuto può risultare ambiguo e difficile da tradurre. Lo sviluppatore o il progettista ha la possibilità di fornire ai localizzatori contesto aggiuntivo e commenti tramite i commenti di localizzazione. Ad esempio, la localizzazione. i commenti seguenti chiariscono l'uso del carattere|".

```
<TextBlock  
    x:Uid="TextBlock_2"  
    DockPanel.Dock="Right"  
    Foreground="White"  
    Margin="5,0,5,0"  
    Localization.Comments="$Content(This character is used as a decorative rule.)">  
    |  
</TextBlock>
```

Questo commento viene associato al contenuto di `TextBlock_1` e, nel caso dello strumento LocBaml (vedere [localizzare un'applicazione](#)), può essere visualizzato nella sesta colonna della `TextBlock_1` riga nel file output.csv:

CHIAVE DI RISORSA	CATEGORY	LEGGIBILE	MODIFICABILE	COMMENTO	VALORE
TextBlock_1:System.Windows.Controls.TextBlock.\$Content	Text	TRUE	TRUE	Questo carattere viene usato come regola decorativa.	

I commenti possono essere inseriti nel contenuto o nella proprietà di qualsiasi elemento usando la sintassi seguente:

```

<TextBlock
    x:Uid="TextBlock_1"
    DockPanel.Dock="Right"
    Foreground="White"
    Margin="5,0,5,0"
    Localization.Comments="$Content(This is a comment on the TextBlock's content.)"
        Margin(This is a comment on the TextBlock's Margin property.)">
    |
</TextBlock>

```

Attributi di localizzazione

Spesso lo sviluppatore o il responsabile della localizzazione deve avere il controllo su tutto ciò che i localizzatori possono leggere e modificare. Ad esempio, il localizzatore non deve tradurre il nome della società o modificare alcuni termini legali. WPF fornisce attributi che consentono di impostare la leggibilità, la modificabilità e la categoria del contenuto o della proprietà di un elemento che può essere utilizzato dallo strumento di localizzazione per bloccare, nascondere o ordinare gli elementi. Per altre informazioni, vedere [Attributes](#). Ai fini di questo esempio, lo strumento LocBaml restituisce solo i valori di questi attributi. I controlli WPF hanno tutti i valori predefiniti per questi attributi, ma è possibile eseguirne l'override. Nell'esempio seguente, ad esempio, viene eseguito l'override degli attributi di localizzazione predefiniti per `TextBlock_1` e il contenuto viene impostato come leggibile ma non modificabile per i localizzatori.

```

<TextBlock
    x:Uid="TextBlock_1"
    Localization.Attributes=
    "$Content(Readable Unmodifiable)">
    Microsoft Corporation
</TextBlock>

```

Oltre agli attributi di leggibilità e di modificabilità, WPF fornisce un'enumerazione delle categorie di interfaccia utente comuni ([LocalizationCategory](#)) che possono essere usate per offrire ai localizzatori un contesto maggiore. È possibile eseguire l'override delle WPF categorie predefinite per i controlli della piattaforma anche in XAML:

```

<TextBlock x:Uid="TextBlock_2">
    <TextBlock.ToolTip>
        <TextBlock
            x:Uid="TextBlock_3"
            Localization.Attributes=
            "$Content(ToolTip Readable Unmodifiable)">
                Microsoft Corporation
            </TextBlock>
        </TextBlock.ToolTip>
        Windows Vista
    </TextBlock>

```

Gli attributi di localizzazione predefiniti che WPF fornisce possono anche essere sottoposti a override tramite codice, pertanto è possibile impostare correttamente i valori predefiniti corretti per i controlli personalizzati. Ad esempio,

```

[Localizability(Readability = Readability.Readable, Modifiability=Modifiability.Unmodifiable,
LocalizationCategory.None)]
public class CorporateLogo : TextBlock
{
    // ...
}

```

Gli attributi per istanza impostati in XAML avranno la precedenza sui valori impostati nel codice nei controlli

personalizzati. Per altre informazioni sugli attributi e i commenti, vedere [attributi e commenti di localizzazione](#).

Fallback dei tipi di carattere e tipi di carattere compositi

Se si specifica un tipo di carattere che non supporta un determinato intervallo di punti di riferimento, WPF esegue automaticamente il fallback a quello che esegue mediante l'interfaccia utente globale. compositefont che si trova nella directory Windows\Fonts. I tipi di carattere compositi funzionano come qualsiasi altro tipo di carattere e possono essere usati in modo esplicito impostando la `FontFamily` di un elemento, ad esempio

`FontFamily="Global User Interface"`. Per specificare le preferenze di fallback del tipo di carattere, è possibile creare un tipo di carattere composito personalizzato e specificare il tipo di carattere da usare per gli specifici intervalli di punti di codice e le lingue.

Per ulteriori informazioni sui tipi di carattere compositi, vedere [FontFamily](#).

Localizzazione della home page Microsoft

È possibile seguire la stessa procedura usata nell'esempio della finestra di dialogo Esegui per localizzare l'applicazione. Il file CSV localizzato per l'arabo è disponibile nell'esempio della [Home page di globalizzazione](#).

Globalizzazione per WPF

03/02/2020 • 13 minutes to read • [Edit Online](#)

In questo argomento vengono introdotti i problemi che è necessario conoscere quando si scrive Windows Presentation Foundation (WPF) applicazioni per il mercato globale. Gli elementi di programmazione della globalizzazione sono definiti in .NET nello spazio dei nomi [System.Globalization](#).

Globalizzazione XAML

Extensible Application Markup Language (XAML) si basa su XML e sfrutta il supporto per la globalizzazione definito nella specifica XML. Le sezioni seguenti descrivono alcune funzionalità di XAML di cui è necessario essere a conoscenza.

Riferimenti ai caratteri

Un riferimento a un carattere fornisce l'unità di codice UTF16 del carattere Unicode particolare che rappresenta, in formato decimale o esadecimale. Nell'esempio seguente viene illustrato un riferimento a un carattere decimale per la lettera MAIUSCOLA copto o ' ፳ ':

```
&#1000;
```

Nell'esempio seguente viene illustrato un riferimento a un carattere esadecimale. Si noti che ha una **x** davanti al numero esadecimale.

```
&x3E8;
```

Codifica

La codifica supportata da XAML sono ASCII, Unicode UTF-16 e UTF-8. L'istruzione Encoding si trova all'inizio del documento XAML. Se non esiste alcun attributo di codifica né un ordine dei byte, il parser userà il valore predefinito UTF-8. UTF-8 e UTF-16 sono i tipi di codifica preferiti. UTF-7 non è supportato. Nell'esempio seguente viene illustrato come specificare una codifica UTF-8 in un file di XAML.

```
?xml encoding="UTF-8"?
```

Attributo Language

XAML usa [XML: lang](#) per rappresentare l'attributo Language di un elemento. Per sfruttare i vantaggi della classe [CultureInfo](#), il valore dell'attributo Language deve essere uno dei nomi di impostazioni cultura predefiniti da [CultureInfo.xml:lang](#) è ereditabile nell'albero di elementi (in base alle regole XML, non necessariamente a causa dell'ereditarietà della proprietà di dipendenza) e il valore predefinito è una stringa vuota se non viene assegnato in modo esplicito.

L'attributo language è molto utile per specificare i dialetti. Il francese, ad esempio, ha ortografia, vocabolario e pronuncia diversi in Francia, Quebec, Belgio e Svizzera. Inoltre, il cinese, il giapponese e il coreano condividono punti di codice in Unicode, ma le forme ideogrammi sono diverse e utilizzano tipi di carattere completamente diversi.

Nell'esempio di Extensible Application Markup Language (XAML) seguente viene usato l'attributo Language `fr-CA` per specificare il francese canadese.

```
<TextBlock xml:lang="fr-CA">Découvrir la France</TextBlock>
```

Unicode

XAML supporta tutte le funzionalità Unicode, inclusi i surrogati. Se il set di caratteri può essere mappato a Unicode, è supportato. GB18030, ad esempio, introduce alcuni caratteri con mapping all'estensione A e B per cinese, giapponese e coreano e le coppie di surrogati, quindi è completamente supportato. Un'applicazione WPF può usare [StringInfo](#) per modificare le stringhe senza comprendere se hanno coppie di surrogati o caratteri di combinazione.

Progettazione di un'interfaccia utente internazionale con XAML

In questa sezione vengono descritte interfaccia utente funzionalità da tenere in considerazione durante la scrittura di un'applicazione.

Testo internazionale

WPF include l'elaborazione incorporata per tutti i sistemi di scrittura supportati da Microsoft .NET Framework.

Sono attualmente supportati gli script seguenti:

- Arabo
- Bengali
- Devanagari
- Cirillico
- Greco
- Gujarati
- Gurmukhi
- Ebraico
- Script ideografici
- Kannada
- Lao
- Latino
- Malayalam
- Mongolo
- Odia
- Siriaco
- Tamil
- Telugu
- Thaana
- Thailandese*
- Tibetano

*In questa versione sono supportate la visualizzazione e la modifica del testo in thailandese. L'interruzione parola non è supportata.

Non sono attualmente supportati gli script seguenti:

- Khmer
- Antico hangul coreano
- Myanmar
- Singalese

Tutti i motori di sistema di scrittura supportano i tipi di carattere OpenType. I tipi di carattere OpenType possono includere le tabelle di layout OpenType che consentono agli autori di tipi di carattere di progettare migliori tipi di carattere tipografici internazionali e di fascia alta. Le tabelle di layout del tipo di carattere OpenType contengono informazioni sulle sostituzioni dei glifi, il posizionamento del glifo, la giustificazione e il posizionamento della linea di base, consentendo alle applicazioni di elaborazione del testo di migliorare il layout

I tipi di carattere OpenType consentono la gestione di set di glifi di grandi dimensioni utilizzando la codifica Unicode. Tale codifica consente un esteso supporto internazionale, oltre alle varianti dei glifi tipografici.

WPF rendering del testo è basato sulla tecnologia Microsoft ClearType dei sottapixel che supporta l'indipendenza dalla risoluzione. Questo migliora considerevolmente la leggibilità e consente di supportare documenti in stile rivista di qualità elevata per tutti gli script.

Layout internazionale

WPF offre un modo molto pratico per supportare layout orizzontali, bidirezionale e verticali. Nel Framework di presentazione è possibile utilizzare la proprietà [FlowDirection](#) per definire il layout. I modelli di direzione del flusso sono:

- *LeftToRight*: layout orizzontale per latino, lingue dell'Asia orientale e così via.
- *RightToLeft*: bidirezionale per arabo, ebraico e così via.

Sviluppo di applicazioni localizzabili

Quando si scrive un'applicazione per il consumo globale, è opportuno ricordare che l'applicazione deve essere localizzabile. Gli argomenti seguenti illustrano alcune importanti considerazioni.

Interfaccia utente multilingue

MUI (Multilingual User Interfaces) è un supporto Microsoft per il cambio di interfacce utente da una lingua a un'altra. Un'applicazione WPF utilizza il modello di assembly per supportare MUI. Un'applicazione contiene assembly indipendenti dalla lingua, ma anche assembly di risorse satellite dipendenti dalla lingua. Il punto di ingresso è un file EXE gestito nell'assembly principale. WPF caricatore di risorse sfrutta il Resource Manager del Framework per supportare la ricerca e il fallback delle risorse. Gli assembly satellite di più lingue usano lo stesso assembly principale. L'assembly di risorse caricato dipende dalla [CurrentUICulture](#) del thread corrente.

Interfaccia utente localizzabile

Le applicazioni WPF utilizzano XAML per definire la loro Interfaccia utente. XAML consente agli sviluppatori di specificare una gerarchia di oggetti con un set di proprietà e una logica. L'utilizzo principale del XAML consiste nello sviluppo di applicazioni WPF, ma può essere utilizzato per specificare una gerarchia di tutti gli oggetti Common Language Runtime (CLR). La maggior parte degli sviluppatori USA XAML per specificare il Interfaccia utente dell'applicazione e usare un linguaggio C# di programmazione come per rispondere all'interazione dell'utente.

Da un punto di vista delle risorse, un file di XAML progettato per descrivere una Interfaccia utente dipendente dalla lingua è un elemento di risorsa e pertanto il formato di distribuzione finale deve essere localizzabile per

supportare le lingue internazionali. Poiché XAML non è in grado di gestire gli eventi, molte applicazioni XAML contengono blocchi di codice per eseguire questa operazione. Per ulteriori informazioni, vedere [Cenni preliminari su XAML \(WPF\)](#). Il codice viene rimosso e compilato in file binari diversi quando un file di XAML viene suddiviso in token nel formato BAML di XAML. Il formato BAML dei file XAML, le immagini e altri tipi di oggetti risorsa gestita vengono incorporati nell'assembly di risorse satellite, che può essere localizzato in altre lingue, o nell'assembly principale quando la localizzazione non è necessaria.

NOTE

WPF applicazioni supportano tutte le risorse di FrameworkCLR, tra cui tabelle di stringhe, immagini e così via.

Compilazione di applicazioni localizzabili

La localizzazione consente di adattare un Interfaccia utente a impostazioni cultura diverse. Per rendere localizzabile un'applicazione WPF, gli sviluppatori devono compilare tutte le risorse localizzabili in un assembly di risorse. L'assembly di risorse viene localizzato in lingue diverse e il code-behind usa l'API di gestione delle risorse per il caricamento. Uno dei file necessari per un'applicazione WPF è un file di progetto (con estensione proj). Tutte le risorse usate nell'applicazione devono essere incluse nel file di progetto. L'esempio seguente da un file con estensione csproj illustra come effettuare questa operazione.

```
<Resource Include="data\picture1.jpg"/>
<EmbeddedResource Include="data\stringtable.en-US.resx"/>
```

Per usare una risorsa nell'applicazione, creare un'istanza di un [ResourceManager](#) e caricare la risorsa che si vuole usare. Nell'esempio riportato di seguito viene illustrato come procedere.

```
void OnClick(object sender, RoutedEventArgs e)
{
    ResourceManager rm = new ResourceManager ("MySampleApp.data.stringtable",
        Assembly.GetExecutingAssembly());
    Text1.Text = rm.GetString("Message");
}
```

Uso di ClickOnce con applicazioni localizzate

ClickOnce è una nuova tecnologia di distribuzione Windows Forms che verrà fornita con Visual Studio 2005, che consente l'installazione e l'aggiornamento di applicazioni Web. Quando un'applicazione distribuita con ClickOnce è localizzata, può essere visualizzata solo nelle impostazioni cultura localizzate. Se, ad esempio, un'applicazione distribuita è localizzata in giapponese, può essere visualizzata solo in Windows giapponese non in lingua inglese. Questo problema si presenta perché si tratta di uno scenario comune per gli utenti giapponesi di eseguire una versione in lingua inglese di Windows.

Per risolvere il problema, è necessario impostare l'attributo di fallback su una lingua neutra. Uno sviluppatore di applicazioni può facoltativamente rimuovere risorse dall'assembly principale e specificare che le risorse sono disponibili in un assembly satellite corrispondente a impostazioni cultura specifiche. Per controllare questo processo, usare il [NeutralResourcesLanguageAttribute](#). Il costruttore della classe [NeutralResourcesLanguageAttribute](#) dispone di due firme, una che accetta un parametro di [UltimateResourceFallbackLocation](#) per specificare il percorso in cui il [ResourceManager](#) deve estrarre le risorse di fallback: assembly principale o assembly satellite. L'esempio seguente mostra come usare l'attributo. Per il percorso di fallback finale, il codice fa in modo che il [ResourceManager](#) cerchi le risorse nella sottodirectory "de" della directory dell'assembly attualmente in esecuzione.

```
[assembly: NeutralResourcesLanguageAttribute(  
    "de" , UltimateResourceFallbackLocation.Satellite)]
```

Vedere anche

- [Cenni preliminari sulla globalizzazione e localizzazione WPF](#)

Cenni preliminari sull'utilizzo del layout automatico

23/10/2019 • 8 minutes to read • [Edit Online](#)

In questo argomento vengono presentate le linee guida per gli sviluppatori su come scrivere applicazioni Windows Presentation Foundation (WPF) con interfacce utente localizzabili (UI). In passato, la localizzazione di un'interfaccia utente era un processo che richiedeva molto tempo. Ogni lingua in cui l'interfaccia utente è stata adattata per la regolazione del pixel è necessaria. Oggi, con la progettazione corretta e gli standard di codifica corretti, le interfacce utente possono essere costruite in modo che i localizzatori abbiano un minor ridimensionamento e un riposizionamento. L'approccio alla scrittura di applicazioni che possono essere ridimensionate e riposizionate più facilmente viene definito layout automatico e può essere eseguito utilizzando la progettazione di applicazioni WPF.

Vantaggi dell'uso del layout automatico

Poiché il sistema di presentazione WPF è potente e flessibile, offre la possibilità di applicare il layout degli elementi di un'applicazione che possono essere modificati in base ai requisiti di lingue diverse. L'elenco seguente indica alcuni vantaggi del layout automatico.

- L'interfaccia utente viene visualizzata correttamente in qualsiasi lingua.
- Riduce l'esigenza di ulteriori modifiche alla posizione e alle dimensioni dei controlli dopo la traduzione del testo.
- Riduce l'esigenza di ulteriori modifiche alle dimensioni delle finestre.
- Il layout dell'interfaccia utente viene visualizzato correttamente in qualsiasi lingua.
- La localizzazione può essere ridotta a semplici attività che vanno poco oltre la traduzione delle stringhe.

Layout automatico e controlli

Il layout automatico consente di modificare automaticamente le dimensioni di un controllo in un'applicazione. Ad esempio, un controllo può cambiare in base alla lunghezza di una stringa. Questa funzionalità consente ai localizzatori di tradurre la stringa senza dover ridimensionare il controllo per adattarlo al testo tradotto. L'esempio seguente crea un pulsante con contenuto in lingua inglese.

```
<Window  
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
    x:Class="ButtonLoc.Panel1"  
    Name="myWindow"  
    SizeToContent="WidthAndHeight"  
    >  
  
<DockPanel>  
    <Button FontSize="28" Height="50">My name is Hope.</Button>  
</DockPanel>  
</Window>
```

Nell'esempio, l'unica operazione da compiere per creare un pulsante in lingua spagnola è modificare il testo. Ad esempio,

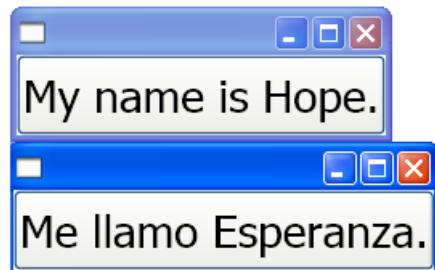
```

<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="ButtonLoc.Pane1"
    Name="myWindow"
    SizeToContent="WidthAndHeight"
    >

    <DockPanel>
        <Button FontSize="28" Height="50">Me llamo Esperanza.</Button>
    </DockPanel>
</Window>

```

Il grafico seguente mostra l'output degli esempi di codice:



Layout automatico e standard di codifica

L'uso dell'approccio di layout automatico richiede un set di regole e standard di codifica e progettazione per produrre un'interfaccia utente completamente localizzabile. Le linee guida riportate di seguito agevolano la codifica del layout automatico.

Non usare posizioni assolute

- Non usare [Canvas](#) perché posiziona gli elementi in modo assoluto.
- Usare [DockPanel](#), [StackPanel](#) e [Grid](#) per posizionare i controlli.

Per una descrizione dei vari tipi di pannelli, vedere [Cenni preliminari sui pannelli](#).

Non impostare dimensioni fisse per una finestra

- Usare [Window.SizeToContent](#). Esempio:

```

<StackPanel
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="GridLoc.Pane1"
    >

```

Aggiungere un FlowDirection

- Aggiungere un [FlowDirection](#) all'elemento radice dell'applicazione.

WPF offre un modo pratico per supportare layout orizzontali, bidirezionali e verticali. Nel Framework di presentazione, è possibile usare la proprietà [FlowDirection](#) per definire il layout. I modelli di direzione del flusso sono:

- [FlowDirection.LeftToRight](#) (LrTb): layout orizzontale per il latino, l'Asia orientale e così via.
- [FlowDirection.RightToLeft](#) (RITb): bidirezionale per l'arabo, l'ebraico e così via.

Usare tipi di carattere compositi invece di tipi di carattere fisici

- Con i tipi di carattere compositi non è necessario localizzare la proprietà [FontFamily](#).
- Gli sviluppatori possono usare uno dei tipi di carattere riportati di seguito oppure crearne uno personalizzato.
 - Interfaccia utente globale
 - Global San Serif
 - Global Serif

Aggiungi XML: lang

- Aggiungere l'attributo `xml:lang` nell'elemento radice dell'interfaccia utente, ad esempio `xml:lang="en-US"` per un'applicazione in lingua inglese.
- Poiché i tipi di carattere composito utilizzano `xml:lang` per determinare il tipo di carattere da utilizzare, impostare questa proprietà per supportare scenari multilingue.

Layout automatico e griglie

L'elemento [Grid](#) è utile per il layout automatico perché consente agli sviluppatori di posizionare gli elementi. Un controllo [Grid](#) è in grado di distribuire lo spazio disponibile tra gli elementi figlio, usando una disposizione di colonne e righe. Gli elementi dell'interfaccia utente possono estendersi su più celle ed è possibile disporre di griglie all'interno di griglie. Le griglie sono utili perché consentono di creare e posizionare interfacce utente complesse. L'esempio seguente illustra l'uso di una griglia per posizionare alcuni pulsanti e del testo. Si noti che l'altezza e la larghezza delle celle sono impostate su [Auto](#); Pertanto, la cella che contiene il pulsante con un'immagine viene modificata in base all'immagine.

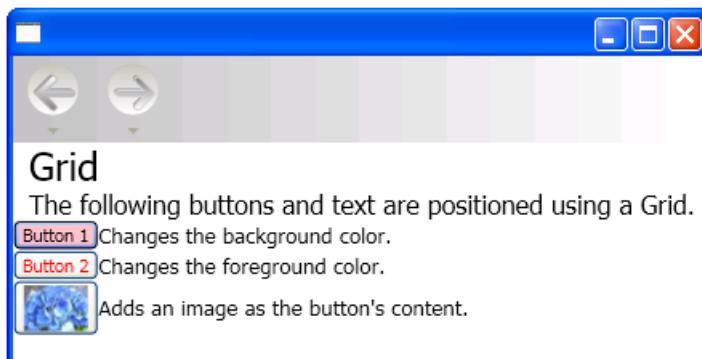
```

<Grid Name="grid" ShowGridLines ="false">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>

<TextBlock Margin="10, 10, 5, 5" Grid.Column="0" Grid.Row="0" FontSize="24">Grid
</TextBlock>
<TextBlock Margin="10, 10, 5, 5" Grid.Column="0" Grid.Row="1" FontSize="12"
    Grid.ColumnSpan="2">The following buttons and text are positioned using a Grid.
</TextBlock>
<Button Margin="10, 10, 5, 5" Grid.Column="0" Grid.Row="2" Background="Pink"
    BorderBrush="Black" BorderThickness="10">Button 1
</Button>
<TextBlock Margin="10, 10, 5, 5" Grid.Column="1" Grid.Row="2" FontSize="12"
    VerticalAlignment="Center" TextWrapping="WrapWithOverflow">Sets the background
    color.
</TextBlock>
<Button Margin="10, 10, 5, 5" Grid.Column="0" Grid.Row="3" Foreground="Red">
    Button 2
</Button>
<TextBlock Margin="10, 10, 5, 5" Grid.Column="1" Grid.Row="3" FontSize="12"
    VerticalAlignment="Center" TextWrapping="WrapWithOverflow">Sets the foreground
    color.
</TextBlock>
<Button Margin="10, 10, 5, 5" Grid.Column="0" Grid.Row="4">
    <Image Source="data\flower.jpg"/>
</Button>
<TextBlock Margin="10, 10, 5, 5" Grid.Column="1" Grid.Row="4" FontSize="12"
    VerticalAlignment="Center" TextWrapping="WrapWithOverflow">Adds an image as
    the button's content.
</TextBlock>
</Grid>

```

La figura seguente illustra la griglia prodotta dal codice precedente.



Griglia di

Layout automatico e griglie basate sull'uso della proprietà IsSharedSizeScope

Un elemento [Grid](#) è utile nelle applicazioni localizzabili per creare controlli che si adattano al contenuto. In alcuni casi, tuttavia, può essere opportuno che i controlli mantengano una determinata dimensione indipendentemente dal contenuto. Ad esempio, nei casi dei pulsanti "OK", "Annulla" e "Sfoglia", probabilmente non si vuole che le dimensioni si adattino al contenuto. In questo caso la proprietà associata [Grid.IsSharedSizeScope](#) è utile per

condividere le stesse dimensioni tra più elementi della griglia. Nell'esempio seguente viene illustrato come condividere dati di ridimensionamento di colonne e righe tra più elementi *Grid*.

```
<StackPanel Orientation="Horizontal" DockPanel.Dock="Top">
    <Button Click="setTrue" Margin="0,0,10,10">Set IsSharedSizeScope="True"</Button>
    <Button Click="setFalse" Margin="0,0,10,10">Set IsSharedSizeScope="False"</Button>
</StackPanel>

<StackPanel Orientation="Horizontal" DockPanel.Dock="Top">

    <Grid ShowGridLines="True" Margin="0,0,10,0">
        <Grid.ColumnDefinitions>
            <ColumnDefinition SharedSizeGroup="FirstColumn"/>
            <ColumnDefinition SharedSizeGroup="SecondColumn"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" SharedSizeGroup="FirstRow"/>
        </Grid.RowDefinitions>

        <Rectangle Fill="Silver" Grid.Column="0" Grid.Row="0" Width="200" Height="100"/>
        <Rectangle Fill="Blue" Grid.Column="1" Grid.Row="0" Width="150" Height="100"/>

        <TextBlock Grid.Column="0" Grid.Row="0" FontWeight="Bold">First Column</TextBlock>
        <TextBlock Grid.Column="1" Grid.Row="0" FontWeight="Bold">Second Column</TextBlock>
    </Grid>

    <Grid ShowGridLines="True">
        <Grid.ColumnDefinitions>
            <ColumnDefinition SharedSizeGroup="FirstColumn"/>
            <ColumnDefinition SharedSizeGroup="SecondColumn"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" SharedSizeGroup="FirstRow"/>
        </Grid.RowDefinitions>

        <Rectangle Fill="Silver" Grid.Column="0" Grid.Row="0"/>
        <Rectangle Fill="Blue" Grid.Column="1" Grid.Row="0"/>

        <TextBlock Grid.Column="0" Grid.Row="0" FontWeight="Bold">First Column</TextBlock>
        <TextBlock Grid.Column="1" Grid.Row="0" FontWeight="Bold">Second Column</TextBlock>
    </Grid>
</StackPanel>

<TextBlock Margin="10" DockPanel.Dock="Top" Name="txt1"/>
```

NOTE

Per l'esempio di codice completo, vedere [condividere le proprietà di ridimensionamento tra le griglie](#).

Vedere anche

- [Globalizzazione per WPF](#)
- [Utilizzare un layout automatico per creare un pulsante](#)
- [Usare una griglia per il layout automatico](#)

Attributi e commenti di localizzazione

23/10/2019 • 8 minutes to read • [Edit Online](#)

i commenti di localizzazione Windows Presentation Foundation (WPF) sono proprietà, all'interno del codice sorgente XAML, fornite dagli sviluppatori per fornire regole e suggerimenti per la localizzazione. I commenti di localizzazione Windows Presentation Foundation (WPF) contengono due set di informazioni: attributi di localizzabilità e commenti di localizzazione in formato libero. Gli attributi di localizzabilità vengono usati dall'API di localizzazione di WPF per indicare le risorse da localizzare. I commenti in formato libero comprendono tutte le informazioni che l'autore dell'applicazione vuole includere.

Commenti di localizzazione

Se gli autori dell'applicazione di markup hanno requisiti per elementi specifici in XAML, ad esempio vincoli sulla lunghezza del testo, la famiglia di caratteri o la dimensione del carattere, possono inviare tali informazioni ai localizzatori con commenti nel codice XAML. Il processo di aggiunta di commenti al codice sorgente è il seguente:

1. Lo sviluppatore di applicazioni aggiunge commenti di localizzazione al codice sorgente XAML.
2. Durante il processo di compilazione, nel file con estensione proj è possibile specificare se mantenere i commenti di localizzazione in formato libero nell'assembly, se rimuoverne una parte o se rimuoverli tutti. I commenti rimossi vengono inseriti in un file separato. Per indicare la scelta si usa un tag `LocalizationDirectivesToLocFile`, ad esempio:

```
<LocalizationDirectivesToLocFile> valore </LocalizationDirectivesToLocFile>
```
3. I valori che è possibile assegnare sono:
 - **None**: sia i commenti che gli attributi rimangono nell'assembly e non viene generato alcun file separato.
 - **CommentsOnly**: dall'assembly vengono rimossi solo i commenti, che vengono inseriti nel LocFile separato.
 - **All**: dall'assembly vengono rimossi sia i commenti che gli attributi, che vengono inseriti entrambi in un LocFile separato.
4. Quando le risorse localizzabili vengono estratte da BAML, gli attributi di localizzabilità vengono rispettati dall'API di localizzazione BAML.
5. I file dei commenti di localizzazione, nei quali sono contenuti solo commenti in formato libero, vengono incorporati al processo di localizzazione in un secondo momento.

Nell'esempio seguente viene illustrato come aggiungere commenti di localizzazione a un file XAML.

```
<TextBlock x:Id = "text01">  
  FontFamily = "Microsoft Sans Serif"  
  FontSize = "12"  
  Localization.Attributes = "$Content (Unmodifiable Readable Text)"  
  FontFamily (Unmodifiable Readable)
```

```

Localization.Comments = "$Content (Trademark)

FontSize (Trademark font size)" >

Microsoft

</TextBlock>

```

Nella sezione Localization.Attributes dell'esempio precedente sono disponibili gli attributi di localizzazione, mentre nella sezione Localization.Comments sono disponibili i commenti in formato libero. La tabella seguente illustra gli attributi, i commenti e il relativo significato per il localizzatore.

ATTRIBUTI DI LOCALIZZAZIONE	SIGNIFICATO
\$Content (testo leggibile non modificabile)	Il contenuto dell'elemento TextBlock non può essere modificato. I localizzatori non possono modificare la parola "Microsoft". Il contenuto è visibile (leggibile) per il localizzatore. La categoria del contenuto è testo.
FontFamily (leggibile non modificabile)	La proprietà della famiglia di caratteri dell'elemento TextBlock non può essere modificata, ma è visibile per il localizzatore.
COMMENTI DI LOCALIZZAZIONE IN FORMATO LIBERO	SIGNIFICATO
\$Content (marchio)	L'autore dell'applicazione indica al localizzatore che il contenuto dell'elemento TextBlock è un marchio.
FontSize (dimensioni del carattere del marchio)	L'autore dell'applicazione indica che la proprietà delle dimensioni del carattere deve rispettare la dimensione standard del marchio.

Attributi di localizzabilità

Le informazioni disponibili nella sezione Localization.Attributes contengono un elenco di coppie: il nome del valore di destinazione e i valori di localizzabilità associati. Il nome di destinazione può essere un nome di proprietà o il nome speciale \$Content. Se si tratta di un nome di proprietà, il valore di destinazione è il valore della proprietà. Se si tratta di \$Content, il valore di destinazione è il contenuto dell'elemento.

Sono disponibili tre tipi di attributi:

- **Category.** Specifica se un valore può essere modificato con uno strumento del localizzatore. Vedere [Category](#).
- **Leggibilità.** Specifica se uno strumento del localizzatore deve essere in grado di leggere (e visualizzare) un valore. Vedere [Readability](#).
- **Modificabilità.** Specifica se uno strumento del localizzatore consente la modifica di un valore. Vedere [Modifiability](#).

Questi attributi possono essere specificati in qualsiasi ordine e devono essere delimitati da uno spazio. Nel caso in cui vengano specificati attributi duplicati, l'ultimo sostituisce i precedenti. Ad esempio, Localization.Attributes = "Unmodifiable Modifiable" imposta Modifiability su Modifiable poiché questo è l'ultimo valore.

Gli attributi Modifiability e Readability sono di facile comprensione. L'attributo Category fornisce categorie predefinite che supportano il localizzatore nella traduzione del testo. Categorie quali Text, Label e Title offrono al localizzatore informazioni sulla modalità di traduzione del testo. Sono disponibili anche categorie speciali: None, inherit, ignore e NeverLocalize.

La tabella seguente illustra il significato delle categorie speciali.

CATEGORY	SIGNIFICATO
nessuno	Per il valore di destinazione non è definita alcuna categoria.
Eredita	Il valore di destinazione eredita la categoria dall'elemento padre.
Ignora	Il valore di destinazione viene ignorato nel processo di localizzazione. Questa categoria influisce solo sul valore corrente e non sui nodi figlio.
NeverLocalize	Il valore corrente non può essere localizzato. Questa categoria viene ereditata dagli elementi figlio di un elemento.

Commenti di localizzazione

La sezione Localization.Comments contiene le stringhe in formato libero relative al valore di destinazione. Gli sviluppatori dell'applicazione possono aggiungere informazioni per offrire ai localizzatori suggerimenti sulla modalità di traduzione del testo delle applicazioni. Il formato dei commenti può essere una qualsiasi stringa racchiusa tra "()." Usare '\' come carattere di escape.

Vedere anche

- [Globalizzazione per WPF](#)
- [Utilizzare un layout automatico per creare un pulsante](#)
- [Usare una griglia per il layout automatico](#)
- [Localizzare un'applicazione](#)

Cenni preliminari sulle funzionalità bidirezionali di WPF

03/02/2020 • 26 minutes to read • [Edit Online](#)

Diversamente da qualsiasi altra piattaforma di sviluppo, WPF dispone di molte funzionalità che supportano lo sviluppo rapido di contenuti bidirezionali, ad esempio, i dati misti da sinistra a destra e da destra a sinistra nello stesso documento. Allo stesso tempo, WPF crea un'esperienza ottimale per gli utenti che richiedono funzionalità bidirezionali, ad esempio gli utenti in lingua araba ed ebraica.

Le sezioni seguenti illustrano molte funzionalità bidirezionali con i relativi esempi in cui viene descritto come ottenere la migliore visualizzazione di contenuto bidirezionale. La maggior parte degli esempi usa XAML, sebbene sia possibile applicare facilmente i concetti C# a o al codice di Microsoft Visual Basic.

FlowDirection

La proprietà di base che definisce la direzione del flusso del contenuto in un'applicazione WPF è [FlowDirection](#). Questa proprietà può essere impostata su uno dei due valori di enumerazione, [LeftToRight](#) o [RightToLeft](#). La proprietà è disponibile per tutti gli elementi WPF che ereditano da [FrameworkElement](#).

Negli esempi seguenti viene impostata la direzione del flusso di un elemento [TextBox](#).

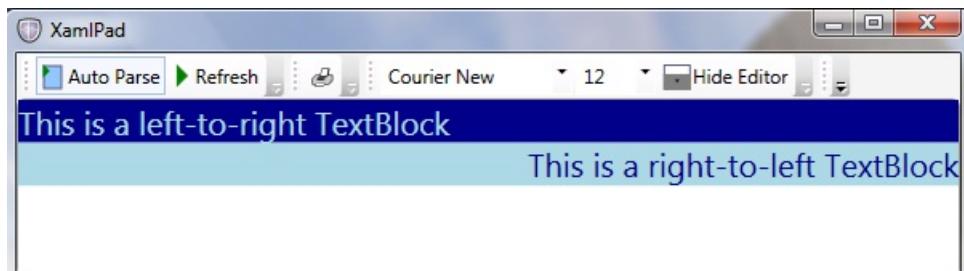
Direzione di flusso da sinistra a destra

```
<TextBlock Background="DarkBlue" Foreground="LightBlue"
FontSize="20" FlowDirection="LeftToRight">
    This is a left-to-right TextBlock
</TextBlock>
```

Direzione di flusso da destra a sinistra

```
<TextBlock Background="LightBlue" Foreground="DarkBlue"
FontSize="20" FlowDirection="RightToLeft">
    This is a right-to-left TextBlock
</TextBlock>
```

La figura seguente illustra il rendering del codice precedente.



Un elemento all'interno di un interfaccia utente albero erederà l'[FlowDirection](#) dal relativo contenitore. Nell'esempio seguente, il [TextBlock](#) si trova all'interno di un [Grid](#), che risiede in un [Window](#). L'impostazione del [FlowDirection](#) per il [Window](#) implica la relativa impostazione anche per il [Grid](#) e [TextBlock](#).

Nell'esempio seguente viene illustrata l'impostazione [FlowDirection](#).

```

<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="FlowDirectionApp.Window1"
    Title="BidiFeatures" Height="200" Width="700"
    FlowDirection="RightToLeft">

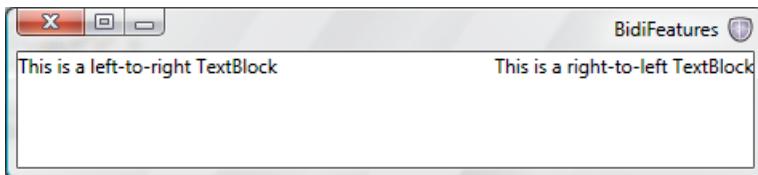
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <TextBlock Grid.Column="0" >
            This is a right-to-left TextBlock
        </TextBlock>

        <TextBlock Grid.Column="1" FlowDirection="LeftToRight">
            This is a left-to-right TextBlock
        </TextBlock>
    </Grid>
</Window>

```

Il [Window](#) di primo livello ha un [FlowDirection](#) di [RightToLeft](#), quindi tutti gli elementi in esso contenuti ereditano anche lo stesso [FlowDirection](#). Affinché un elemento esegua l'override di un [FlowDirection](#) specificato, deve aggiungere una modifica della direzione esplicita, ad esempio la seconda [TextBlock](#) nell'esempio precedente, che cambia in [LeftToRight](#). Quando non viene definito alcun [FlowDirection](#), viene applicata la [LeftToRight](#) predefinita.

Il grafico seguente mostra l'output dell'esempio precedente:



FlowDocument

Molte piattaforme di sviluppo, ad esempio HTML, Win32 e Java forniscono supporto speciale per lo sviluppo di contenuto bidirezionale. I linguaggi di markup, ad esempio HTML, assegnano ai writer di contenuti il markup necessario per visualizzare il testo in qualsiasi direzione, ad esempio il tag HTML 4.0, "dir", che accetta "RTL" o "ltr" come valori. Questo tag è simile alla proprietà [FlowDirection](#), ma la proprietà [FlowDirection](#) funziona in modo più avanzato per il layout del contenuto testuale e può essere usata per contenuti diversi dal testo.

In WPF, una [FlowDocument](#) è un elemento Interfaccia utente versatile che può ospitare una combinazione di testo, tabelle, immagini e altri elementi. Gli esempi nelle sezioni seguenti usano questo elemento.

L'aggiunta di testo a un [FlowDocument](#) può essere eseguita in un modo più approfondito. Un modo semplice per eseguire questa operazione consiste nell'utilizzare un [Paragraph](#) che è un elemento a livello di blocco utilizzato per raggruppare contenuto come testo. Per aggiungere testo a elementi a livello di riga, gli esempi utilizzano [Span](#) e [Run](#). [Span](#) è un elemento di contenuto del flusso di livello inline usato per raggruppare altri elementi inline, mentre una [Run](#) è un elemento di contenuto del flusso di livello inline destinato a contenere un'esecuzione di testo non formattato. Un [Span](#) può contenere più elementi [Run](#).

Il primo documento di esempio contiene un documento con un numero di nomi di condivisione di rete; ad esempio `\server1\folder\file.ext`. Indipendentemente dal fatto che questo collegamento di rete si trovi in un documento in arabo o in inglese, si vuole che venga sempre visualizzato nello stesso modo. Il grafico seguente illustra l'uso dell'elemento [span](#) e Mostra il collegamento in un documento di [RightToLeft](#) arabo:

The screenshot shows the XamlPad application interface. The main window displays a WPF XAML document with the following code:

```
<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    FlowDirection="RightToLeft">
    <FlowDocument>
        <Paragraph>
            <Span FlowDirection="RightToLeft" >
                ستجد الملف هنا:\\server1\filename\filename1.txt
                ثم باقى النص!
            </Span>
        </Paragraph>
    </FlowDocument>
</Page>
```

The output pane shows the rendered text in Arabic, where the first line is right-to-left and the second line starts with "ثم باقى النص!" (Then the rest of the text!). A message at the bottom indicates the markup was saved.

Poiché il testo è **RightToLeft**, tutti i caratteri speciali, ad esempio "\", separano il testo in un ordine da destra a sinistra. Ciò comporta che il collegamento non venga visualizzato nell'ordine corretto, pertanto per risolvere il problema, è necessario incorporare il testo per mantenere un flusso di **Run** separato **LeftToRight**. Invece di avere un **Run** separato per ogni lingua, un modo migliore per risolvere il problema consiste nell'incorporare il testo in lingua inglese usato meno di frequente in una **Span** araba più grande.

Nell'immagine seguente viene illustrato questo problema utilizzando l'elemento Run incorporato in un elemento span:

The screenshot shows the XamlPad application interface. The main window displays a WPF XAML document with the following code:

```
<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    FlowDirection="RightToLeft">
    <FlowDocument>
        <Paragraph>
            <Span FlowDirection="RightToLeft" >
                ستجد الملف هنا:\\server1\filename\filename1.txt
                <Run FlowDirection="LeftToRight">\server1\filename\filename1.txt</Run>
                ثم باقى النص!
            </Span>
        </Paragraph>
    </FlowDocument>
</Page>
```

The output pane shows the rendered text in Arabic, where the first line is right-to-left and the second line contains a link "\server1\filename\filename1.txt" which is displayed correctly. A message at the bottom indicates the markup was saved.

Nell'esempio seguente viene illustrato l'utilizzo di **Run** e **Span** elementi nei documenti.

```

<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    FlowDirection="RightToLeft">

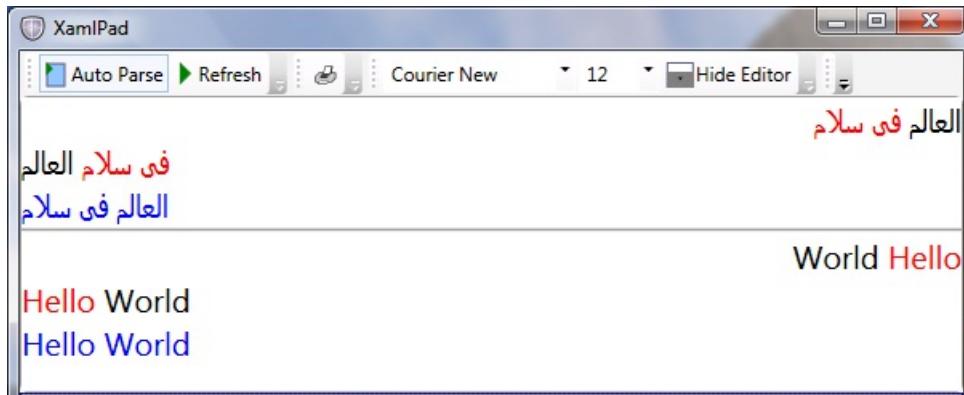
    <FlowDocument>
        <Paragraph>
            <Span FlowDirection="RightToLeft" >
                ستجد الملف هنا:
                <Run FlowDirection="LeftToRight">
                    \\server1\filename\filename1.txt</Run>
                ثم باقى النص!
            </Span>
        </Paragraph>
    </FlowDocument>
</Page>

```

Elementi Span

L'elemento **Span** funziona come separatore di limiti tra testi con direzioni di flusso diverse. Anche **Span** elementi con la stessa direzione del flusso sono considerati con ambiti bidirezionali diversi, il che significa che gli elementi **Span** sono ordinati nel **FlowDirection** del contenitore, solo il contenuto all'interno dell'elemento **Span** segue il **FlowDirection** del **Span**.

Il grafico seguente mostra la direzione del flusso di diversi elementi **TextBlock**.



Nell'esempio seguente viene illustrato come utilizzare gli elementi **Span** e **Run** per produrre i risultati mostrati nel grafico precedente.

```

<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
<StackPanel >

    <TextBlock FontSize="20" FlowDirection="RightToLeft">
        <Run FlowDirection="LeftToRight">العالم</Run>
        <Run FlowDirection="LeftToRight" Foreground="Red" >فى سلام</Run>
    </TextBlock>

    <TextBlock FontSize="20" FlowDirection="LeftToRight">
        <Run FlowDirection="RightToLeft">العالم</Run>
        <Run FlowDirection="RightToLeft" Foreground="Red" >فى سلام</Run>
    </TextBlock>

    <TextBlock FontSize="20" Foreground="Blue">العالم فى سلام</TextBlock>

    <Separator/>

    <TextBlock FontSize="20" FlowDirection="RightToLeft">
        <Span Foreground="Red" FlowDirection="LeftToRight">Hello</Span>
        <Span FlowDirection="LeftToRight">World</Span>
    </TextBlock>

    <TextBlock FontSize="20" FlowDirection="LeftToRight">
        <Span Foreground="Red" FlowDirection="RightToLeft">Hello</Span>
        <Span FlowDirection="RightToLeft">World</Span>
    </TextBlock>

    <TextBlock FontSize="20" Foreground="Blue">Hello World</TextBlock>

</StackPanel>

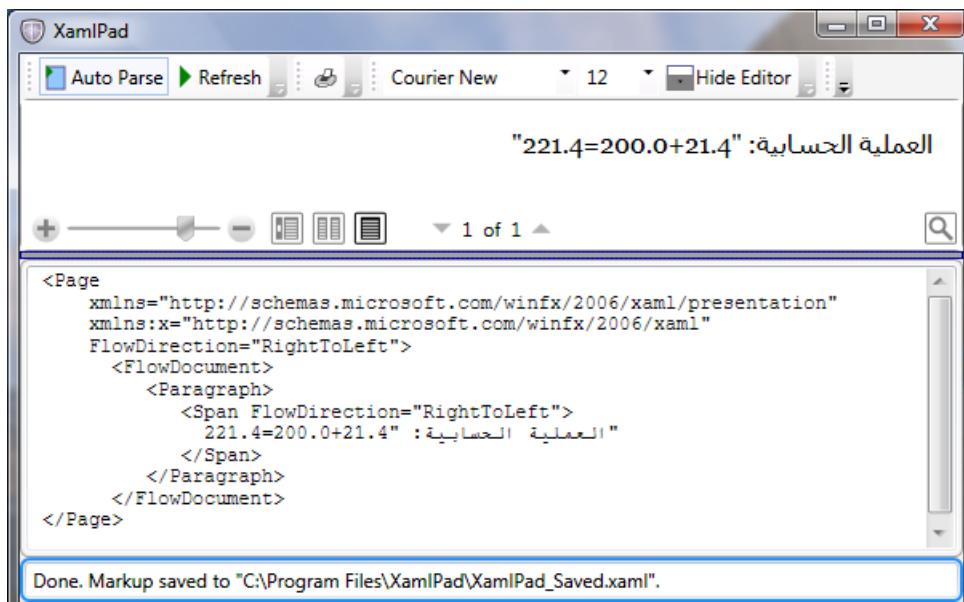
</Page>

```

Negli elementi **TextBlock** dell'esempio, gli elementi **Span** sono disposti in base alla **FlowDirection** degli elementi padre, ma il testo all'interno di ogni **Span** elemento viene trasmesso in base alla propria **FlowDirection**. Questa condizione può essere applicata all'alfabeto latino e arabo oppure a qualsiasi altra lingua.

Aggiunta di **xml:lang**

Nell'immagine seguente viene illustrato un altro esempio in cui vengono utilizzati numeri e espressioni aritmetiche, ad esempio `"200.0+21.4=221.4"`. Si noti che è impostata solo la **FlowDirection**.

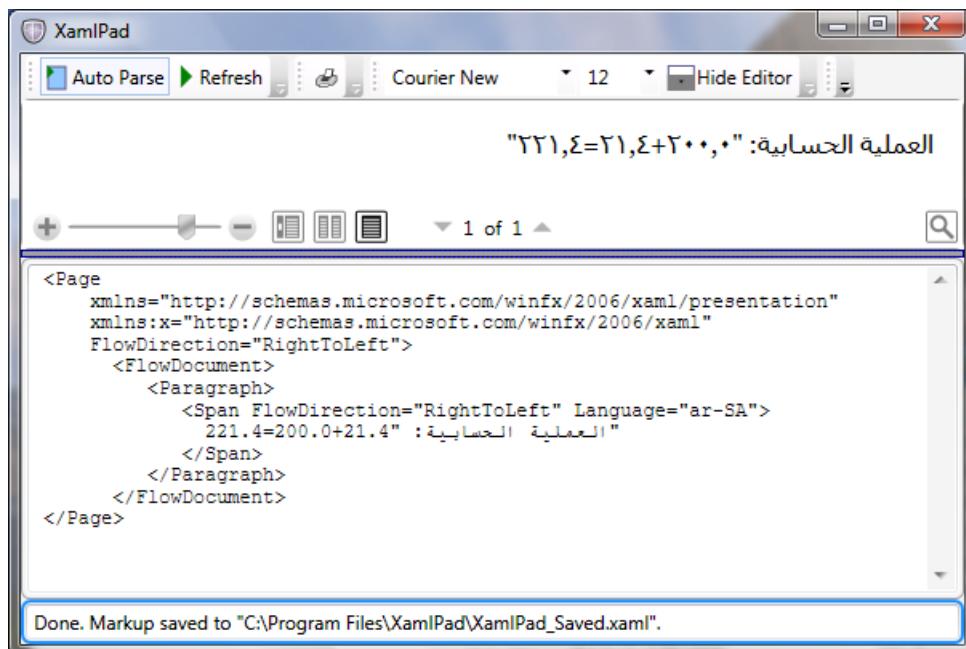


Gli utenti di questa applicazione rimarranno delusi dall'output, anche se la **FlowDirection** è corretta, i numeri non

sono deformati come numeri arabi.

Gli elementi XAML possono includere un attributo XML (`xml:lang`) che definisce la lingua di ogni elemento. XAML supporta inoltre un principio del linguaggio XML in base al quale i valori `xml:lang` applicati agli elementi padre nell'albero vengono utilizzati dagli elementi figlio. Nell'esempio precedente, dal momento che non è stata definita una lingua per l'elemento `Run` o uno degli elementi di primo livello, è stato usato il `xml:lang` predefinito, `en-US` per XAML. L'algoritmo di shaping dei numeri interni di Windows Presentation Foundation (WPF) seleziona i numeri nella lingua corrispondente, in questo caso l'inglese. Per rendere il rendering dei numeri arabi correttamente `xml:lang` necessario impostare.

Il grafico seguente mostra l'esempio con `xml:lang` aggiunto.



Nell'esempio seguente viene aggiunto `xml:lang` all'applicazione.

```
<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    FlowDirection="RightToLeft">
    <FlowDocument>
        <Paragraph>
            <Span FlowDirection="RightToLeft" Language="ar-SA">
                ٢٢١.٤=٢١.٤+٢٠٠.٠
            </Span>
        </Paragraph>
    </FlowDocument>
</Page>
```

Tenere presente che molti linguaggi hanno valori `xml:lang` diversi a seconda dell'area di destinazione, ad esempio `"ar-SA"` e `"ar-EG"` rappresentano due varianti di arabo. Negli esempi precedenti viene illustrato che è necessario definire i valori `xml:lang` e `FlowDirection`.

FlowDirection con elementi non di testo

`FlowDirection` definisce non solo il modo in cui il testo fluisce in un elemento testuale, ma anche la direzione di flusso di quasi tutti gli altri elementi Interfaccia utente. Nell'immagine seguente viene illustrato un `ToolBar` che utilizza una `LinearGradientBrush` orizzontale per creare lo sfondo con una sfumatura da sinistra a destra.

The screenshot shows the XamlPad application interface. At the top, there's a toolbar with buttons for Auto Parse, Refresh, Hide Editor, and font size selection. The main area displays XAML code for a page with a tool bar containing four buttons. The tool bar has a background gradient from dark red to white. Below the XAML, a status bar indicates "Done. Markup saved to 'C:\Program Files\XamlPad\XamlPad_Saved.xaml'".

```
<Page
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >

<ToolBar>
  <ToolBar.Background>
    <LinearGradientBrush StartPoint="0,0.5" EndPoint="1,1">
      <LinearGradientBrush.GradientStops>
        <GradientStop Color="DarkRed" Offset="0" />
        <GradientStop Color="DarkBlue" Offset="0.3" />
        <GradientStop Color="LightBlue" Offset="0.6" />
        <GradientStop Color="White" Offset="1" />
      </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>
  </ToolBar.Background>

  <Button FontSize="12" Foreground="White">Button1</Button>
  <Rectangle Width="20"/>
  <Button FontSize="12" Foreground="White">Button2</Button>
  <Rectangle Width="20"/>
  <Button FontSize="12" Foreground="White">Button3</Button>
  <Rectangle Width="20"/>
  <Button FontSize="12" Foreground="White">Button4</Button>
  <Rectangle Width="20"/>
</ToolBar>

</Page>
```

Done. Markup saved to "C:\Program Files\XamlPad\XamlPad_Saved.xaml".

Dopo aver impostato il [FlowDirection](#) su [RightToLeft](#), non solo i pulsanti [ToolBar](#) vengono disposti da destra a sinistra, ma anche il [LinearGradientBrush](#) riallinea gli offset per il flusso da destra a sinistra.

Il grafico seguente illustra il riallineamento del [LinearGradientBrush](#).

The screenshot shows the XamlPad application interface. At the top, there's a toolbar with buttons for Auto Parse, Refresh, Hide Editor, and font size selection. The main area displays XAML code for a page with a tool bar containing four buttons. The tool bar has a background gradient from white to dark red. The buttons are arranged from right to left. Below the XAML, a status bar indicates "Done. Markup saved to 'C:\Program Files\XamlPad\XamlPad_Saved.xaml'".

```
<Page
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >

<ToolBar FlowDirection="RightToLeft">
  <ToolBar.Background>
    <LinearGradientBrush StartPoint="0,0.5" EndPoint="1,1">
      <LinearGradientBrush.GradientStops>
        <GradientStop Color="DarkRed" Offset="0" />
        <GradientStop Color="DarkBlue" Offset="0.3" />
        <GradientStop Color="LightBlue" Offset="0.6" />
        <GradientStop Color="White" Offset="1" />
      </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>
  </ToolBar.Background>

  <Button FontSize="12" Foreground="White">Button1</Button>
  <Rectangle Width="20"/>
  <Button FontSize="12" Foreground="White">Button2</Button>
  <Rectangle Width="20"/>
  <Button FontSize="12" Foreground="White">Button3</Button>
  <Rectangle Width="20"/>
  <Button FontSize="12" Foreground="White">Button4</Button>
  <Rectangle Width="20"/>
</ToolBar>

</Page>
```

Done. Markup saved to "C:\Program Files\XamlPad\XamlPad_Saved.xaml".

Nell'esempio seguente viene disegnato un [RightToLeftToolBar](#). Per estrarlo da sinistra a destra, rimuovere l'attributo [FlowDirection](#) nella [ToolBar](#).

```

<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <ToolBar FlowDirection="RightToLeft" Height="50" DockPanel.Dock="Top">
        <ToolBar.Background>
            <LinearGradientBrush StartPoint="0,0.5" EndPoint="1,1">
                <LinearGradientBrush.GradientStops>
                    <GradientStop Color="DarkRed" Offset="0" />
                    <GradientStop Color="DarkBlue" Offset="0.3" />
                    <GradientStop Color="LightBlue" Offset="0.6" />
                    <GradientStop Color="White" Offset="1" />
                </LinearGradientBrush.GradientStops>
            </LinearGradientBrush>
        </ToolBar.Background>

        <Button FontSize="12" Foreground="White">Button1</Button>
        <Rectangle Width="20"/>
        <Button FontSize="12" Foreground="White">Button2</Button>
        <Rectangle Width="20"/>
        <Button FontSize="12" Foreground="White">Button3</Button>
        <Rectangle Width="20"/>
        <Button FontSize="12" Foreground="White">Button4</Button>
        <Rectangle Width="20"/>
    </ToolBar>
</Page>

```

Eccezioni di FlowDirection

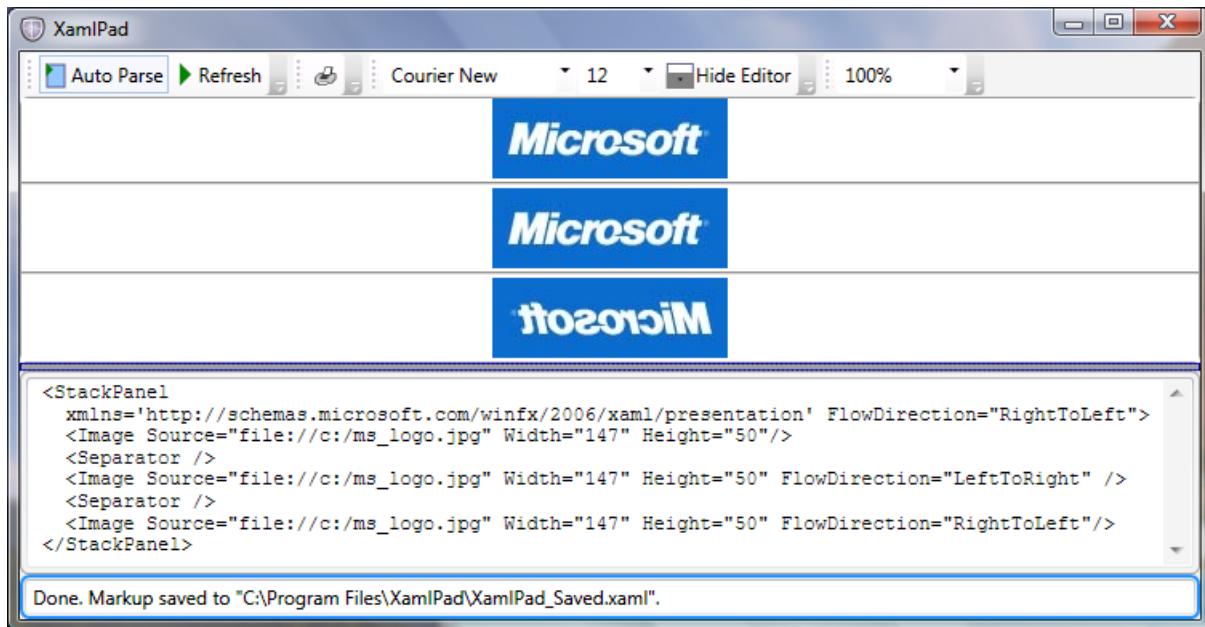
Esistono alcuni casi in cui [FlowDirection](#) non si comporta come previsto. In questa sezione vengono illustrate due di queste eccezioni.

Immagine

Un [Image](#) rappresenta un controllo che visualizza un'immagine. In XAML può essere usato con una proprietà [Source](#) che definisce l'URI (Uniform Resource Identifier) dell'[Image](#) da visualizzare.

A differenza di altri elementi Interfaccia utente, un [Image](#) non eredita il [FlowDirection](#) dal contenitore. Tuttavia, se il [FlowDirection](#) viene impostato in modo esplicito su [RightToLeft](#), viene visualizzato un [Image](#) capovolto orizzontalmente. Questa condizione viene implementata come una funzionalità utile per gli sviluppatori di contenuti bidirezionali, perché in alcuni casi il capovolgimento in senso orizzontale dell'immagine genera l'effetto desiderato.

Il grafico seguente mostra una [Imagecapovolto](#).



Nell'esempio seguente viene illustrato che il [Image](#) non riesce a ereditare il [FlowDirection](#) dal [StackPanel](#) che lo contiene.

NOTE

È necessario disporre di un file denominato **ms_logo.jpg** nella C:\ per eseguire questo esempio.

```
<StackPanel
    xmlns='http://schemas.microsoft.com/winfx/2006/xaml/presentation'
    FlowDirection="RightToLeft">

    <Image Source="file:///c:/ms_logo.jpg"
        Width="147" Height="50"/>
    <Separator Height="10"/>
    <Image Source="file:///c:/ms_logo.jpg"
        Width="147" Height="50" FlowDirection="LeftToRight" />
    <Separator Height="10"/>
    <Image Source="file:///c:/ms_logo.jpg"
        Width="147" Height="50" FlowDirection="RightToLeft"/>
</StackPanel>
```

NOTE

Incluso nei file di download è un file **ms_logo.jpg**. Nel codice si presuppone che il file con estensione jpg non si trovi all'interno del progetto ma in un'altra posizione nell'unità C:. È necessario copiare il file con estensione jpg dai file di progetto nell'unità C:\ oppure modificare il codice affinché la ricerca venga eseguita all'interno del progetto. A tale scopo, modificare

```
Source="file:///c:/ms_logo.jpg" Source="ms_logo.jpg".
```

Percorsi

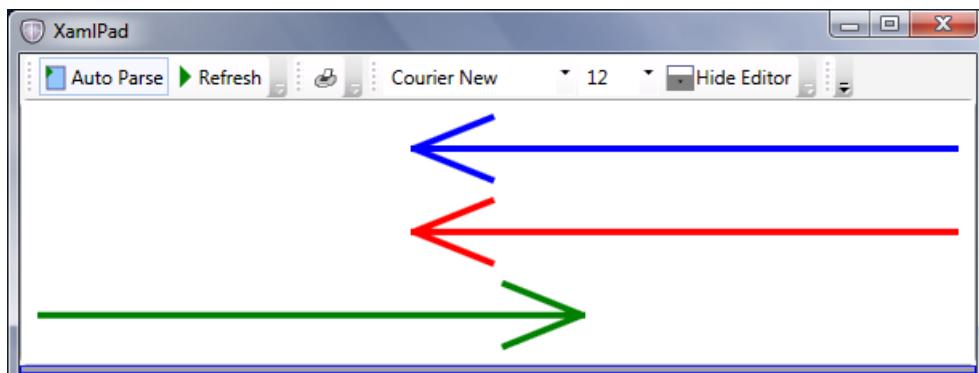
Oltre a una [Image](#), un altro elemento interessante è [Path](#). Path è un oggetto che consente di disegnare una serie di righe e curve collegate. Si comporta in modo simile a un [Image](#) relativo [FlowDirection](#); ad esempio, il [RightToLeftFlowDirection](#) è un mirror orizzontale della relativa [LeftToRight](#) uno. Tuttavia, a differenza di un [Image](#), [Path](#) eredita il [FlowDirection](#) dal contenitore e non è necessario specificarlo in modo esplicito.

L'esempio seguente disegna una freccia semplice con 3 linee. La prima freccia eredita la direzione del flusso [RightToLeft](#) dal [StackPanel](#) in modo che i punti di inizio e di fine vengano misurati dalla radice sul lato destro. La seconda freccia con un [RightToLeft](#) esplicito [FlowDirection](#) inizia anche sul lato destro. Tuttavia, la radice iniziale

della terza freccia è collocata sul lato sinistro. Per ulteriori informazioni sul disegno, vedere [LineGeometry](#) e [GeometryGroup](#).

```
<StackPanel  
    xmlns='http://schemas.microsoft.com/winfx/2006/xaml/presentation'  
    FlowDirection="RightToLeft">  
  
    <Path Stroke="Blue" StrokeThickness="4">  
        <Path.Data>  
            <GeometryGroup >  
                <LineGeometry StartPoint="300,10" EndPoint="350,30" />  
                <LineGeometry StartPoint="10,30" EndPoint="352,30" />  
                <LineGeometry StartPoint="300,50" EndPoint="350,30" />  
            </GeometryGroup>  
        </Path.Data>  
    </Path>  
  
    <Path Stroke="Red" StrokeThickness="4" FlowDirection="RightToLeft">  
        <Path.Data>  
            <GeometryGroup >  
                <LineGeometry StartPoint="300,10" EndPoint="350,30" />  
                <LineGeometry StartPoint="10,30" EndPoint="352,30" />  
                <LineGeometry StartPoint="300,50" EndPoint="350,30" />  
            </GeometryGroup>  
        </Path.Data>  
    </Path>  
  
    <Path Stroke="Green" StrokeThickness="4" FlowDirection="LeftToRight">  
        <Path.Data>  
            <GeometryGroup >  
                <LineGeometry StartPoint="300,10" EndPoint="350,30" />  
                <LineGeometry StartPoint="10,30" EndPoint="352,30" />  
                <LineGeometry StartPoint="300,50" EndPoint="350,30" />  
            </GeometryGroup>  
        </Path.Data>  
    </Path>  
</StackPanel>
```

Il grafico seguente mostra l'output dell'esempio precedente con le frecce disegnate usando l'elemento `Path`:



I [Image](#) e [Path](#) sono due esempi di come Windows Presentation Foundation (WPF) USA [FlowDirection](#). Oltre a definire Interfaccia utente elementi in una direzione specifica all'interno di un contenitore, è possibile utilizzare [FlowDirection](#) con elementi quali [InkPresenter](#) che esegue il rendering di input penna su una superficie, [LinearGradientBrush](#) [RadialGradientBrush](#). Ogni volta che è necessario un comportamento da destra a sinistra per il contenuto che simula un comportamento da sinistra a destra o viceversa, Windows Presentation Foundation (WPF) fornisce tale funzionalità.

Sostituzione numerica

In passato, Windows supportava la sostituzione dei numeri consentendo la rappresentazione di forme culturali

diverse per le stesse cifre mantenendo l'archiviazione interna di queste cifre unificate tra impostazioni locali diverse, ad esempio i numeri vengono archiviati nel relativo valori esadecimali ben noti, 0x40, 0x41, ma visualizzati in base alla lingua selezionata.

In questo modo le applicazioni possono elaborare valori numerici senza la necessità di convertirli da una lingua a un'altra. ad esempio, un utente può aprire un foglio di calcolo di Microsoft Excel in una finestra araba localizzata e visualizzare i numeri con la forma araba, ma aprirlo in un Versione europea di Windows e vedere rappresentazione europea degli stessi numeri. Questa condizione è necessaria anche per altri simboli, quali i separatori virgola e il simbolo della percentuale, perché sono spesso associati a numeri nello stesso documento.

In Windows Presentation Foundation (WPF) è stata mantenuta la stessa tradizione e aggiunto un ulteriore supporto a questa funzionalità che consente un maggiore controllo dell'utente sul momento e sulla modalità d'su della sostituzione. Sebbene questa funzionalità sia stata progettata per tutte le lingue, è particolarmente utile per i contenuti bidirezionali in cui la definizione delle cifre per una lingua specifica rappresenta solitamente una sfida per gli sviluppatori di applicazioni, a causa delle diverse impostazioni cultura con cui un'applicazione può essere eseguita.

La proprietà di base che controlla il funzionamento della sostituzione dei numeri in Windows Presentation Foundation (WPF) è la proprietà di dipendenza **Substitution**. La classe **NumberSubstitution** specifica il modo in cui devono essere visualizzati i numeri nel testo. Include tre proprietà pubbliche che ne definiscono il comportamento. Di seguito è riportato un riepilogo di ognuna delle proprietà:

CultureSource:

Questa proprietà specifica come vengono determinate le impostazioni cultura per i numeri. Accetta uno dei tre valori di enumerazione **NumberCultureSource**.

- **Override**: le impostazioni cultura del numero sono quelle della proprietà **CultureOverride**.
- **Text**: le impostazioni cultura del numero sono quelle della sequenza di testo. Nel markup, questo sarebbe `xml:lang` o la relativa proprietà alias `Language` (`Language` o `Language`). Inoltre, è l'impostazione predefinita per le classi che derivano da **FrameworkContentElement**. Tali classi includono `System.Windows.Documents.Paragraph`, `System.Windows.Documents.Table`, `System.Windows.Documents.TableCell` e così via.
- **User**: le impostazioni cultura del numero sono quelle del thread corrente. Questa proprietà è l'impostazione predefinita per tutte le sottoclassi di **FrameworkElement**, ad esempio `Page`, `Window` e `TextBlock`.

CultureOverride:

La proprietà **CultureOverride** viene utilizzata solo se la proprietà **CultureSource** è impostata su **Override** e in caso contrario viene ignorata. Specifica le impostazioni cultura del numero. Il valore `null`, il valore predefinito, viene interpretato come en-US.

Substitution:

Questa proprietà specifica il tipo di sostituzione numerica da eseguire. Accetta uno dei valori di enumerazione **NumberSubstitutionMethod** seguenti:

- **AsCulture**: il metodo di sostituzione viene determinato in base alla proprietà `NumberFormatInfo.DigitSubstitution` delle impostazioni cultura del numero. Questa è l'impostazione predefinita.
- **Context**: se le impostazioni cultura del numero sono una lingua araba o persiana, specifica che le cifre dipendono dal contesto.
- **European**: i numeri vengono sempre sottoposti a rendering come cifre europee.
- **NativeNational**: il rendering dei numeri viene eseguito usando le cifre nazionali per le impostazioni cultura

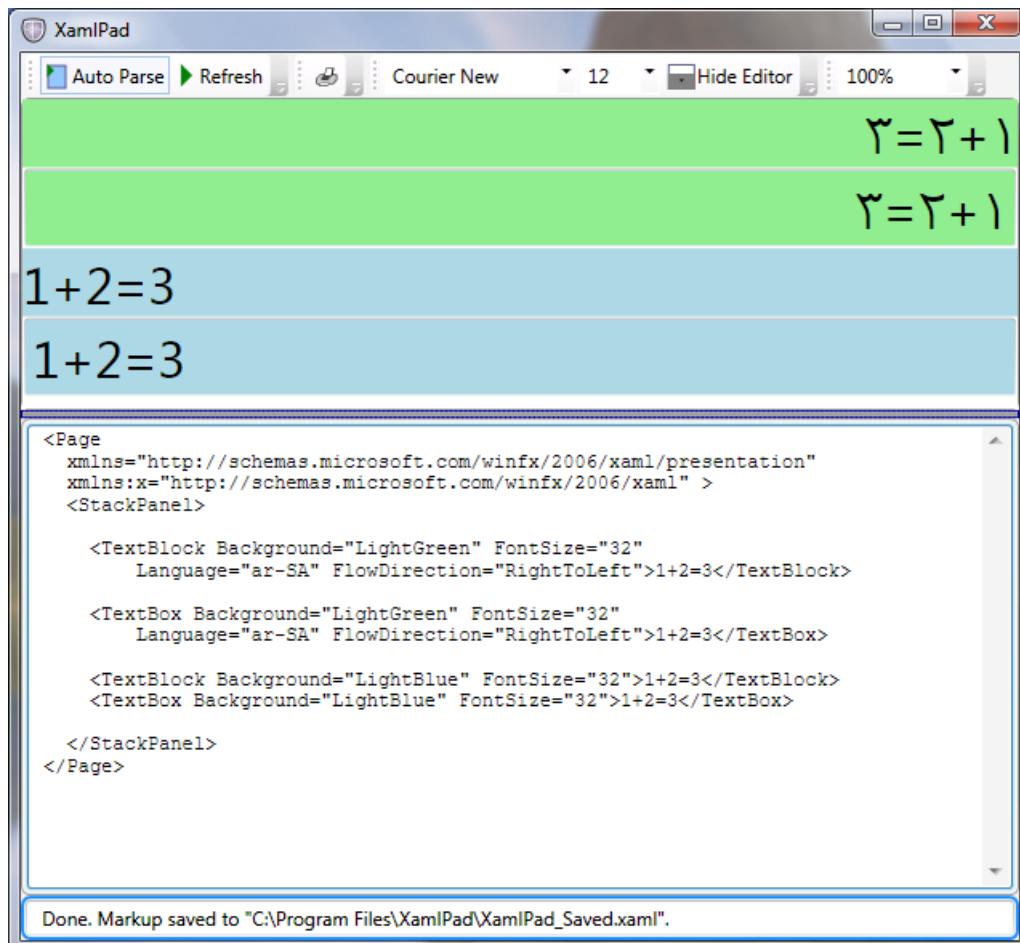
del numero, come specificato dalle [NumberFormat](#)delle impostazioni cultura.

- **Traditional:** il rendering dei numeri viene eseguito usando le cifre tradizionali per le impostazioni cultura del numero. Per la maggior parte delle impostazioni cultura, equivale a [NativeNational](#). Tuttavia, [NativeNational](#) restituisce cifre latine per alcune impostazioni cultura arabe, mentre questo valore genera cifre arabe per tutte le impostazioni cultura arabe.

Cosa significano questi valori per uno sviluppatore di contenuti bidirezionali? Nella maggior parte dei casi, lo sviluppatore potrebbe avere la necessità di definire [FlowDirection](#) e la lingua di ogni elemento di Interfaccia utente testuale, ad esempio `Language="ar-SA"` e la logica di [NumberSubstitution](#) si occupi di visualizzare i numeri in base alla Interfaccia utente corretta. Nell'esempio seguente viene illustrato l'utilizzo di numeri arabi e inglesi in un'applicazione Windows Presentation Foundation (WPF) eseguita in una versione araba di Windows.

```
<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
    <StackPanel>
        <TextBlock Background="LightGreen" FontSize="32"
            Language="ar-SA" FlowDirection="RightToLeft">۱+۲=۳</TextBlock>
        <TextBox Background="LightGreen" FontSize="32"
            Language="ar-SA" FlowDirection="RightToLeft">۱+۲=۳</TextBox>
        <TextBlock Background="LightBlue" FontSize="32">۱+۲=۳</TextBlock>
        <TextBlock Background="LightBlue" FontSize="32">۱+۲=۳</TextBlock>
    </StackPanel>
</Page>
```

Il grafico seguente mostra l'output dell'esempio precedente se si esegue una versione araba di Windows con i numeri arabi e inglesi visualizzati:



Il [FlowDirection](#) era importante in questo caso, perché l'impostazione della [FlowDirection](#) su [LeftToRight](#) avrebbe dovuto restituire cifre europee. Le sezioni seguenti descrivono come ottenere una visualizzazione unificata delle

cifre in tutto il documento. Se l'esempio non viene eseguito nella versione araba di Windows, tutte le cifre vengono visualizzate come cifre europee.

Definizione delle regole di sostituzione

In un'applicazione reale, è necessario impostare l'oggetto Language a livello di codice. Si desidera, ad esempio, impostare l'attributo `xml:lang` in modo che corrisponda a quello utilizzato dal Interfaccia utente del sistema oppure modificare la lingua in base allo stato dell'applicazione.

Se si desidera apportare modifiche in base allo stato dell'applicazione, utilizzare altre funzionalità fornite da Windows Presentation Foundation (WPF).

Per prima cosa, impostare la `NumberSubstitution.CultureSource="Text"` del componente dell'applicazione. L'uso di questa impostazione garantisce che le impostazioni non provengano dal Interfaccia utente per gli elementi di testo con "User" come impostazione predefinita, ad esempio `TextBlock`.

Ad esempio,

```
<TextBlock  
    Name="text1" NumberSubstitution.CultureSource="Text">  
    1234+5679=6913  
</TextBlock>
```

Nel codice corrispondente C#, impostare la proprietà `Language`, ad esempio, su `"ar-SA"`.

```
text1.Language = System.Windows.Markup.XmlLanguage.GetLanguage("ar-SA");
```

Se è necessario impostare la proprietà `Language` sulla lingua dell'interfaccia utente dell'utente corrente, usare il codice seguente.

```
text1.Language =  
System.Windows.Markup.XmlLanguage.GetLanguage(System.Globalization.CultureInfo.CurrentCulture.IetfLanguageTa  
g);
```

`CultureInfo.CurrentCulture` rappresenta le impostazioni cultura correnti utilizzate dal thread corrente in fase di esecuzione.

L'esempio XAML finale dovrebbe essere simile all'esempio seguente.

```
<Page x:Class="WindowsApplication.Window1"  
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
      Title="Code Sample" Height="300" Width="300"  
>  
  <StackPanel>  
    <TextBlock Language="ar-SA"  
              FlowDirection="RightToLeft">3=2+1 : عربي  
    </TextBlock>  
    <TextBlock Language="ar-SA"  
              FlowDirection="RightToLeft"  
              NumberSubstitution.Substitution="European">3=2+1 : عربي :  
    </TextBlock>  
  </StackPanel>  
</Page>
```

L'esempio C# finale dovrebbe essere simile al seguente.

```

namespace BidiTest
{
    public partial class Window1 : Window
    {

        public Window1()
        {
            InitializeComponent();

            string currentLanguage =
                System.Globalization.CultureInfo.CurrentCulture.IetfLanguageTag;

            text1.Language = System.Windows.Markup.XmlLanguage.GetLanguage(currentLanguage);

            if (currentLanguage.ToLower().StartsWith("ar"))
            {
                text1.FlowDirection = FlowDirection.RightToLeft;
            }
            else
            {
                text1.FlowDirection = FlowDirection.LeftToRight;
            }
        }
    }
}

```

Il grafico seguente mostra l'aspetto della finestra per entrambi i linguaggi di programmazione, visualizzando i numeri arabi:



Uso della proprietà Substitution

Il modo in cui la sostituzione dei numeri funziona in Windows Presentation Foundation (WPF) dipende sia dal linguaggio dell'elemento di testo che dal relativo [FlowDirection](#). Se il [FlowDirection](#) è da sinistra a destra, viene eseguito il rendering delle cifre europee. Tuttavia, se è preceduto da testo arabo o la lingua è impostata su "AR" e la [FlowDirection](#) è [RightToLeft](#), viene eseguito il rendering delle cifre arabe.

In alcuni casi, tuttavia, è possibile creare un'applicazione unificata usando, ad esempio, le cifre europee per tutti gli utenti O le cifre arabe in [Table](#) celle con un [Style](#) specifico. Un modo semplice per eseguire questa operazione consiste nell'usare la proprietà [Substitution](#).

Nell'esempio seguente, per la prima [TextBlock](#) non è impostata la proprietà [Substitution](#), quindi l'algoritmo Visualizza le cifre arabe come previsto. Tuttavia, nel secondo [TextBlock](#), la sostituzione è impostata su [European](#) override della sostituzione predefinita per i numeri arabi e vengono visualizzate le cifre europee.

```
<Page x:Class="WindowsApplication.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Code Sample" Height="300" Width="300"
>
    <StackPanel>
        <TextBlock Language="ar-SA"
            FlowDirection="RightToLeft">3=2+1 :عربى
        </TextBlock>
        <TextBlock Language="ar-SA"
            FlowDirection="RightToLeft"
            NumberSubstitution.Substitution="European">3=2+1 :عربى
        </TextBlock>
    </StackPanel>
</Page>
```

Procedure relative alla globalizzazione e alla localizzazione

08/01/2020 • 2 minutes to read • [Edit Online](#)

Negli argomenti di questa sezione viene descritto come sviluppare applicazioni internazionali.

In questa sezione

[Localizzare un'applicazione](#)

[Utilizzare un layout automatico per creare un pulsante](#)

[Usare una griglia per il layout automatico](#)

[Usare un oggetto ResourceDictionary per gestire le risorse di tipo stringa localizzabili](#)

[Usare le risorse in applicazioni localizzabili](#)

Riferimenti

[System.Globalization](#)

[FlowDirection](#)

[NeutralResourcesLanguageAttribute](#)

[Gestione di xml:lang in XAML](#)

Sezioni correlate

Procedura: Localizzare un'applicazione

10/02/2020 • 15 minutes to read • [Edit Online](#)

Questa esercitazione spiega come creare un'applicazione localizzata usando lo strumento LocBaml.

NOTE

Lo strumento LocBaml non è un'applicazione di produzione. Viene presentato come esempio in cui vengono usate alcune delle API di localizzazione e illustra come scrivere uno strumento di localizzazione.

Panoramica

Questo documento offre un approccio graduale alla localizzazione di un'applicazione. Innanzitutto viene preparata l'applicazione in modo che sia possibile estrarre il testo da convertire. Dopo la conversione del testo, il testo convertito verrà incluso in una nuova copia dell'applicazione originale.

Requisiti

Nel corso di questa discussione verrà usato Microsoft Build Engine (MSBuild), che è un compilatore eseguito dalla riga di comando.

Inoltre, verrà spiegato come usare un file di progetto. Per istruzioni sull'uso di MSBuild e dei file di progetto, vedere [compilazione e distribuzione](#).

Tutti gli esempi di questa discussione usano en-US (inglese-Stati Uniti) come impostazioni cultura. In questo modo è possibile eseguire i passaggi degli esempi senza installare un'altra lingua.

Creare un'applicazione di esempio

In questo passaggio viene preparata l'applicazione per la localizzazione. Negli esempi di Windows Presentation Foundation (WPF) viene fornito un esempio HelloApp che verrà usato per gli esempi di codice in questa discussione. Se si vuole usare questo esempio, scaricare i file di Extensible Application Markup Language (XAML) dall'esempio di [strumento LocBaml](#).

1. Sviluppare l'applicazione fino al punto in cui si vuole iniziare la localizzazione.
2. Specificare il linguaggio di sviluppo nel file di progetto in modo che MSBuild generi un assembly principale e un assembly satellite (un file con estensione resources.dll) per contenere le risorse della lingua neutra. Il file di progetto nell'esempio HelloApp è HelloApp.csproj. In questo file la lingua di sviluppo viene identificata come segue:

```
<UICulture>en-US</UICulture>
```

3. Aggiungere gli UID ai file XAML. Gli UID vengono usati per rilevare le modifiche apportate ai file e per identificare gli elementi da convertire. Per aggiungere gli UID ai file, eseguire **updateuid** nel file di progetto:

MSBuild-t:updateuid HelloApp.csproj

Per verificare che non siano presenti UID mancanti o duplicati, eseguire **checkuid**:

MSBuild-t:checkuid HelloApp.csproj

Dopo l'esecuzione di **updateuid**, i file devono contenere gli UID. Ad esempio, il file Pane1.xaml di HelloApp dovrebbe includere quanto segue:

```
<StackPanel x:Uid="StackPanel_1">  
  
    <TextBlock x:Uid="TextBlock_1">Hello World</TextBlock>  
  
    <TextBlock x:Uid="TextBlock_2">Goodbye World</TextBlock>  
  
</StackPanel>
```

Creare l'assembly satellite per le risorse della lingua di sistema

Dopo aver configurato l'applicazione per generare un assembly satellite per le risorse della lingua di sistema, è necessario compilare l'applicazione. Viene generato l'assembly principale dell'applicazione, nonché l'assembly satellite per le risorse della lingua di sistema richiesto da LocBaml per la localizzazione. Per compilare l'applicazione:

1. Compilare HelloApp per creare una libreria di collegamento dinamico (DLL):

msbuild helloapp.csproj

2. L'assembly principale dell'applicazione appena creato, HelloApp.exe, viene inserito nella cartella seguente:

```
C:\HelloApp\Bin\Debug\
```

3. L'assembly satellite per le risorse della lingua di sistema appena creato, HelloApp.resources.dll, viene inserito nella cartella seguente:

```
C:\HelloApp\Bin\Debug\en-US\
```

Compilare lo strumento LocBaml

1. Tutti i file necessari per compilare LocBaml si trovano negli esempi in WPF. Scaricare i C# file dall'[esempio dello strumento LocBaml](#).

2. Eseguire il file di progetto (LocBaml.csproj) per compilare lo strumento dalla riga di comando:

msbuild locbaml.csproj

3. Passare alla directory Bin\Release per trovare il file eseguibile appena creato (locbaml.exe). Esempio:
C:\LocBaml\Bin\Release\locbaml.exe

4. Le opzioni che è possibile specificare quando si esegue LocBaml sono le seguenti:

- **Parse o -p:** analizza BAML, le risorse o i file dll per generare un file con estensione CSV o txt.
- **generate o -g:** genera un file binario localizzato usando un file convertito.
- **out o -o {DirectoryName} :** nome del file di output.
- **culture o -cul {culture} :** impostazioni locali degli assembly di output.
- **Translation o -Trans {Translation.csv} :** file convertito o localizzato.
- **asmPath o -asmpath: {DirectoryName} :** se il codice XAML contiene controlli personalizzati, è necessario fornire **asmPath** all'assembly del controllo personalizzato.
- **nologo:** non visualizza loghi o informazioni sul copyright.
- **verbose:** visualizza le informazioni sulla modalità dettagliata.

NOTE

Se è necessario un elenco delle opzioni quando si esegue lo strumento, digitare **LocBaml.exe** e premere INVIO.

Usare LocBaml per analizzare un file

Ora che è stato creato lo strumento LocBaml, è possibile usarlo per analizzare HelloApp.resources.dll ed estrarre il contenuto testuale che verrà localizzato.

1. Copiare LocBaml.exe nella cartella bin\debug dell'applicazione in cui è stato creato l'assembly principale dell'applicazione.
2. Per analizzare il file dell'assembly satellite e archiviare l'output come file CSV, usare il comando seguente:

LocBaml.exe /parse HelloApp.resources.dll /out:Hello.csv

NOTE

Se il file di input, HelloApp.resources.dll, non è nella stessa directory di LocBaml.exe, spostare uno dei file in modo che entrambi siano nella stessa directory.

3. Quando si esegue LocBaml per analizzare i file, l'output è costituito da sette campi delimitati da virgole (file CSV) o da tabulazioni (file TXT). Di seguito viene visualizzato il file CSV analizzato per HelloApp.resources.dll:

HelloApp.g.en-US.resources:window1.baml,Stack1:System.Windows.Controls.StackPanel.\$Content,Ignore,TRUE,,#Text1;#Text2;

HelloApp.g.en-US.resources:window1.baml,Text1:System.Windows.Controls.TextBlock.\$Content,None,TRUE, TRUE,,Hello World
--

HelloApp.g.en-US.resources:window1.baml,Text2:System.Windows.Controls.TextBlock.\$Content,None,TRUE, TRUE,,Goodbye World
--

I campi di sette sono:

- a. **Nome BAML**. Il nome della risorsa BAML rispetto all'assembly satellite per la lingua di origine.
- b. **Chiave di risorsa**. L'identificatore della risorsa localizzata.
- c. **Category**. Tipo di valore. Vedere [attributi e commenti di localizzazione](#).
- d. **Readability**. Se il valore può essere letto da un localizzatore. Vedere [attributi e commenti di localizzazione](#).
- e. **Modifiability**. Se il valore può essere modificato da un localizzatore. Vedere [attributi e commenti di localizzazione](#).
- f. **Commenti**. Descrizione aggiuntiva del valore per determinarne la modalità di localizzazione. Vedere [attributi e commenti di localizzazione](#).
- g. **Valore**. Il valore di testo da convertire nelle impostazioni cultura desiderate.

La tabella seguente mostra come viene eseguito il mapping di questi campi ai valori delimitati del file CSV:

NOME BAML	CHIAVE DI RISORSA	CATEGORY	LEGGIBILITÀ	MODIFICABILITÀ	COMMENTI	VALORE
HelloApp.g.en-US.resources:window1.xaml	Stack1:System.Windows.Controls.StackPanel.Content	Ignora	FALSE	FALSE		#Text1;#Text2
HelloApp.g.en-US.resources:window1.xaml	Text1:System.Windows.Controls.TextBlock.Content	nessuno	TRUE	TRUE		Hello World
HelloApp.g.en-US.resources:window1.xaml	Text2:System.Windows.Controls.TextBlock.Content	nessuno	TRUE	TRUE		Goodbye World

Si noti che tutti i valori per il campo **Commenti** non contengono valori. Se un campo non ha un valore, è vuoto. Si noti inoltre che l'elemento nella prima riga non è leggibile né modificabile e ha "Ignora" come valore di **categoria**, il che indica che il valore non è localizzabile.

- Per facilitare l'individuazione degli elementi localizzabili nei file analizzati, in particolare nei file di grandi dimensioni, è possibile ordinare o filtrare gli elementi per **categoria**, **leggibilità** e **modificabilità**. Ad esempio, è possibile escludere i valori non leggibili e non modificabili.

Convertire il contenuto localizzabile

Usare gli strumenti disponibili per convertire il contenuto estratto. Per eseguire questa operazione, è possibile scrivere le risorse in un file con estensione CSV e visualizzarle in Microsoft Excel, in modo da apportare modifiche alla conversione nell'ultima colonna (valore).

Usare LocBaml per generare un nuovo file resources.dll

Il contenuto identificato analizzando HelloApp.resources.dll con LocBaml è stato convertito e deve essere reinserito nell'applicazione originale. Usare l'opzione **generate** o **-g** per generare un nuovo file resources.dll.

- Usare la sintassi seguente per generare un nuovo file HelloApp.resources.dll. Contrassegnare le impostazioni cultura come en-US (/cul:en-US).

```
LocBaml.exe /generate HelloApp.resources.dll /trans:Hello.csv /out:c:\ /cul:en-US
```

NOTE

Se il file di input Hello.csv non è presente nella stessa directory del file eseguibile LocBaml.exe, spostare uno dei file in modo che entrambi siano nella stessa directory.

- Sostituire il file HelloApp.resources.dll precedente nella directory C:\HelloApp\Bin\Debug\en-US\HelloApp.resources.dll con il file HelloApp.resources.dll appena creato.
- Ora "Hello World" e "Goodbye World" possono essere convertiti nell'applicazione.
- Per eseguire la conversione in una lingua diversa, usare le impostazioni cultura della lingua in cui si sta eseguendo la conversione. L'esempio seguente mostra come eseguire la conversione in lingua francese

canadese:

LocBaml.exe/generate HelloApp.resources.dll/Trans:Hellofr-CA.csv/out:c:/CUL:fr-CA

5. Nello stesso assembly dell'assembly principale dell'applicazione, creare una nuova cartella specifica per le impostazioni cultura dove ospitare il nuovo assembly satellite. Per la lingua francese canadese, la cartella sarà fr-CA.
6. Copiare l'assembly satellite generato nella nuova cartella.
7. Per testare il nuovo assembly satellite, è necessario modificare le impostazioni cultura con cui verrà eseguita l'applicazione. Questa operazione può essere eseguita in due modi:
 - Modificare le impostazioni internazionali del sistema operativo (**avviare | il Pannello | di controllo Opzioni internazionali e della lingua**).
 - Aggiungere il codice seguente al file App.xaml.cs nell'applicazione:

```
<Application
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SDKSample.App"
    x:Uid="Application_1"
    StartupUri="Window1.xaml">
</Application>
```

```
using System.Windows;
using System.Globalization;
using System.Threading;

namespace SDKSample
{
    public partial class App : Application
    {
        public App()
        {
            // Change culture under which this application runs
            CultureInfo ci = new CultureInfo("fr-CA");
            Thread.CurrentThread.CurrentCulture = ci;
            Thread.CurrentThread.CurrentUICulture = ci;
        }
    }
}
```

```
Imports System.Windows
Imports System.Globalization
Imports System.Threading

Namespace SDKSample
    Partial Public Class App
        Inherits Application
        Public Sub New()
            ' Change culture under which this application runs
            Dim ci As New CultureInfo("fr-CA")
            Thread.CurrentThread.CurrentCulture = ci
            Thread.CurrentThread.CurrentUICulture = ci
        End Sub
    End Class
End Namespace
```

Alcuni suggerimenti per l'uso di LocBaml

- Tutti gli assembly dipendenti che definiscono i controlli personalizzati devono essere copiati nella directory locale di LocBaml o installati in Global Assembly Cache. Questa operazione è necessaria perché l'API di localizzazione deve avere accesso agli assembly dipendenti quando legge il codice XAML binario (BAML).
- Se l'assembly principale è firmato, anche la DLL di risorse generata deve essere firmata per poter essere caricata.
- La versione della DLL di risorsa localizzata deve essere sincronizzata con l'assembly principale.

Passaggi successivi

A questo punto dovrebbero essere state acquisite le conoscenze di base sull'uso dello strumento LocBaml. Si dovrebbe essere in grado di creare un file che contiene UID. Usando lo strumento LocBaml, si dovrebbe essere in grado di analizzare un file per estrarre il contenuto localizzabile e, dopo la conversione del contenuto, generare un file resources.dll che inserisce il contenuto convertito. Questo argomento non include tutti i dettagli, ma offre le informazioni necessarie per usare LocBaml per la localizzazione delle applicazioni.

Vedere anche

- [Globalizzazione per WPF](#)
- [Cenni preliminari sull'utilizzo del layout automatico](#)

Procedura: Usare il layout automatico per creare un pulsante

23/10/2019 • 2 minutes to read • [Edit Online](#)

Questo esempio descrive come usare un approccio basato sul layout automatico per la creazione di un pulsante in un'applicazione localizzabile.

Localizzazione di un interfaccia utente può essere un processo molto tempo. Spesso sono necessari il ridimensionamento e il riposizionamento degli elementi, oltre alla traduzione del testo. In precedenza ogni lingua che un Interfaccia utente richiedeva delle modifiche è stato adattato. Ora con le funzionalità di Windows Presentation Foundation (WPF) è possibile progettare elementi che riducono la necessità di modifiche. L'approccio alla scrittura di applicazioni che è possibile ridimensionare e riposizionare con maggiore semplicità viene definito `automatic layout`.

Esempio

I seguenti due Extensible Application Markup Language (XAML) esempi di creano applicazioni che creano un pulsante, uno con testo in lingua inglese e uno con testo in spagnolo. Il codice è lo stesso ad eccezione del testo. Il pulsante si regola per adattarsi al testo.

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="ButtonLoc.Pane1"
    Name="myWindow"
    SizeToContent="WidthAndHeight"
    >

<DockPanel>
    <Button FontSize="28" Height="50">My name is Hope.</Button>
</DockPanel>
</Window>
```

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="ButtonLoc.Pane1"
    Name="myWindow"
    SizeToContent="WidthAndHeight"
    >

<DockPanel>
    <Button FontSize="28" Height="50">Me llamo Esperanza.</Button>
</DockPanel>
</Window>
```

La figura seguente mostra l'output degli esempi di codice con i pulsanti ridimensionabile automatico:



Vedere anche

- [Cenni preliminari sull'utilizzo del layout automatico](#)
- [Usare una griglia per il layout automatico](#)

Procedura: Usare una griglia per il layout automatico

23/10/2019 • 2 minutes to read • [Edit Online](#)

Questo esempio descrive come usare una griglia nell'approccio basato sul layout automatico per la creazione di un'applicazione localizzabile.

Localizzazione di un interfaccia utente può essere un processo molto tempo. Spesso sono necessari il ridimensionamento e il riposizionamento degli elementi, oltre alla traduzione del testo. In precedenza ogni lingua che un Interfaccia utente richiedeva delle modifiche è stato adattato. Ora con le funzionalità di Windows Presentation Foundation (WPF) è possibile progettare elementi che riducono la necessità di modifiche. Viene chiamato l'approccio alla scrittura di applicazioni che possono risultare più semplice ridimensionare e riposizionare `auto layout`.

Nell'esempio Extensible Application Markup Language (XAML) viene illustrato l'utilizzo di una griglia per posizionare alcuni pulsanti e del testo. Si noti che l'altezza e la larghezza delle celle sono impostate per `Auto`; pertanto la cella che contiene il pulsante con un'immagine viene modificata per adattarsi all'immagine. Poiché il `Grid` elemento regolato in base al relativo contenuto può essere utile quando si acquisisce l'approccio di layout automatico per la progettazione di applicazioni che possono essere localizzate.

Esempio

Nell'esempio riportato di seguito viene illustrato come usare una griglia.

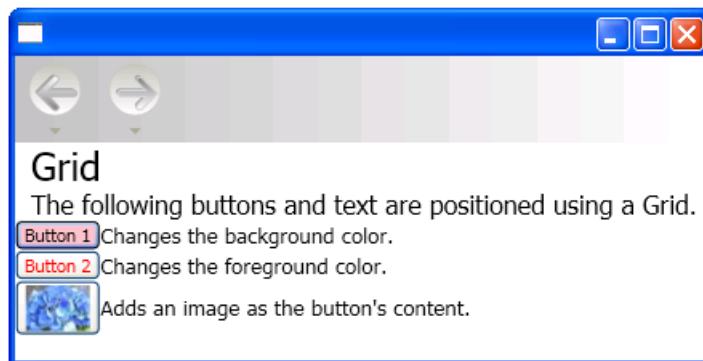
```

<Grid Name="grid" ShowGridLines ="false">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>

<TextBlock Margin="10, 10, 5, 5" Grid.Column="0" Grid.Row="0" FontSize="24">Grid
</TextBlock>
<TextBlock Margin="10, 10, 5, 5" Grid.Column="0" Grid.Row="1" FontSize="12"
    Grid.ColumnSpan="2">The following buttons and text are positioned using a Grid.
</TextBlock>
<Button Margin="10, 10, 5, 5" Grid.Column="0" Grid.Row="2" Background="Pink"
    BorderBrush="Black" BorderThickness="10">Button 1
</Button>
<TextBlock Margin="10, 10, 5, 5" Grid.Column="1" Grid.Row="2" FontSize="12"
    VerticalAlignment="Center" TextWrapping="WrapWithOverflow">Sets the background
    color.
</TextBlock>
<Button Margin="10, 10, 5, 5" Grid.Column="0" Grid.Row="3" Foreground="Red">
    Button 2
</Button>
<TextBlock Margin="10, 10, 5, 5" Grid.Column="1" Grid.Row="3" FontSize="12"
    VerticalAlignment="Center" TextWrapping="WrapWithOverflow">Sets the foreground
    color.
</TextBlock>
<Button Margin="10, 10, 5, 5" Grid.Column="0" Grid.Row="4">
    <Image Source="data\flower.jpg"/>
</Button>
<TextBlock Margin="10, 10, 5, 5" Grid.Column="1" Grid.Row="4" FontSize="12"
    VerticalAlignment="Center" TextWrapping="WrapWithOverflow">Adds an image as
    the button's content.
</TextBlock>
</Grid>

```

La figura seguente mostra l'output dell'esempio di codice.



Grid

Vedere anche

- [Cenni preliminari sull'utilizzo del layout automatico](#)
- [Utilizzare un layout automatico per creare un pulsante](#)

Procedura: Usare un oggetto ResourceDictionary per gestire le risorse di tipo stringa localizzabili

23/10/2019 • 2 minutes to read • [Edit Online](#)

In questo esempio viene illustrato come utilizzare un [ResourceDictionary](#) per assemblare risorse stringa localizzabili per applicazioni Windows Presentation Foundation (WPF).

Per usare un oggetto ResourceDictionary per gestire le risorse di tipo stringa localizzabili

1. Creare un [ResourceDictionary](#) che contiene le stringhe da localizzare. Il codice seguente illustra un esempio.

```
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:system="clr-namespace:System;assembly=mscorlib">

    <!-- String resource that can be localized -->
    <system:String x:Key="localizedMessage">en-US Message</system:String>

</ResourceDictionary>
```

Questo codice definisce una risorsa, stringa `localizedMessage`, di tipo [String](#), dal [System](#) dello spazio dei nomi in mscorelib.dll.

2. Aggiungere il [ResourceDictionary](#) all'applicazione, usando il codice seguente.

```
<Application.Resources>
    <ResourceDictionary>
        <ResourceDictionary.MergedDictionaries>
            <ResourceDictionary Source="StringResources.xaml" />
        </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
</Application.Resources>
```

3. Usare la risorsa stringa dal markup, usando Extensible Application Markup Language (XAML) analogo al seguente.

```
<!-- Declarative use of string resource from StringResources.xaml resource dictionary -->
<TextBox DockPanel.Dock="Top" Text="{StaticResource localizedMessage}" />
```

4. Usare la risorsa di tipo stringa dal code-behind, usando un codice analogo al seguente.

```
// Programmatic use of string resource from StringResources.xaml resource dictionary
string localizedMessage = (string)Application.Current.FindResource("localizedMessage");
MessageBox.Show(localizedMessage);
```

```
' Programmatic use of string resource from StringResources.xaml resource dictionary
Dim localizedMessage As String = CStr(Application.Current.FindResource("localizedMessage"))
MessageBox.Show(localizedMessage)
```

5. Localizzare l'applicazione. Per altre informazioni, vedere [localizzare un'applicazione](#).

Procedura: Usare risorse in applicazioni localizzabili

23/10/2019 • 2 minutes to read • [Edit Online](#)

Localizzare significa adattare un' Interfaccia utente a impostazioni cultura diverse. A questo scopo, testo quale titoli, didascalie, elementi di caselle di riepilogo e così via deve essere tradotto. Per facilitare la traduzione, gli elementi localizzabili vengono raccolti in file di risorse. Visualizzare **localizzare un'applicazione** per informazioni su come creare un file di risorse per la localizzazione. Per rendere un WPF applicazioni localizzabili, gli sviluppatori devono compilare tutte le risorse localizzabili in un assembly di risorse. L'assembly di risorse viene localizzato in lingue diverse e il code-behind Usa l'API Gestione risorse da caricare. Uno dei file richiesti per un WPF applicazione è un file di progetto (proj). Tutte le risorse usate nell'applicazione devono essere incluse nel file di progetto.

Nell'esempio di codice seguente viene illustrata questa operazione.

Esempio

XAML

```
<Resource Include="data\picture1.jpg"/>  
<EmbeddedResource Include="data\stringtable.en-US.restext"/>
```

Per usare una risorsa nell'applicazione, si crea un'istanza **ResourceManager** e caricare la risorsa da usare. L'esempio di codice seguente illustra come eseguire questa operazione.

```
void OnClick(object sender, RoutedEventArgs e)  
{  
    ResourceManager rm = new ResourceManager ("MySampleApp.data.stringtable",  
        Assembly.GetExecutingAssembly());  
    Text1.Text = rm.GetString("Message");  
}
```

Layout

23/10/2019 • 18 minutes to read • [Edit Online](#)

In questo argomento viene descritto il sistema di layout Windows Presentation Foundation (WPF). Comprendere come e quando si verificano i calcoli di layout è essenziale per la creazione di interfacce utente in WPF.

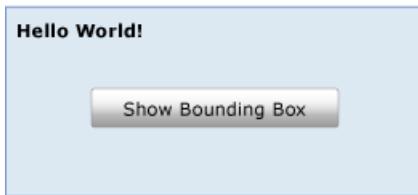
Di seguito sono elencate le diverse sezioni di questo argomento:

- [Rettangoli di selezione degli elementi](#)
- [Sistema di layout](#)
- [Misurazione e disposizione degli elementi figlio](#)
- [Elementi Panel e comportamenti di layout personalizzati](#)
- [Considerazioni sulle prestazioni del layout](#)
- [Rendering dei sub-pixel e arrotondamento del layout](#)
- [Argomenti successivi](#)

Rettangoli di selezione degli elementi

Quando si pensa al layout in WPF, è importante comprendere il rettangolo di delimitazione che racchiude tutti gli elementi. Ogni [FrameworkElement](#) oggetto utilizzato dal sistema di layout può essere considerato come un rettangolo con slot nel layout. La [LayoutInformation](#) classe restituisce i limiti dell'allocazione del layout di un elemento o dello slot. Le dimensioni del rettangolo vengono determinate calcolando lo spazio disponibile sullo schermo, la dimensione di tutti i vincoli, le proprietà specifiche del layout (ad esempio Margin e Padding) e il singolo comportamento dell'elemento [Panel](#) padre. Elaborando questi dati, il sistema di layout è in grado di calcolare la posizione di tutti gli elementi [Panelfiglio](#) di un particolare. È importante ricordare che le caratteristiche di ridimensionamento definite nell'elemento padre, ad esempio [Border](#), influiscono sui relativi elementi figlio.

La figura seguente mostra un layout semplice.



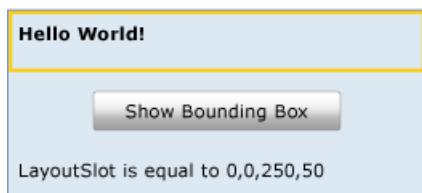
È possibile ottenere questo layout usando il codice XAML seguente.

```

<Grid Name="myGrid" Background="LightSteelBlue" Height="150">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="250"/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
        <RowDefinition />
    </Grid.RowDefinitions>
    <TextBlock Name="txt1" Margin="5" FontSize="16" FontFamily="Verdana" Grid.Column="0" Grid.Row="0">Hello
    World!</TextBlock>
    <Button Click="getLayoutSlot1" Width="125" Height="25" Grid.Column="0" Grid.Row="1">Show Bounding
    Box</Button>
    <TextBlock Name="txt2" Grid.Column="1" Grid.Row="2"/>
</Grid>

```

Un singolo [TextBlock](#) elemento è ospitato all'interno [Grid](#) di un oggetto. Mentre il testo riempie solo l'angolo superiore sinistro della prima colonna, lo spazio allocato per l'oggetto [TextBlock](#) è effettivamente molto più grande. Il rettangolo di delimitazione di [FrameworkElement](#) any può essere recuperato tramite il [GetLayoutSlot](#) metodo. Nell'illustrazione seguente viene mostrato il rettangolo di delimitazione [TextBlock](#) per l'elemento.



Come illustrato dal rettangolo giallo, lo spazio allocato per l' [TextBlock](#) elemento è in realtà molto più grande di quanto appaia. Con l'aggiunta di elementi aggiuntivi all' [Grid](#) oggetto, l'allocazione può ridursi o espandersi, a seconda del tipo e delle dimensioni degli elementi aggiunti.

Lo slot di layout dell' [TextBlock](#) oggetto viene convertito in [Path](#) un oggetto utilizzando [GetLayoutSlot](#) il metodo. Questa tecnica può essere utile per la visualizzazione del rettangolo di selezione di un elemento.

```

private void getLayoutSlot1(object sender, System.Windows.RoutedEventArgs e)
{
    RectangleGeometry myRectangleGeometry = new RectangleGeometry();
    myRectangleGeometry.Rect = LayoutInformation.GetLayoutSlot(txt1);
    Path myPath = new Path();
    myPath.Data = myRectangleGeometry;
    myPath.Stroke = Brushes.LightGoldenrodYellow;
    myPath.StrokeThickness = 5;
    Grid.SetColumn(myPath, 0);
    Grid.SetRow(myPath, 0);
    myGrid.Children.Add(myPath);
    txt2.Text = "LayoutSlot is equal to " + LayoutInformation.GetLayoutSlot(txt1).ToString();
}

```

```

Private Sub getLayoutSlot1(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Dim myRectangleGeometry As New RectangleGeometry
    myRectangleGeometry.Rect = LayoutInformation.GetLayoutSlot(txt1)
    Dim myPath As New Path
    myPath.Data = myRectangleGeometry
    myPath.Stroke = Brushes.LightGoldenrodYellow
    myPath.StrokeThickness = 5
    Grid.SetColumn(myPath, 0)
    Grid.SetRow(myPath, 0)
    myGrid.Children.Add(myPath)
    txt2.Text = "LayoutSlot is equal to " + LayoutInformation.GetLayoutSlot(txt1).ToString()
End Sub

```

Sistema di layout

Nella forma più semplice, il layout è un sistema ricorsivo che fa sì che un elemento venga ridimensionato, posizionato e disegnato. In particolare, il layout descrive il processo di misurazione e disposizione dei membri della [Panel Children](#) raccolta di un elemento. Il layout è un processo intensivo. Più grande [Children](#) è la raccolta, maggiore è il numero di calcoli che devono essere effettuati. La complessità può anche essere introdotta in base al comportamento di layout [Panel](#) definito dall'elemento proprietario della raccolta. Un oggetto relativamente [Panel](#) semplice, [Canvas](#) ad esempio, può avere prestazioni significativamente migliori rispetto a un [Panel Grid](#) più complesso, ad esempio.

Ogni volta che un elemento [UIElement](#) figlio modifica la posizione, è possibile attivare un nuovo passaggio dal sistema di layout. Di conseguenza, è importante identificare gli eventi che possono richiamare il sistema di layout, perché una chiamata non necessaria può causare scarse prestazioni dell'applicazione. Di seguito viene descritto il processo avviato quando viene richiamato il sistema di layout.

1. Un figlio [UIElement](#) inizia il processo di layout con le relative proprietà principali misurate.
2. Le proprietà di [FrameworkElement](#) ridimensionamento definite in vengono valutate, [Width](#), [Height](#) e [Margin](#), ad esempio, [Height](#).
3. [Panel](#) viene applicata la logica specifica, ad esempio [Dock](#) direzione o [Orientation](#) stack.
4. Il contenuto viene disposto dopo la misurazione di tutti gli elementi figlio.
5. La [Children](#) raccolta viene disegnata sullo schermo.
6. Il processo viene richiamato di nuovo [Children](#) se vengono aggiunti ulteriori alla raccolta, [LayoutTransform](#) viene applicato un oggetto o [UpdateLayout](#) viene chiamato il metodo.

Questo processo e il modo in cui viene richiamato vengono descritti con maggiori dettagli nelle sezioni seguenti.

Misurazione e disposizione degli elementi figlio

Il sistema di layout completa due passaggi per ogni membro della [Children](#) raccolta, un passaggio di misurazione e un passaggio di disposizione. Ogni elemento [Panel](#) figlio fornisce i [MeasureOverride](#) propri [ArrangeOverride](#) metodi e per ottenere il proprio comportamento di layout specifico.

Durante il passaggio di misurazione, viene valutato ogni [Children](#) membro della raccolta. Il processo inizia con una chiamata al [Measure](#) metodo. Questo metodo viene chiamato all'interno dell'implementazione dell'elemento [Panel](#) padre e non deve essere chiamato in modo esplicito affinché il layout venga eseguito.

In [Visibility](#) [UIElement](#) primoluogo, vengono valutate le proprietà dimensionate di, ad esempio, [Clip](#). Viene generato un valore denominato [constraintSize](#) che viene passato a [MeasureCore](#).

In secondo luogo, le proprietà del [FrameworkElement](#) Framework definite in vengono elaborate, che

`constraintSize` influiscono sul valore di. Queste proprietà in genere descrivono le caratteristiche di ridimensionamento dell'oggetto sottostante [UIElement](#), [Width](#) e [Margin](#) ad esempio [Height](#), [Style](#), e. Ognuna di queste proprietà può modificare lo spazio necessario per visualizzare l'elemento. [MeasureOverride](#) viene quindi chiamato con `constraintSize` come parametro.

NOTE

Esiste una differenza tra le [Height](#) proprietà di [ActualHeight](#) e [Width](#) [ActualWidth](#) e. Ad esempio, la [ActualHeight](#) proprietà è un valore calcolato in base ad altri input di altezza e al sistema di layout. Il valore viene impostato dal sistema di layout stesso, in base a un passaggio di rendering effettivo, e può quindi essere leggermente indietro rispetto al valore impostato delle proprietà [Height](#), ad esempio, che costituiscono la base della modifica di input.

Poiché [ActualHeight](#) è un valore calcolato, è necessario tenere presente che potrebbero essere presenti più modifiche segnalate in modo incrementale o incrementale come risultato di varie operazioni da parte del sistema di layout. Il sistema di layout può calcolare lo spazio di misurazione necessario per gli elementi figlio, i vincoli dell'elemento padre e così via.

L'obiettivo finale del passaggio di misurazione è che l'elemento figlio determini il [DesiredSize](#) relativo oggetto, che si [MeasureCore](#) verifica durante la chiamata. Il [DesiredSize](#) valore viene archiviato da [Measure](#) per l'utilizzo durante il passaggio di disposizione del contenuto.

Il passaggio di disposizione inizia con una chiamata al [Arrange](#) metodo. Durante il passaggio di disposizione, l'[Panel](#) elemento padre genera un rettangolo che rappresenta i limiti del figlio. Questo valore viene passato al metodo [ArrangeCore](#) per l'elaborazione.

Il [ArrangeCore](#) metodo valuta l'oggetto [DesiredSize](#) del figlio e valuta eventuali margini aggiuntivi che potrebbero influire sulle dimensioni di rendering dell'elemento. [ArrangeCore](#) genera un `arrangesize` oggetto, che viene passato [ArrangeOverride](#) al metodo di [Panel](#) come parametro. [ArrangeOverride](#) genera l' `finalsize` oggetto dell'oggetto figlio. Infine, il [ArrangeCore](#) metodo esegue una valutazione finale delle proprietà di offset, ad esempio il margine e l'allineamento, e inserisce l'elemento figlio all'interno dello slot di layout. L'elemento figlio non deve necessariamente (e in genere non lo fa) occupare l'intero spazio allocato. Il controllo viene quindi restituito al padre [Panel](#) e il processo di layout è completo.

Elementi Panel e comportamenti di layout personalizzati

WPF include un gruppo di elementi che derivano da [Panel](#). Questi [Panel](#) elementi abilitano molti layout complessi. Ad esempio, è possibile ottenere facilmente gli elementi di stack usando l' [StackPanel](#) elemento, mentre i layout più complessi e di flusso libero sono possibili usando un [Canvas](#).

Nella tabella seguente sono riepilogati gli elementi [Panel](#) di layout disponibili.

NOME ELEMENTO PANEL	DESCRIZIONE
Canvas	Definisce un'area all'interno della Canvas quale è possibile posizionare in modo esplicito gli elementi figlio in base alle coordinate relative all'area.
DockPanel	Definisce un'area all'interno della quale è possibile disporre elementi figlio in orizzontale o in verticale, l'uno rispetto all'altro.
Grid	Definisce un'area griglia flessibile costituita da righe e colonne.
StackPanel	Dispone gli elementi figlio su una sola riga che può essere orientata orizzontalmente o verticalmente.

NOME ELEMENTO PANEL	DESCRIZIONE
VirtualizingPanel	Fornisce un Framework per Panel gli elementi che virtualizzano la raccolta dei dati figlio. Questa è una classe abstract.
WrapPanel	Posiziona gli elementi figlio in sequenza da sinistra a destra, interrompendo il contenuto al raggiungimento del bordo della casella contenitore e facendolo ripartire dalla riga successiva. L'ordinamento successivo viene eseguito in sequenza dall'alto verso il basso o da destra a sinistra, a seconda del Orientation valore della proprietà.

Per le applicazioni che richiedono un layout non possibile utilizzando uno degli elementi [Panel](#) predefiniti, è possibile ottenere comportamenti di layout personalizzati ereditando da [Panel](#) ed eseguendo l'override dei [MeasureOverride](#) metodi e [ArrangeOverride](#).

Considerazioni sulle prestazioni del layout

Il layout è un processo ricorsivo. Ogni elemento figlio in una [Children](#) raccolta viene elaborato durante ogni chiamata del sistema di layout. Di conseguenza, è consigliabile evitare l'attivazione del sistema di layout quando non è necessaria. Le considerazioni seguenti possono essere utili per ottenere prestazioni migliori.

- Identificare le modifiche dei valori di proprietà che forzeranno un aggiornamento ricorsivo da parte del sistema di layout.

Le proprietà di dipendenza i cui valori possono provocare l'inizializzazione del sistema di layout sono contrassegnate con flag pubblici. [AffectsMeasure](#) [AffectsArrange](#) forniscono indicazioni utili per le modifiche dei valori di proprietà che forzano un aggiornamento ricorsivo da parte del sistema di layout. In generale, qualsiasi proprietà che può influire sulle dimensioni del rettangolo di delimitazione di un elemento deve [AffectsMeasure](#) avere un flag impostato su true. Per altre informazioni, vedere [Panoramica sulle proprietà di dipendenza](#).

- Quando possibile, utilizzare un [RenderTransform](#) anziché un [LayoutTransform](#)

Un [LayoutTransform](#) oggetto può essere un metodo molto utile per influenzare il contenuto di interfaccia utente un oggetto. Tuttavia, se l'effetto della trasformazione non ha alcun impatto sulla posizione di altri elementi, è preferibile usare [RenderTransform](#) invece, perché [RenderTransform](#) non richiama il sistema di layout. [LayoutTransform](#) applica la trasformazione e impone un aggiornamento ricorsivo del layout per tenere conto della nuova posizione dell'elemento interessato.

- Evitare chiamate non necessarie [UpdateLayout](#).

Il [UpdateLayout](#) metodo impone un aggiornamento ricorsivo del layout e spesso non è necessario. Se non si ha la certezza della necessità di un aggiornamento completo, lasciare che sia il sistema di layout a chiamare automaticamente questo metodo.

- Quando si lavora con una [Children](#) raccolta di grandi dimensioni, [VirtualizingStackPanel](#) è consigliabile usare un [StackPanel](#) anziché un normale.

Virtualizzando la raccolta figlio, il [VirtualizingStackPanel](#) mantiene solo gli oggetti in memoria che sono attualmente all'interno del viewport del padre. Come conseguenza, le prestazioni risultano notevolmente migliorate nella maggior parte degli scenari.

Rendering dei sub-pixel e arrotondamento del layout

Il sistema grafico WPF USA unità indipendenti dal dispositivo per abilitare la risoluzione e l'indipendenza del

dispositivo. Ogni Device Independent Pixel scala automaticamente con l'impostazione dpi (punti per pollice) del sistema. In questo modo, le applicazioni WPF vengono ridimensionate correttamente per diverse impostazioni dpi e l'applicazione è in grado di riconoscere automaticamente i dpi.

Questa indipendenza dpi può tuttavia creare un rendering perimetrale irregolare a causa dell'anti-aliasing. Questi elementi, osservabili in genere come bordi sfocati o semitrasparenti, possono essere presenti quando la posizione dei bordi si trova al centro di un pixel del dispositivo invece che tra pixel del dispositivo. Il sistema di layout permette di ovviare a questo problema con l'arrotondamento del layout. L'arrotondamento del layout avviene quando il sistema di layout arrotonda valori di pixel non integrali durante il passaggio di layout.

L'arrotondamento del layout è disabilitato per impostazione predefinita. Per abilitare l'arrotondamento del layout, [UseLayoutRounding](#) impostare la `true` proprietà su [FrameworkElement](#) on any. Poiché si tratta di una proprietà di dipendenza, il valore viene propagato a tutti gli elementi figlio nell'albero visuale. Per abilitare l'arrotondamento del layout per l'intera interfaccia [UseLayoutRounding](#) utente `true`, impostare su sul contenitore radice. Per un esempio, vedere [UseLayoutRounding](#).

Argomenti successivi

La capacità di identificare il modo in cui gli elementi vengono misurati e disposti è il primo passaggio per la comprensione del layout. Per ulteriori informazioni sugli elementi disponibili [Panel](#), vedere [Cenni preliminari sui pannelli](#). Per meglio determinare le diverse proprietà di posizionamento che possono influire sul layout, vedere [Panoramica su allineamento, margini e spaziatura interna](#). Quando si è pronti per riunire tutti gli elementi in un'applicazione leggera, vedere [procedura dettagliata: Prima applicazione desktop WPF](#).

Vedere anche

- [FrameworkElement](#)
- [UIElement](#)
- [Cenni preliminari sugli elementi Panel](#)
- [Panoramica su allineamento, margini e spaziatura interna](#)
- [Ottimizzazione delle prestazioni: layout e progettazione](#)

Migrazione e interoperabilità

31/01/2020 • 2 minutes to read • [Edit Online](#)

Questa pagina contiene collegamenti a documenti che illustrano come implementare l'interoperatività tra applicazioni Windows Presentation Foundation (WPF) e altri tipi di applicazioni Microsoft Windows.

In questa sezione

[Interoperatività di WPF e Windows Form](#)

[Interoperatività di WPF e Win32](#)

[Interoperatività di WPF e Direct3D9](#)

Riferimenti

TERMINE	DEFINIZIONE
WindowsFormsHost	Elemento che è possibile utilizzare per ospitare un controllo Windows Forms come elemento di una pagina di WPF.
ElementHost	Controllo Windows Forms che è possibile utilizzare per ospitare un controllo Windows Presentation Foundation (WPF).
HwndSource	Ospita un'area WPF all'interno di un'applicazione Win32.
HwndHost	Classe di base per WindowsFormsHost , definisce alcune funzionalità di base utilizzate da tutte le tecnologie basate su HWND quando sono ospitate da un'applicazione WPF. Sottoclassare questo oggetto per ospitare una finestra Win32 in un'applicazione WPF.
BrowserInteropHelper	Classe helper per la creazione di report sulle condizioni dell'ambiente browser per un'applicazione WPF ospitata da un browser.

Sezioni correlate

Interoperatività di WPF e Windows Form

31/01/2020 • 18 minutes to read • [Edit Online](#)

WPF e Windows Forms presentano due architetture diverse per la creazione di interfacce dell'applicazione. Lo spazio dei nomi [System.Windows.Forms.Integration](#) fornisce classi che consentono scenari comuni di interoperatività. Le due classi principali che implementano le funzionalità di interoperatività sono [WindowsFormsHost](#) e [ElementHost](#). Questo argomento descrive gli scenari di interoperatività supportati e indica quelli non supportati.

NOTE

Particolare attenzione viene dedicata allo scenario del *controllo ibrido*. Un controllo ibrido è costituito da un controllo di una tecnologia annidato in un controllo di un'altra tecnologia. Questa situazione viene anche definita *interoperatività annidata*. Un *controllo ibrido multilivello* prevede più di un livello di annidamento dei controlli ibridi. Un esempio di interoperabilità annidata multilivello è un controllo Windows Forms che contiene un controllo WPF, che contiene un altro controllo Windows Forms. I controlli ibridi multilivello non sono supportati.

Hosting di controlli Windows Form in WPF

Gli scenari di interoperatività seguenti sono supportati quando un controllo WPF ospita un controllo Windows Forms:

- Il controllo WPF può ospitare uno o più controlli Windows Forms con XAML.
- Può ospitare uno o più controlli Windows Forms usando il codice.
- Può ospitare Windows Forms controlli contenitore che contengono altri controlli Windows Forms.
- Può ospitare un form Master-Details con un WPF master e Windows Forms dettagli.
- Può ospitare un form Master-Details con un Windows Forms master e WPF dettagli.
- Può ospitare uno o più controlli ActiveX.
- Può ospitare uno o più controlli composti.
- Può ospitare controlli ibridi tramite Extensible Application Markup Language (XAML).
- Può ospitare controlli ibridi tramite il codice.

Supporto del layout

Nell'elenco seguente vengono descritte le limitazioni note quando l'elemento [WindowsFormsHost](#) tenta di integrare il controllo Windows Forms ospitato nel sistema di layout di WPF.

- In alcuni casi, i controlli Windows Forms non possono essere ridimensionati o possono essere ridimensionati solo a dimensioni specifiche. Ad esempio, un controllo Windows Forms [ComboBox](#) supporta solo una singola altezza, che è definita dalle dimensioni del carattere del controllo. In un WPF layout dinamico, che presuppone che gli elementi possano essere allungati verticalmente, un controllo [ComboBox](#) ospitato non si estenderà come previsto.
- Non è possibile ruotare o inclinare i controlli Windows Forms. Ad esempio, quando si ruota l'interfaccia utente di 90 gradi, i controlli Windows Forms ospitati manterranno la posizione verticale.
- Nella maggior parte dei casi, i controlli Windows Forms non supportano la scalabilità proporzionale. Anche

se le dimensioni complessive del controllo vengono ridimensionate, i controlli figlio e gli elementi componente del controllo potrebbero non venire ridimensionati come previsto. Questa limitazione dipende dal modo in cui ogni controllo Windows Forms supporta il ridimensionamento.

- In un'interfaccia utente WPF è possibile modificare l'ordine z degli elementi per controllare il comportamento di sovrapposizione. Un controllo Windows Forms ospitato viene disegnato in un HWND separato, pertanto viene sempre disegnato sopra gli elementi di WPF.
- I controlli Windows Forms supportano la scalabilità automatica in base alle dimensioni del carattere. In un'interfaccia utente WPF, modificando le dimensioni del carattere non si ridimensiona l'intero layout, anche se è possibile che vengano ridimensionati dinamicamente singoli elementi.

Proprietà di ambiente

Alcune delle proprietà di ambiente dei controlli WPF hanno Windows Forms equivalenti. Queste proprietà di ambiente vengono propagate ai controlli Windows Forms ospitati ed esposte come proprietà pubbliche nel controllo [WindowsFormsHost](#). Il controllo [WindowsFormsHost](#) converte ogni proprietà di ambiente WPF nell'equivalente Windows Forms.

Per altre informazioni, vedere [Mapping di proprietà di Windows Form e WPF](#).

Comportamento di

La tabella seguente descrive il comportamento dell'interoperatività.

COMPORTAMENTO DI	SUPPORTATO	NON SUPPORTATO
Trasparenza	Il rendering del controllo Windows Forms supporta la trasparenza. Lo sfondo del controllo WPF padre può diventare lo sfondo dei controlli Windows Forms ospitati.	Alcuni controlli Windows Forms non supportano la trasparenza. Ad esempio, i controlli TextBox e ComboBox non saranno trasparenti se ospitati da WPF.

COMPORTAMENTO DI	SUPPORTATO	NON SUPPORTATO
Tabulazione	<p>L'ordine di tabulazione per i controlli Windows Forms ospitati è identico a quello in cui tali controlli sono ospitati in un'applicazione basata su Windows Forms.</p> <p>La tabulazione da un controllo WPF a un controllo Windows Forms con il tasto TAB e i tasti MAIUSC + TAB funzionano come di consueto.</p> <p>Windows Forms controlli che hanno un valore della proprietà <code>TabStop</code> di <code>false</code> non ricevono lo stato attivo quando l'utente esegue la tabulazione dei controlli.</p> <ul style="list-style-type: none"> -Ogni controllo <code>WindowsFormsHost</code> dispone di un valore <code>TabIndex</code>, che determina quando il controllo <code>WindowsFormsHost</code> riceve lo stato attivo. -Windows Forms i controlli contenuti all'interno di un contenitore di <code>WindowsFormsHost</code> seguono l'ordine specificato dalla proprietà <code>TabIndex</code>. La tabulazione dall'ultimo indice di tabulazione assegna lo stato attivo al successivo controllo WPF, se presente. Se non esiste un altro controllo WPF attivabile, la tabulazione torna al primo controllo Windows Forms nell'ordine di tabulazione. - <code>TabIndex</code> valori per i controlli all'interno del <code>WindowsFormsHost</code> sono relativi ai controlli di pari livello Windows Forms contenuti nel controllo <code>WindowsFormsHost</code>. - La tabulazione rispetta il comportamento specifico del controllo. Se, ad esempio, si preme il tasto TAB in un controllo <code>TextBox</code> con un valore <code>AcceptsTab</code> proprietà, <code>true</code> immette una scheda nella casella di testo anziché spostare lo stato attivo. 	Non applicabile.

COMPORTAMENTO DI	SUPPORTATO	NON SUPPORTATO
Navigazione con i tasti di direzione	<p>-La navigazione con i tasti di direzione nel controllo WindowsFormsHost è identica a quella di un normale controllo contenitore Windows Forms: i tasti freccia su e freccia sinistra selezionano il controllo precedente, la freccia giù e i tasti freccia destra selezionano il controllo successivo.</p> <p>-I tasti freccia su e freccia sinistra dal primo controllo contenuto nel controllo WindowsFormsHost eseguono la stessa azione del tasto di scelta rapida MAIUSC + TAB. Se è presente un controllo WPF attivabile, lo stato attivo si sposta all'esterno del controllo WindowsFormsHost. Questo comportamento differisce dal comportamento di ContainerControl standard in quanto non viene eseguito il wrapping all'ultimo controllo. Se non esiste un altro controllo WPF attivabile, lo stato attivo torna all'ultimo controllo Windows Forms nell'ordine di tabulazione.</p> <p>-I tasti freccia giù e freccia destra dall'ultimo controllo contenuto nel controllo WindowsFormsHost eseguono la stessa azione del tasto TAB. Se è presente un controllo WPF attivabile, lo stato attivo si sposta all'esterno del controllo WindowsFormsHost. Questo comportamento differisce dal comportamento di ContainerControl standard in quanto non viene eseguito il wrapping al primo controllo. Se non esiste un altro controllo WPF attivabile, lo stato attivo torna al primo controllo Windows Forms nell'ordine di tabulazione.</p>	Non applicabile.
Tasti di scelta rapida	I tasti di scelta rapida funzionano come di consueto, salvo diversa indicazione nella colonna "Non supportato".	I tasti di scelta rapida duplicati tra diverse tecnologie non funzionano come i normali tasti di scelta rapida duplicati. Quando un acceleratore viene duplicato tra le tecnologie, con almeno un controllo Windows Forms e l'altro in un controllo WPF, il controllo Windows Forms riceve sempre il tasto di scelta rapida. Lo stato attivo non passa tra i controlli quando il tasto di scelta rapida duplicato viene premuto.

COMPORTAMENTO DI	SUPPORTATO	NON SUPPORTATO
Tasti di scelta rapida	Le combinazioni di tasti funzionano come di consueto, salvo diversa indicazione nella colonna "Non supportato".	<ul style="list-style-type: none"> -Windows Forms i tasti di scelta rapida gestiti in fase di pre-elaborazione hanno sempre la precedenza sui tasti di scelta rapida WPF. Se, ad esempio, è stato definito un controllo ToolStrip con i tasti di scelta rapida CTRL + S ed è presente un comando WPF associato a CTRL + S, il gestore di controllo ToolStrip viene sempre richiamato per primo, indipendentemente dallo stato attivo. -Windows Forms i tasti di scelta rapida gestiti dall'evento KeyDown vengono elaborati per ultimi in WPF. È possibile evitare questo comportamento eseguendo l'override del metodo IsInputKey del controllo Windows Forms o gestendo l'evento PreviewKeyDown. Restituire <code>true</code> dal metodo IsInputKey oppure impostare il valore della proprietà PreviewKeyDownEventArgs.IsInputKey su <code>true</code> nel gestore dell'evento PreviewKeyDown.
Comportamento di AcceptsReturn, AcceptsTab e di altri controlli	Le proprietà che modificano il comportamento predefinito della tastiera funzionano come di consueto, presupponendo che il controllo Windows Forms esegua l'override del metodo IsInputKey per restituire <code>true</code> .	Windows Forms controlli che modificano il comportamento predefinito della tastiera gestendo l'evento KeyDown vengono elaborati per ultimi nel controllo WPF host. Per questo motivo possono generare un comportamento imprevisto.
Eventi Enter e Leave	Quando lo stato attivo non passa al controllo ElementHost che lo contiene, gli eventi Enter e Leave vengono generati come di consueto quando lo stato attivo cambia in un singolo controllo WindowsFormsHost .	<ul style="list-style-type: none"> Gli eventi Enter e Leave non vengono generati quando si verificano i cambiamenti seguenti dello stato attivo: -Dall'interno all'esterno di un controllo WindowsFormsHost. -Dall'esterno all'interno di un controllo WindowsFormsHost. -All'esterno di un controllo WindowsFormsHost. -Da un controllo Windows Forms ospitato in un controllo WindowsFormsHost a un controllo ElementHost ospitato all'interno della stessa WindowsFormsHost.
Multithreading	Sono supportati tutti i tipi di multithreading.	Entrambe le tecnologie Windows Forms e WPF presuppongono un modello di concorrenza a thread singolo. Durante il debug, le chiamate agli oggetti framework da altri thread generano un'eccezione per applicare questo requisito.
Sicurezza -	Tutti gli scenari di interoperatività richiedono l'attendibilità totale.	Non sono ammessi scenari di interoperatività con attendibilità parziale.

COMPORTAMENTO DI	SUPPORTATO	NON SUPPORTATO
Accessibilità	Sono supportati tutti gli scenari di accessibilità. I prodotti di Assistive Technology funzionano correttamente quando vengono usati per applicazioni ibride che contengono controlli sia Windows Forms che WPF.	Non applicabile.
Appunti	Tutte le operazioni degli Appunti funzionano come di consueto. Sono incluse le trascinamenti e le paste tra Windows Forms e WPF i controlli.	Non applicabile.
Funzionalità di trascinamento	Tutte le operazioni di trascinamento funzionano come di consueto. Sono incluse le operazioni tra Windows Forms e i controlli di WPF.	Non applicabile.

Hosting di controlli WPF in Windows Form

Gli scenari di interoperatività seguenti sono supportati quando un controllo Windows Forms ospita un controllo WPF:

- Hosting di uno o più controlli WPF tramite il codice.
- Associazione di una finestra delle proprietà a uno o più controlli WPF ospitati.
- Hosting di una o più pagine WPF in un form.
- Apertura di una finestra WPF.
- Hosting di un form Master-Details con un Windows Forms master e WPF dettagli.
- Hosting di un form Master-Details con un WPF master e Windows Forms dettagli.
- Hosting di controlli WPF personalizzati.
- Hosting di controlli ibridi.

Proprietà di ambiente

Alcune delle proprietà di ambiente dei controlli Windows Forms hanno WPF equivalenti. Queste proprietà di ambiente vengono propagate ai controlli WPF ospitati ed esposte come proprietà pubbliche nel controllo [ElementHost](#). Il controllo [ElementHost](#) converte ogni proprietà di ambiente Windows Forms nell'equivalente WPF.

Per altre informazioni, vedere [Mapping di proprietà di Windows Form e WPF](#).

Comportamento di

La tabella seguente descrive il comportamento dell'interoperatività.

COMPORTAMENTO DI	SUPPORTATO	NON SUPPORTATO
Trasparenza	Il rendering del controllo WPF supporta la trasparenza. Lo sfondo del controllo Windows Forms padre può diventare lo sfondo dei controlli WPF ospitati.	Non applicabile.

COMPORTAMENTO DI	SUPPORTATO	NON SUPPORTATO
Multithreading	Sono supportati tutti i tipi di multithreading.	Entrambe le tecnologie Windows Forms e WPF presuppongono un modello di concorrenza a thread singolo. Durante il debug, le chiamate agli oggetti framework da altri thread generano un'eccezione per applicare questo requisito.
Sicurezza -	Tutti gli scenari di interoperatività richiedono l'attendibilità totale.	Non sono ammessi scenari di interoperatività con attendibilità parziale.
Accessibilità	Sono supportati tutti gli scenari di accessibilità. I prodotti di Assistive Technology funzionano correttamente quando vengono usati per applicazioni ibride che contengono controlli sia Windows Forms che WPF.	Non applicabile.
Appunti	Tutte le operazioni degli Appunti funzionano come di consueto. Sono incluse le trascinamenti e le paste tra Windows Forms e WPF i controlli.	Non applicabile.
Funzionalità di trascinamento	Tutte le operazioni di trascinamento funzionano come di consueto. Sono incluse le operazioni tra Windows Forms e i controlli di WPF.	Non applicabile.

Vedere anche

- [ElementHost](#)
- [WindowsFormsHost](#)
- [Procedura dettagliata: hosting di controlli Windows Form in WPF](#)
- [Procedura dettagliata: Hosting di controlli Windows Form compositi in WPF](#)
- [Procedura dettaglia: hosting di un controllo WPF composito in Windows Form](#)
- [Mapping di proprietà di Windows Form e WPF](#)

Architettura di input per l'interoperabilità tra Windows Form e WPF

31/01/2020 • 9 minutes to read • [Edit Online](#)

L'interoperabilità tra il WPF e Windows Forms richiede che entrambe le tecnologie dispongano dell'elaborazione dell'input da tastiera appropriata. Questo argomento descrive il modo in cui queste tecnologie implementano l'elaborazione dei messaggi e della tastiera per abilitare l'interoperatività uniforme nelle applicazioni ibride.

In questo argomento sono contenute le seguenti sottosezioni:

- Moduli e finestre di dialogo non modali
- Elaborazione della tastiera e del messaggio WindowsFormsHost
- Elaborazione di messaggi e tastiera ElementHost

Moduli e finestre di dialogo non modali

Chiamare il metodo [EnableWindowsFormsInterop](#) sull'elemento [WindowsFormsHost](#) per aprire un form o una finestra di dialogo non modale da un'applicazione basata su WPF.

Chiamare il metodo [EnableModelessKeyboardInterop](#) sul controllo [ElementHost](#) per aprire una pagina di WPF non modale in un'applicazione basata su Windows Forms.

Elaborazione della tastiera e del messaggio WindowsFormsHost

Quando è ospitato da un'applicazione basata su WPF, Windows Forms l'elaborazione di tastiera e messaggi è costituita dai seguenti elementi:

- La classe [WindowsFormsHost](#) acquisisce i messaggi dal ciclo di messaggi WPF, implementato dalla classe [ComponentDispatcher](#).
- La classe [WindowsFormsHost](#) crea un ciclo di messaggi Windows Forms surrogati per garantire che si verifichi l'elaborazione ordinaria di Windows Forms tastiera.
- La classe [WindowsFormsHost](#) implementa l'interfaccia [IKeyboardInputSink](#) per coordinare la gestione dello stato attivo con WPF.
- I controlli [WindowsFormsHost](#) si registrano e avviano i cicli di messaggi.

Le sezioni seguenti descrivono in modo più dettagliato queste parti del processo.

Acquisizione di messaggi dal ciclo di messaggi WPF

La classe [ComponentDispatcher](#) implementa la gestione del ciclo di messaggi per WPF. La classe [ComponentDispatcher](#) fornisce hook per consentire ai client esterni di filtrare i messaggi prima di WPF li elabora.

L'implementazione dell'interoperatività gestisce l'evento [ComponentDispatcher.ThreadFilterMessage](#), che consente ai controlli Windows Forms di elaborare i messaggi prima di WPF controlli.

Ciclo di messaggi Windows Forms surrogati

Per impostazione predefinita, la classe [System.Windows.Forms.Application](#) contiene il ciclo di messaggi primario per le applicazioni Windows Forms. Durante l'interoperatività, il ciclo di messaggi Windows Forms non elabora i messaggi. Pertanto, è necessario riprodurre questa logica. Il gestore per l'evento

`ComponentDispatcher.ThreadFilterMessage` esegue i passaggi seguenti:

1. Filtra il messaggio usando l'interfaccia `IMessageFilter`.
2. Chiama il metodo `Control.PreProcessMessage`.
3. Converte e invia il messaggio, se necessario.
4. Passa il messaggio al controllo di hosting, se nessun altro controllo elabora il messaggio.

Implementazione di `IKeyboardInputSink`

Il ciclo di messaggi surrogati gestisce la gestione della tastiera. Pertanto, il `IKeyboardInputSink.Tablnto` metodo è l'unico membro `IKeyboardInputSink` che richiede un'implementazione nella classe `WindowsFormsHost`.

Per impostazione predefinita, la classe `HwndHost` restituisce `false` per la relativa implementazione di `IKeyboardInputSink.Tablnto`. In questo modo si impedisce la tabulazione da un controllo WPF a un controllo Windows Forms.

L'implementazione `WindowsFormsHost` del metodo `IKeyboardInputSink.Tablnto` esegue i passaggi seguenti:

1. Trova il primo o l'ultimo controllo Windows Forms contenuto dal controllo `WindowsFormsHost` e che può ricevere lo stato attivo. La scelta del controllo dipende dalle informazioni di attraversamento.
2. Imposta lo stato attivo sul controllo e restituisce `true`.
3. Se nessun controllo può ricevere lo stato attivo, restituisce `false`.

Registrazione `WindowsFormsHost`

Quando viene creato l'handle della finestra per un controllo `WindowsFormsHost`, il controllo `WindowsFormsHost` chiama un metodo statico interno che registra la sua presenza per il ciclo di messaggi.

Durante la registrazione, il controllo `WindowsFormsHost` esamina il ciclo di messaggi. Se il ciclo di messaggi non è stato avviato, viene creato il gestore dell'evento `ComponentDispatcher.ThreadFilterMessage`. Il ciclo di messaggi viene considerato in esecuzione quando viene collegato il gestore dell'evento `ComponentDispatcher.ThreadFilterMessage`.

Quando l'handle della finestra viene eliminato definitivamente, il controllo `WindowsFormsHost` si rimuove dalla registrazione.

Elaborazione di messaggi e tastiera ElementHost

Quando è ospitato da un Windows Forms Application, WPF elaborazione della tastiera e del messaggio è costituita dagli elementi seguenti:

- implementazioni dell'interfaccia `HwndSource`, `IKeyboardInputSink` e `IKeyboardInputSite`.
- Tabulazione e tasti di direzione.
- Chiavi di comando e chiavi della finestra di dialogo.
- Elaborazione dell'acceleratore Windows Forms.

Le sezioni seguenti descrivono queste parti in modo più dettagliato.

Implementazioni di interfacce

In Windows Forms, i messaggi della tastiera vengono instradati all'handle della finestra del controllo con lo stato attivo. Nel controllo `ElementHost` questi messaggi vengono instradati all'elemento ospitato. A tale scopo, il controllo `ElementHost` fornisce un'istanza di `HwndSource`. Se il controllo `ElementHost` dispone dello stato attivo, l'istanza `HwndSource` instrada la maggior parte degli input da tastiera, in modo che possa essere elaborata dalla classe `InputManager` di WPF.

La classe [HwndSource](#) implementa le interfacce [IKeyboardInputSink](#) e [IKeyboardInputSite](#).

L'interoperatività della tastiera si basa sull'implementazione del metodo [OnNoMoreTabStops](#) per gestire il tasto TAB e l'input del tasto di direzione che sposta lo stato attivo fuori dagli elementi ospitati.

Tabulazione e tasti di direzione

Viene eseguito il mapping della logica di selezione Windows Forms ai metodi [IKeyboardInputSink.TabInto](#) e [OnNoMoreTabStops](#) per implementare la navigazione tra le schede e i tasti freccia. Eseguendo l'override del metodo [Select](#) viene eseguito questo mapping.

Chiavi di comando e chiavi della finestra di dialogo

Per fornire WPF la prima opportunità per elaborare le chiavi di comando e le chiavi di dialogo, Windows Forms la pre-elaborazione dei comandi è connessa al metodo [TranslateAccelerator](#). L'override del metodo [Control.ProcessCmdKey](#) connette le due tecnologie.

Con il metodo [TranslateAccelerator](#), gli elementi ospitati sono in grado di gestire qualsiasi messaggio chiave, ad esempio WM_KEYDOWN, WM_KEYUP, WM_SYSKEYDOWN o WM_SYSKEYUP, inclusi i tasti di comando, ad esempio TAB, ENTER, ESC e i tasti di direzione. Se un messaggio chiave non viene gestito, viene inviato il Windows Forms gerarchia predecessore per la gestione.

Elaborazione acceleratore

Per elaborare correttamente i tasti di scelta rapida, è necessario che Windows Forms elaborazione dell'acceleratore sia connessa alla classe di [AccessKeyManager](#) WPF. Inoltre, tutti i messaggi di WM_CHAR devono essere indirizzati correttamente agli elementi ospitati.

Poiché l'implementazione [HwndSource](#) predefinita del metodo [TranslateChar](#) restituisce `false`, WM_CHAR i messaggi vengono elaborati utilizzando la logica seguente:

- Viene eseguito l'override del metodo [Control.IsInputChar](#) per assicurarsi che tutti i messaggi di WM_CHAR vengano inviati agli elementi ospitati.
- Se viene premuto il tasto ALT, il messaggio viene WM_SYSCHAR. Windows Forms non esegue la pre-elaborazione di questo messaggio tramite il metodo [IsInputChar](#). Pertanto, viene eseguito l'override del metodo [ProcessMnemonic](#) per eseguire una query sul WPF [AccessKeyManager](#) per un tasto di scelta rapida registrato. Se viene trovato un acceleratore registrato, [AccessKeyManager](#) lo elabora.
- Se il tasto ALT non viene premuto, la classe WPF [InputManager](#) elabora l'input non gestito. Se l'input è un acceleratore, il [AccessKeyManager](#) lo elabora. L'evento [PostProcessInput](#) viene gestito per WM_CHAR messaggi non elaborati.

Quando l'utente preme il tasto ALT, i segnali visivi Accelerator vengono visualizzati nell'intero form. Per supportare questo comportamento, tutti i controlli [ElementHost](#) sul form attivo ricevono WM_SYSKEYDOWN messaggi, indipendentemente dal controllo con lo stato attivo.

I messaggi vengono inviati solo ai controlli [ElementHost](#) nel modulo attivo.

Vedere anche

- [EnableWindowsFormsInterop](#)
- [EnableModelessKeyboardInterop](#)
- [ElementHost](#)
- [WindowsFormsHost](#)
- [Procedura dettagliata: Hosting di controlli Windows Form compositi in WPF](#)
- [Procedura dettaglia: hosting di un controllo WPF composito in Windows Form](#)
- [Interoperatività di WPF e Win32](#)

Considerazioni sul layout per l'elemento WindowsFormsHost

10/02/2020 • 12 minutes to read • [Edit Online](#)

In questo argomento viene descritto il modo in cui l'elemento [WindowsFormsHost](#) interagisce con il sistema di layout di WPF.

WPF e Windows Forms supportano la logica diversa, ma simile, per il ridimensionamento e il posizionamento di elementi in un form o una pagina. Quando si crea un'interfaccia utente ibrida che ospita Windows Forms controlli in WPF, l'elemento [WindowsFormsHost](#) integra i due schemi di layout.

Differenze nel layout tra WPF e Windows Forms

WPF usa il layout indipendente dalla risoluzione. Tutte le dimensioni del layout WPF vengono specificate usando i *pixel indipendenti dal dispositivo*. Un pixel indipendente dal dispositivo è un 90 ° di un pollice di dimensioni e indipendente dalla risoluzione, quindi si ottengono risultati simili indipendentemente dal fatto che venga eseguito il rendering in un monitor 72 dpi o in una stampante 19.200-dpi.

WPF si basa anche sul *layout dinamico*. Ciò significa che un elemento dell'interfaccia utente dispone di se stesso in un form o in una pagina in base al relativo contenuto, al relativo contenitore di layout padre e alle dimensioni dello schermo disponibili. Il layout dinamico semplifica la localizzazione regolando automaticamente le dimensioni e la posizione degli elementi dell'interfaccia utente quando le stringhe che contengono cambiano la lunghezza.

Il layout in Windows Forms è dipendente dal dispositivo e con maggiore probabilità è statico. In genere, i controlli Windows Forms sono posizionati in modo assoluto in un modulo che usa dimensioni specificate in pixel hardware. Tuttavia, Windows Forms supporta alcune funzionalità di layout dinamico, come riepilogato nella tabella seguente.

Funzionalità layout	Descrizione
Ridimensionamento automatico	Alcuni Windows Forms controlli si ridimensionano per visualizzare correttamente il contenuto. Per ulteriori informazioni, vedere Panoramica delle proprietà AutoSize .
Ancoraggio e ancoraggio	I controlli Windows Forms supportano il posizionamento e il ridimensionamento in base al contenitore padre. Per altre informazioni, vedere Control.Anchor e Control.Dock .
Scalabilità automatica	I controlli contenitore si ridimensionano e i relativi elementi figlio in base alla risoluzione del dispositivo di output o alle dimensioni, in pixel, del tipo di carattere del contenitore predefinito. Per ulteriori informazioni, vedere la pagina relativa alla scalabilità automatica in Windows Forms .
Contenitori di layout	I controlli FlowLayoutPanel e TableLayoutPanel dispongono i controlli figlio e le dimensioni stesse in base al relativo contenuto.

Limitazioni del layout

In generale, i controlli Windows Forms non possono essere ridimensionati e trasformati nella misura massima

consentita nell'WPF. Nell'elenco seguente vengono descritte le limitazioni note quando l'elemento [WindowsFormsHost](#) tenta di integrare il controllo Windows Forms ospitato nel sistema di layout di WPF.

- In alcuni casi, i controlli Windows Forms non possono essere ridimensionati o possono essere ridimensionati solo a dimensioni specifiche. Ad esempio, un controllo Windows Forms [ComboBox](#) supporta solo una singola altezza, che è definita dalle dimensioni del carattere del controllo. In un WPF layout dinamico in cui gli elementi possono estendersi verticalmente, un controllo [ComboBox](#) ospitato non si estenderà come previsto.
- Non è possibile ruotare o inclinare i controlli Windows Forms. L'elemento [WindowsFormsHost](#) genera l'evento [LayoutError](#) se si applica una trasformazione di inclinazione o rotazione. Se non si gestisce l'evento [LayoutError](#), viene generata un'[InvalidOperationException](#).
- Nella maggior parte dei casi, i controlli Windows Forms non supportano la scalabilità proporzionale. Anche se le dimensioni complessive del controllo vengono ridimensionate, i controlli figlio e gli elementi componente del controllo potrebbero non venire ridimensionati come previsto. Questa limitazione dipende dal modo in cui ogni controllo Windows Forms supporta il ridimensionamento. Non è inoltre possibile ridimensionare Windows Forms controlli fino a una dimensione di 0 pixel.
- I controlli Windows Forms supportano la scalabilità automatica, in cui il form si ridimensiona automaticamente e i controlli in base alle dimensioni del carattere. In un'interfaccia utente WPF, modificando le dimensioni del carattere non si ridimensiona l'intero layout, anche se è possibile che vengano ridimensionati dinamicamente singoli elementi.

Ordine Z

In un'interfaccia utente WPF è possibile modificare l'ordine z degli elementi per controllare il comportamento di sovrapposizione. Un controllo Windows Forms ospitato viene disegnato in un HWND separato, pertanto viene sempre disegnato sopra gli elementi di WPF.

Viene anche disegnato un controllo Windows Forms ospitato su tutti gli elementi di [Adorner](#).

Comportamento del layout

Nelle sezioni seguenti vengono descritti aspetti specifici del comportamento del layout quando si ospitano controlli Windows Forms in WPF.

Ridimensionamento, conversione di unità e indipendenza dei dispositivi

Ogni volta che l'elemento [WindowsFormsHost](#) esegue operazioni che coinvolgono WPF e Windows Forms dimensioni, sono coinvolti due sistemi di Coordinate: i pixel indipendenti dal dispositivo per WPF e i pixel hardware per Windows Forms. Pertanto, per ottenere un layout coerente è necessario applicare le conversioni appropriate di unità e scalabilità.

La conversione tra i sistemi di coordinate dipende dalla risoluzione del dispositivo corrente e da eventuali trasformazioni di layout o rendering applicate all'elemento [WindowsFormsHost](#) o ai relativi predecessori.

Se il dispositivo di output è 96 dpi e non è stato applicato alcun ridimensionamento all'elemento [WindowsFormsHost](#), un pixel indipendente dal dispositivo è uguale a un pixel hardware.

Tutti gli altri casi richiedono la scalabilità del sistema di coordinate. Il controllo ospitato non viene ridimensionato. Al contrario, l'elemento [WindowsFormsHost](#) tenta di ridimensionare il controllo ospitato e tutti i relativi controlli figlio. Poiché Windows Forms non supporta completamente il ridimensionamento, l'elemento [WindowsFormsHost](#) viene ridimensionato in base al livello supportato da determinati controlli.

Eseguire l'override del metodo [ScaleChild](#) per fornire un comportamento di ridimensionamento personalizzato per il controllo Windows Forms ospitato.

Oltre al ridimensionamento, l'elemento [WindowsFormsHost](#) gestisce i case di arrotondamento e overflow, come

descritto nella tabella seguente.

PROBLEMA DI CONVERSIONE	DESCRIZIONE
Arrotondamento	WPF dimensioni in pixel indipendenti dal dispositivo vengono specificate come <code>double</code> e Windows Forms dimensioni dei pixel hardware vengono specificate come <code>int</code> . Nei casi in cui le dimensioni basate su <code>double</code> vengono convertite in dimensioni basate su <code>int</code> , l'elemento WindowsFormsHost usa l'arrotondamento standard, in modo che i valori frazionari minori di 0,5 siano arrotondati per difetto a 0.
Overflow	Quando l'elemento WindowsFormsHost converte da valori <code>double</code> a valori <code>int</code> , è possibile che l'overflow sia possibile. I valori maggiori di MaxValue vengono impostati su MaxValue .

Proprietà correlate al layout

Le proprietà che controllano il comportamento del layout nei controlli Windows Forms e negli elementi WPF sono mappate in modo appropriato dall'elemento [WindowsFormsHost](#). Per altre informazioni, vedere [Mapping di proprietà di Windows Form e WPF](#).

Modifiche del layout nel controllo ospitato

Le modifiche di layout nel controllo Windows Forms ospitato vengono propagate in WPF per attivare gli aggiornamenti del layout. Il metodo [InvalidateMeasure](#) su [WindowsFormsHost](#) garantisce che le modifiche apportate al layout nel controllo ospitato provochino l'esecuzione del motore di layout di WPF.

Controlli Windows Forms continuamente dimensionati

Windows Forms controlli che supportano il ridimensionamento continuo completamente interagiscono con il sistema di layout di WPF. L'elemento [WindowsFormsHost](#) usa i metodi [MeasureOverride](#) e [ArrangeOverride](#) come di consueto per ridimensionare e disporre il controllo Windows Forms ospitato.

Algoritmo di dimensionamento

L'elemento [WindowsFormsHost](#) utilizza la procedura seguente per ridimensionare il controllo ospitato:

1. L'elemento [WindowsFormsHost](#) esegue l'override dei metodi [MeasureOverride](#) e [ArrangeOverride](#).
2. Per determinare le dimensioni del controllo ospitato, il metodo [MeasureOverride](#) chiama il metodo [GetPreferredSize](#) del controllo ospitato con un vincolo convertito dal vincolo passato al metodo [MeasureOverride](#).
3. Il metodo [ArrangeOverride](#) tenta di impostare il controllo ospitato sul vincolo di dimensione specificato.
4. Se la proprietà [Size](#) del controllo ospitato corrisponde al vincolo specificato, il controllo contenuto viene ridimensionato in modo da essere dimensionato per il vincolo.

Se la proprietà [Size](#) non corrisponde al vincolo specificato, il controllo ospitato non supporta il ridimensionamento continuo. Ad esempio, il controllo [MonthCalendar](#) consente solo dimensioni discrete. Le dimensioni consentite per questo controllo sono costituite da numeri interi (che rappresentano il numero di mesi) per altezza e larghezza. In casi come questo, l'elemento [WindowsFormsHost](#) si comporta come segue:

- Se la proprietà [Size](#) restituisce una dimensione maggiore del vincolo specificato, l'elemento [WindowsFormsHost](#) ritaglia il controllo ospitato. L'altezza e la larghezza vengono gestite separatamente, quindi il controllo ospitato può essere ritagliato in entrambe le direzioni.
- Se la proprietà [Size](#) restituisce una dimensione inferiore a quella del vincolo specificato, [WindowsFormsHost](#) accetta questo valore di dimensione e restituisce il valore al sistema di layout WPF.

Vedere anche

- [ElementHost](#)
- [WindowsFormsHost](#)
- [Procedura dettagliata: Disposizione di controlli Windows Form in WPF](#)
- [Disposizione di controlli Windows Forms nell'esempio WPF](#)
- [Mapping di proprietà di Windows Form e WPF](#)
- [Migrazione e interoperabilità](#)

Controlli Windows Form e controlli WPF equivalenti

31/01/2020 • 2 minutes to read • [Edit Online](#)

Molti controlli Windows Forms hanno controlli WPF equivalenti, ma alcuni controlli Windows Forms non hanno equivalenti in WPF. In questo argomento vengono confrontati i tipi di controllo forniti dalle due tecnologie.

È sempre possibile usare l'interoperatività per ospitare Windows Forms controlli che non hanno equivalenti nelle applicazioni basate su WPF.

Nella tabella seguente vengono illustrati i controlli Windows Forms e i componenti con funzionalità di controllo WPF equivalenti.

CONTROLLO WINDOWS FORM	CONTROLLO EQUIVALENTE WPF	NOTE
BindingNavigator	Nessun controllo equivalente.	
BindingSource	CollectionViewSource	
Button	Button	
CheckBox	CheckBox	
CheckedListBox	ListBox con Composition.	
ColorDialog	Nessun controllo equivalente.	
ComboBox	ComboBox	ComboBox non supporta il completamento automatico.
ContextMenuStrip	ContextMenu	
DataGridView	DataGrid	
DateTimePicker	DatePicker	
DomainUpDown	TextBox e due controlli RepeatButton.	
ErrorProvider	Nessun controllo equivalente.	
FlowLayoutPanel	WrapPanel o StackPanel	
FolderBrowserDialog	Nessun controllo equivalente.	
FontDialog	Nessun controllo equivalente.	
Form	Window	Window non supporta le finestre figlio.
GroupBox	GroupBox	

CONTROLLO WINDOWS FORM	CONTROLLO EQUIVALENTE WPF	NOTE
HelpProvider	Nessun controllo equivalente.	Nessuna Guida sensibile al contesto. La guida "cosa è questo" è stata sostituita dalle descrizioni comandi.
HScrollBar	ScrollBar	Lo scorrimento è incorporato nei controlli contenitore.
ImageList	Nessun controllo equivalente.	
Label	Label	
LinkLabel	Nessun controllo equivalente.	È possibile usare la classe Hyperlink per ospitare collegamenti ipertestuali all'interno del contenuto del flusso.
ListBox	ListBox	
ListView	ListView	Il controllo ListView fornisce una visualizzazione dettagli di sola lettura.
MaskedTextBox	Nessun controllo equivalente.	
MenuStrip	Menu	Menu stile del controllo può approssimare il comportamento e l'aspetto della classe System.Windows.Forms.ToolStripProfessionalRenderer .
MonthCalendar	Calendar	
NotifyIcon	Nessun controllo equivalente.	
NumericUpDown	TextBox e due controlli RepeatButton .	
OpenFileDialog	OpenFileDialog	La classe OpenFileDialog è un wrapper WPF intorno al controllo Win32.
PageSetupDialog	Nessun controllo equivalente.	
Panel	Canvas	
PictureBox	Image	
PrintDialog	PrintDialog	
PrintDocument	Nessun controllo equivalente.	
PrintPreviewControl	DocumentViewer	
PrintPreviewDialog	Nessun controllo equivalente.	
ProgressBar	ProgressBar	

CONTROLLO WINDOWS FORM	CONTROLLO EQUIVALENTE WPF	NOTE
PropertyGrid	Nessun controllo equivalente.	
RadioButton	RadioButton	
RichTextBox	RichTextBox	
SaveFileDialog	SaveFileDialog	La classe SaveFileDialog è un wrapper WPF intorno al controllo Win32.
ScrollableControl	ScrollViewer	
SoundPlayer	MediaPlayer	
SplitContainer	GridSplitter	
StatusStrip	StatusBar	
TabControl	TabControl	
TableLayoutPanel	Grid	
TextBox	TextBox	
Timer	DispatcherTimer	
ToolStrip	ToolBar	
ToolStripContainer	ToolBar con Composition.	
ToolStripDropDown	ToolBar con Composition.	
ToolStripDropDownMenu	ToolBar con Composition.	
ToolStripPanel	ToolBar con Composition.	
ToolTip	ToolTip	
TrackBar	Slider	
TreeView	TreeView	
UserControl	UserControl	
VScrollBar	ScrollBar	Lo scorrimento è incorporato nei controlli contenitore.

CONTROLLO WINDOWS FORM	CONTROLLO EQUIVALENTE WPF	NOTE
WebBrowser	Frame, <code>System.Windows.Controls.WebBrowser</code>	<p>Il controllo Frame può ospitare pagine HTML.</p> <p>A partire da .NET Framework 3,5 SP1, il controllo System.Windows.Controls.WebBrowser può ospitare pagine HTML ed eseguire il backup anche del controllo Frame.</p>

Vedere anche

- [ElementHost](#)
- [WindowsFormsHost](#)
- [Progettazione WPF per sviluppatori Windows Forms](#)
- [Procedura dettagliata: hosting di controlli Windows Form in WPF](#)
- [Procedura dettaglia: hosting di un controllo WPF composito in Windows Form](#)
- [Migrazione e interoperabilità](#)

Mapping di proprietà di Windows Form e WPF

31/01/2020 • 11 minutes to read • [Edit Online](#)

Le tecnologie Windows Forms e WPF hanno due modelli di proprietà simili ma diversi. Il *mapping delle proprietà* supporta l'interoperatività tra le due architetture e offre le funzionalità seguenti:

- Consente di eseguire facilmente il mapping delle modifiche delle proprietà rilevanti nell'ambiente host al controllo o all'elemento ospitato.
- Fornisce la gestione predefinita per il mapping delle proprietà utilizzate più di frequente.
- Consente di rimuovere facilmente, eseguire l'override o estendere le proprietà predefinite.
- Garantisce che le modifiche ai valori delle proprietà nell'host vengano automaticamente rilevate e convertite nel controllo o nell'elemento ospitato.

NOTE

Gli eventi di modifica delle proprietà non vengono propagati al controllo di hosting o alla gerarchia degli elementi. La conversione di proprietà non viene eseguita se il valore locale di una proprietà non cambia a causa di impostazioni dirette, stili, ereditarietà, data binding o altri meccanismi che modificano il valore della proprietà.

Utilizzare la proprietà [PropertyMap](#) sull'elemento [WindowsFormsHost](#) e la proprietà [PropertyMap](#) sul controllo [ElementHost](#) per accedere al mapping delle proprietà.

Mapping di proprietà con l'elemento WindowsFormsHost

L'elemento [WindowsFormsHost](#) converte le proprietà di WPF predefinite negli equivalenti Windows Forms utilizzando la seguente tabella di conversione.

HOSTING DI WINDOWS PRESENTATION FOUNDATION	WINDOWS FORM	COMPORTAMENTO DI INTEROPERATIVITÀ

HOSTING DI WINDOWS PRESENTATION FOUNDATION	WINDOWS FORM	COMPORTAMENTO DI INTEROPERATIVITÀ
<p>Background <code>(System.Windows.Media.Brush)</code></p>	<p>BackColor <code>(System.Drawing.Color)</code></p>	<p>L'elemento <code>WindowsFormsHost</code> imposta la proprietà <code>BackColor</code> del controllo ospitato e la proprietà <code>BackgroundImage</code> del controllo ospitato. Il mapping viene eseguito utilizzando le regole seguenti:</p> <ul style="list-style-type: none"> -Se <code>Background</code> è un colore a tinta unita, viene convertito e utilizzato per impostare la proprietà <code>BackColor</code> del controllo ospitato. La proprietà <code>BackColor</code> non è impostata sul controllo ospitato, perché il controllo ospitato può ereditare il valore della proprietà <code>BackColor</code>. Nota: Il controllo ospitato non supporta la trasparenza. Qualsiasi colore assegnato a <code>BackColor</code> deve essere completamente opaco, con un valore alfa di 0xFF. -Se <code>Background</code> non è un colore a tinta unita, il controllo <code>WindowsFormsHost</code> crea una bitmap dalla proprietà <code>Background</code>. Il controllo <code>WindowsFormsHost</code> assegna questa bitmap alla proprietà <code>BackgroundImage</code> del controllo ospitato. Ciò fornisce un effetto simile alla trasparenza. Nota: È possibile eseguire l'override di questo comportamento oppure è possibile rimuovere il mapping della proprietà <code>Background</code>.
<p>Cursor</p>	<p>Cursor</p>	<p>Se il mapping predefinito non è stato riassegnato, <code>WindowsFormsHost</code> controllo attraversa la gerarchia predecessore fino a quando non trova un predecessore con il relativo set di proprietà <code>Cursor</code>. Questo valore viene convertito nel cursore di Windows Forms corrispondente più vicino.</p> <p>Se il mapping predefinito per la proprietà <code>ForceCursor</code> non è stato riassegnato, l'attraversamento viene arrestato sul primo predecessore con <code>ForceCursor</code> impostato su <code>true</code>.</p>
<p>FlowDirection <code>(System.Windows.FlowDirection)</code></p>	<p>RightToLeft <code>(System.Windows.Forms.RightToLeft)</code></p>	<p><code>LeftToRight</code> esegue il mapping a <code>No</code>.</p> <p><code>RightToLeft</code> esegue il mapping a <code>Yes</code>.</p> <p><code>Inherit</code> non è mappato.</p> <p><code>FlowDirection.RightToLeft</code> esegue il mapping a <code>RightToLeft.Yes</code>.</p>

HOSTING DI WINDOWS PRESENTATION FOUNDATION	WINDOWS FORM	COMPORTAMENTO DI INTEROPERATIVITÀ
FontStyle	Style sul System.Drawing.Font del controllo ospitato	Il set di proprietà di WPF viene convertito in un Font corrispondente. Quando una di queste proprietà viene modificata, viene creato un nuovo Font . Per Normal : Italic è disabilitato. Per Italic o Oblique : Italic è abilitato.
FontWeight	Style sul System.Drawing.Font del controllo ospitato	Il set di proprietà di WPF viene convertito in un Font corrispondente. Quando una di queste proprietà viene modificata, viene creato un nuovo Font . Per Black , Bold , DemiBold , ExtraBold , Heavy , Medium , SemiBold UltraBold : Bold è abilitato. Per ExtraLight , Light , Normal , Regular , Thin UltraLight : Bold è disabilitata.
FontFamily FontSize FontStretch FontStyle FontWeight	Font (System.Drawing.Font)	<p>Il set di proprietà di WPF viene convertito in un Fontcorrispondente. Quando una di queste proprietà viene modificata, viene creato un nuovo Font. Il controllo Windows Forms ospitato viene ridimensionato in base alle dimensioni del carattere.</p> <p>Le dimensioni del carattere in WPF sono espresse come un 90 ° sesto di un pollice e in Windows Forms come uno di un centimetro di pollice. La conversione corrispondente è:</p> <p>Dimensioni carattere Windows Forms = WPF dimensione carattere * 72,0/96,0.</p>
Foreground (System.Windows.Media.Brush)	ForeColor (System.Drawing.Color)	<p>Il mapping della proprietà Foreground viene eseguito utilizzando le regole seguenti:</p> <ul style="list-style-type: none"> -Se Foreground è un SolidColorBrush, utilizzare Color per ForeColor. -Se Foreground è un GradientBrush, utilizzare il colore del GradientStop con il valore di offset più basso per ForeColor. -Per qualsiasi altro tipo di Brush, lasciare ForeColor invariato. Ciò significa che viene utilizzato il valore predefinito.
IsEnabled	Enabled	Quando viene impostato IsEnabled , WindowsFormsHost elemento imposta la proprietà Enabled nel controllo ospitato.

HOSTING DI WINDOWS PRESENTATION FOUNDATION	WINDOWS FORM	COMPORTAMENTO DI INTEROPERATIVITÀ
Padding	Padding	<p>Tutti e quattro i valori della proprietà Padding nel controllo Windows Forms ospitato vengono impostati sullo stesso valore di Thickness.</p> <p>-I valori maggiori di MaxValue sono impostati su MaxValue. - I valori minori di MinValue sono impostati su MinValue.</p>
Visibility	Visible	<ul style="list-style-type: none"> - Visible viene eseguito il mapping a <code>Visible = true</code>. Il controllo Windows Forms ospitato è visibile. Non è consigliabile impostare in modo esplicito la proprietà Visible sul controllo ospitato su <code>false</code>. - Collapsed viene eseguito il mapping a <code>Visible = true</code> o <code>false</code>. Il controllo Windows Forms ospitato non viene disegnato e la relativa area è compressa. - Hidden: il controllo Windows Forms ospitato occupa spazio nel layout, ma non è visibile. In questo caso, la proprietà Visible è impostata su <code>true</code>. Non è consigliabile impostare in modo esplicito la proprietà Visible sul controllo ospitato su <code>false</code>.

Le proprietà associate sugli elementi del contenitore sono completamente supportate dall'elemento [WindowsFormsHost](#).

Per ulteriori informazioni, vedere [procedura dettagliata: mapping di proprietà tramite l'elemento WindowsFormsHost](#).

Aggiornamenti alle proprietà padre

Le modifiche apportate alla maggior parte delle proprietà padre generano notifiche per il controllo figlio ospitato. Nell'elenco seguente vengono descritte le proprietà che non provocano notifiche quando cambiano i valori.

- [Background](#)
- [Cursor](#)
- [ForceCursor](#)
- [Visibility](#)

Se ad esempio si modifica il valore della proprietà [Background](#) dell'elemento [WindowsFormsHost](#), la proprietà [BackColor](#) del controllo ospitato non viene modificata.

Mapping di proprietà con il controllo ElementHost

Le proprietà seguenti forniscono una notifica di modifica incorporata. Non chiamare il metodo [OnPropertyChanged](#) quando si esegue il mapping di queste proprietà:

- Ridimensiona automaticamente
- BackColor
- BackgroundImage
- BackgroundImageLayout
- BindingContext
- CausesValidation
- ContextMenu
- ContextMenuStrip
- Cursore
- Dock
- Enabled
- Carattere
- ForeColor
- Percorso
- Margine
- Spaziatura interna
- Padre
- Region
- RightToLeft
- Dimensioni
- TabIndex
- TabStop
- Testo
- Visibile

Il controllo [ElementHost](#) converte le proprietà di Windows Forms predefinite negli equivalenti WPF utilizzando la seguente tabella di conversione.

Per ulteriori informazioni, vedere [procedura dettagliata: mapping delle proprietà tramite il controllo ElementHost](#).

HOSTING DI WINDOWS FORMS	WINDOWS PRESENTATION FOUNDATION	COMPORTAMENTO DI INTEROPERATIVITÀ
<p>BackColor <code>(System.Drawing.Color)</code></p>	<p>Background <code>(System.Windows.Media.Brush)</code> nell'elemento Hosted</p>	<p>L'impostazione di questa proprietà impone un ridisegno con un ImageBrush. Se la proprietà BackColorTransparent è impostata su <code>false</code> (valore predefinito), questo ImageBrush si basa sull'aspetto del controllo ElementHost, incluse le proprietà BackColor, BackgroundImage, BackgroundImageLayout e tutti i gestori di disegno collegati.</p> <p>Se la proprietà BackColorTransparent è impostata su <code>true</code>, il ImageBrush si basa sull'aspetto dell'elemento padre del controllo ElementHost, incluse le proprietà BackColor, BackgroundImage, BackgroundImageLayout e tutti i gestori di disegno associati del padre.</p>
<p>BackgroundImage <code>(System.Drawing.Image)</code></p>	<p>Background <code>(System.Windows.Media.Brush)</code> nell'elemento Hosted</p>	<p>L'impostazione di questa proprietà determina lo stesso comportamento descritto per il mapping del BackColor.</p>
<p>BackgroundImageLayout</p>	<p>Background <code>(System.Windows.Media.Brush)</code> nell'elemento Hosted</p>	<p>L'impostazione di questa proprietà determina lo stesso comportamento descritto per il mapping del BackColor.</p>
<p>Cursor <code>(System.Windows.Forms.Cursor)</code></p>	<p>Cursor <code>(System.Windows.Input.Cursor)</code></p>	<p>Il cursore Windows Forms standard viene convertito nel cursore WPF standard corrispondente. Se il Windows Forms non è un cursore standard, viene assegnato il valore predefinito.</p>
<p>Enabled</p>	<p>.IsEnabled</p>	<p>Quando viene impostato Enabled, il controllo ElementHost imposta la proprietà .IsEnabled sull'elemento Hosted.</p>
<p>Font <code>(System.Drawing.Font)</code></p>	<p>FontFamily FontSize FontStretch FontStyle FontWeight</p>	<p>Il valore Font viene convertito in un set di proprietà del tipo di carattere WPF corrispondente.</p>
<p>Bold</p>	<p>FontWeight sull'elemento Hosted</p>	<p>Se Bold è <code>true</code>, FontWeight viene impostato su Bold.</p> <p>Se Bold è <code>false</code>, FontWeight viene impostato su Normal.</p>

HOSTING DI WINDOWS FORMS	WINDOWS PRESENTATION FOUNDATION	COMPORTAMENTO DI INTEROPERATIVITÀ
Italic	FontStyle sull'elemento Hosted	Se Italic è <code>true</code> , FontStyle viene impostato su Italic . Se Italic è <code>false</code> , FontStyle viene impostato su Normal .
Strikeout	TextDecorations sull'elemento Hosted	Si applica solo quando si ospita un controllo TextBlock .
Underline	TextDecorations sull'elemento Hosted	Si applica solo quando si ospita un controllo TextBlock .
RightToLeft (System.Windows.Forms.RightToLeft)	FlowDirection (FlowDirection)	No esegue il mapping a LeftToRight . Yes esegue il mapping a RightToLeft .
Visible	Visibility	Il controllo ElementHost imposta la proprietà Visibility sull'elemento Hosted utilizzando le regole seguenti: - Visible = <code>true</code> esegue il mapping a Visible . - Visible = <code>false</code> esegue il mapping a Hidden .

Vedere anche

- [ElementHost](#)
- [WindowsFormsHost](#)
- [Interoperatività di WPF e Win32](#)
- [Interoperatività di WPF e Windows Form](#)
- [Procedura dettagliata: Mapping di proprietà tramite l'elemento WindowsFormsHost](#)
- [Procedura dettagliata: Mapping delle proprietà tramite il controllo ElementHost](#)

Risoluzione dei problemi relativi ad applicazioni ibride

31/01/2020 • 13 minutes to read • [Edit Online](#)

In questo argomento vengono elencati alcuni problemi comuni che possono verificarsi durante la creazione di applicazioni ibride che utilizzano le tecnologie WPF e Windows Forms.

Sovrapposizione di controlli

I controlli potrebbero sovrapporsi in maniera imprevista. Windows Forms utilizza un HWND separato per ogni controllo. WPF usa un HWND per tutto il contenuto in una pagina. Questa differenza di implementazione provoca comportamenti di sovrapposizione imprevisti.

Un controllo Windows Forms ospitato in WPF viene sempre visualizzato sopra il contenuto del WPF.

WPF contenuto ospitato in un controllo [ElementHost](#) viene visualizzato in corrispondenza dell'ordine z del controllo [ElementHost](#). È possibile sovrapporre [ElementHost](#) controlli, ma il contenuto della WPF ospitata non combina né interagisce.

Child Property

Le classi [WindowsFormsHost](#) e [ElementHost](#) possono ospitare solo un singolo controllo o elemento figlio. Per ospitarne più di uno, è necessario usare un contenitore come contenuto figlio. Ad esempio, è possibile aggiungere Windows Forms pulsante e i controlli casella di controllo a un controllo [System.Windows.Forms.Panel](#) e quindi assegnare il pannello alla proprietà [Child](#) di un controllo [WindowsFormsHost](#). Tuttavia, non è possibile aggiungere separatamente i controlli Button e check box allo stesso controllo [WindowsFormsHost](#).

Ridimensionamento

WPF e Windows Forms hanno modelli di scalabilità diversi. Alcune WPF le trasformazioni di ridimensionamento sono significative per i controlli Windows Forms, mentre altre non lo sono. Ad esempio, il ridimensionamento di un controllo Windows Forms su 0 funziona, ma se si prova a ridimensionare lo stesso controllo a un valore diverso da zero, le dimensioni del controllo restano pari a 0. Per altre informazioni, vedere [Considerazioni sul layout per l'elemento WindowsFormsHost](#).

Adapter

È possibile che si verifichino confusioni quando si utilizzano le classi [WindowsFormsHost](#) e [ElementHost](#), perché includono un contenitore nascosto. Entrambe le classi [WindowsFormsHost](#) e [ElementHost](#) hanno un contenitore nascosto, denominato *Adapter*, che usano per ospitare il contenuto. Per l'elemento [WindowsFormsHost](#), l'adapter deriva dalla classe [System.Windows.Forms.ContainerControl](#). Per il controllo [ElementHost](#), l'adapter deriva dall'elemento [DockPanel](#). I riferimenti all'adattatore presenti in altri argomenti sull'interoperabilità alludono a tale contenitore.

Nidificazione

Annidamento di un elemento [WindowsFormsHost](#) all'interno di un controllo [ElementHost](#) non è supportato. Anche l'annidamento di un controllo [ElementHost](#) all'interno di un elemento [WindowsFormsHost](#) non è supportato.

Stato attivo

Lo stato attivo funziona in modo diverso in WPF e Windows Forms, il che significa che possono verificarsi problemi di messa a fuoco in un'applicazione ibrida. Ad esempio, se lo stato attivo si trova all'interno di un elemento [WindowsFormsHost](#) e si riduce a icona e si ripristina la pagina o si visualizza una finestra di dialogo modale, lo stato attivo all'interno dell'elemento [WindowsFormsHost](#) può andare perduto. L'elemento [WindowsFormsHost](#) ha ancora lo stato attivo, ma il controllo all'interno di esso non può.

Lo stato attivo influisce anche sulla convalida dei dati. La convalida funziona in un elemento [WindowsFormsHost](#), ma non funziona quando si esce dalla tabulazione dell'elemento [WindowsFormsHost](#) o tra due elementi [WindowsFormsHost](#) diversi.

Mapping proprietà

Alcuni mapping di proprietà richiedono un'interpretazione completa per colmare implementazioni non simili tra le tecnologie WPF e Windows Forms. I mapping di proprietà consentono al codice di rispondere alle modifiche apportate ai tipi di carattere, ai colori e ad altre proprietà. In generale, i mapping di proprietà funzionano rimanendo in ascolto di eventi *PropertyChanged* o chiamate *OnPropertyChanged* e impostando le proprietà appropriate sul controllo figlio o sul relativo adattatore. Per altre informazioni, vedere [Mapping di proprietà di Windows Form e WPF](#).

Proprietà relative al layout nel contenuto ospitato

Quando viene assegnata la proprietà [WindowsFormsHost.Child](#) o [ElementHost.Child](#), vengono impostate automaticamente diverse proprietà correlate al layout nel contenuto ospitato. La modifica di queste proprietà di contenuto può determinare comportamenti di layout imprevisti.

Il contenuto ospitato è ancorato in modo da riempire il [WindowsFormsHost](#) e [ElementHost](#) padre. Per abilitare questo comportamento di riempimento, durante l'impostazione della proprietà figlio vengono impostate varie proprietà. Nella tabella seguente sono elencate le proprietà di contenuto impostate dalle classi [ElementHost](#) e [WindowsFormsHost](#).

CLASSE HOST	PROPRIETÀ DI CONTENUTO
ElementHost	Height Width Margin VerticalAlignment HorizontalAlignment
WindowsFormsHost	Margin Dock AutoSize Location MaximumSize

Non impostare queste proprietà direttamente sul contenuto ospitato. Per altre informazioni, vedere [Considerazioni sul layout per l'elemento WindowsFormsHost](#).

Applicazioni di navigazione

Le applicazioni di navigazione potrebbero non mantenere lo stato dell'utente. L'elemento [WindowsFormsHost](#) ricrea i controlli quando viene utilizzato in un'applicazione di navigazione. La ricreazione di controlli figlio si verifica quando l'utente si sposta dalla pagina che ospita l'elemento [WindowsFormsHost](#) e quindi torna al suo interno. Qualsiasi contenuto digitato dall'utente andrà perduto.

Interoperabilità del ciclo di messaggi

Quando si utilizzano Windows Forms cicli di messaggi, è possibile che i messaggi non vengano elaborati come previsto. Il metodo [EnableWindowsFormsInterop](#) viene chiamato dal costruttore di [WindowsFormsHost](#). Questo metodo aggiunge un filtro messaggi al ciclo di messaggi WPF. Questo filtro chiama il metodo [Control.PreProcessMessage](#) se una [System.Windows.Forms.Control](#) è la destinazione del messaggio e converte/invia il messaggio.

Se si visualizza un [Window](#) in un ciclo di messaggi Windows Forms con [Application.Run](#), non è possibile digitare niente a meno che non si chiami il metodo [EnableModelessKeyboardInterop](#). Il metodo [EnableModelessKeyboardInterop](#) accetta una [Window](#) e aggiunge un [System.Windows.Forms.IMessageFilter](#), che reindirizza i messaggi relativi alla chiave al ciclo di messaggi WPF. Per altre informazioni, vedere [Architettura di input per l'interoperabilità tra Windows Form e WPF](#).

Opacità e sovrapposizione

La classe [HwndHost](#) non supporta la sovrapposizione. Ciò significa che l'impostazione della proprietà [Opacity](#) sull'elemento [WindowsFormsHost](#) non ha alcun effetto e che non si verificherà alcuna fusione con altre finestre WPF che hanno [AllowsTransparency](#) impostato su `true`.

Dispose

L'eliminazione non corretta delle classi può determinare una perdita di risorse. Nelle applicazioni ibride verificare che le classi [WindowsFormsHost](#) e [ElementHost](#) vengano eliminate o che si verifichino perdite di risorse. Windows Forms elimina i controlli [ElementHost](#) quando il padre [Form](#) non modale viene chiuso. WPF Elimina gli elementi [WindowsFormsHost](#) quando l'applicazione viene arrestata. È possibile mostrare un elemento [WindowsFormsHost](#) in un [Window](#) in un ciclo di messaggi di Windows Forms. In questo caso, è possibile che il codice non riceva notifica della chiusura dell'applicazione.

Abilitazione degli stili di visualizzazione

Gli stili di visualizzazione di Microsoft Windows XP in un controllo Windows Forms potrebbero non essere abilitati. Il metodo [Application.EnableVisualStyles](#) viene chiamato nel modello per un Windows Forms Application. Sebbene questo metodo non venga chiamato per impostazione predefinita, se si utilizza Visual Studio per creare un progetto, si otterranno gli stili di visualizzazione di Microsoft Windows XP per i controlli, se è disponibile la versione 6,0 di Comctl32.dll. Prima di creare handle nel thread, è necessario chiamare il metodo [EnableVisualStyles](#). Per altre informazioni, vedere [Procedura: Abilitare stili di visualizzazione in un'applicazione ibrida](#).

Controlli con licenza

I controlli di Windows Forms con licenza che visualizzano le informazioni sulle licenze in una finestra di messaggio all'utente possono causare un comportamento imprevisto per un'applicazione ibrida. Alcuni controlli con licenza visualizzano una finestra di dialogo in risposta alla creazione di handle. Ad esempio, un controllo con licenza potrebbe informare l'utente che è richiesta una licenza oppure che sono consentiti ancora tre usi del controllo a scopo di valutazione.

L'elemento [WindowsFormsHost](#) deriva dalla classe [HwndHost](#) e l'handle del controllo figlio viene creato all'interno del [BuildWindowCore](#) metodo. La classe [HwndHost](#) non consente l'elaborazione dei messaggi nel metodo [BuildWindowCore](#), ma la visualizzazione di una finestra di dialogo comporta l'invio dei messaggi. Per abilitare questo scenario di gestione delle licenze, chiamare il metodo [Control.CreateControl](#) sul controllo prima di assegnarlo come figlio dell'elemento di [WindowsFormsHost](#).

WPF Designer

È possibile progettare il contenuto WPF utilizzando WPF Designer per Visual Studio. Nelle sezioni seguenti sono elencati alcuni problemi comuni che possono verificarsi durante la creazione di applicazioni ibride con WPF Designer.

Proprietà BackColorTransparent ignorata in fase di progettazione

Il [BackColorTransparent](#) proprietà potrebbe non funzionare come previsto in fase di progettazione.

Se un controllo WPF non si trova in un elemento padre visibile, il runtime WPF ignorerà il valore [BackColorTransparent](#). Il motivo per cui [BackColorTransparent](#) possibile ignorare è che [ElementHost](#) oggetto viene creato in un [AppDomain](#) separato. Tuttavia, quando si esegue l'applicazione, [BackColorTransparent](#) funziona come previsto.

Visualizzazione dell'elenco errori della fase di progettazione quando viene eliminata la cartella obj

Se la cartella obj viene eliminata, viene visualizzato l'elenco errori della fase di progettazione.

Quando si progetta usando [ElementHost](#), il Progettazione Windows Form usa i file generati nella cartella debug o versione all'interno della cartella obj del progetto. Se si eliminano questi file, viene visualizzato l'elenco errori della fase di progettazione. Per risolvere questo problema, ricompilare il progetto. Per altre informazioni, vedere [Errori in fase di progettazione in Progettazione Windows Form](#).

ElementHost e IME

I controlli WPF ospitati in un [ElementHost](#) attualmente non supportano la proprietà [ImeMode](#). Le modifiche apportate al [ImeMode](#) verranno ignorate dai controlli ospitati.

Vedere anche

- [ElementHost](#)
- [WindowsFormsHost](#)
- [Interoperabilità in WPF Designer](#)
- [Architettura di input per l'interoperabilità tra Windows Form e WPF](#)
- [Procedura: Abilitare stili di visualizzazione in un'applicazione ibrida](#)
- [Considerazioni sul layout per l'elemento WindowsFormsHost](#)
- [Mapping di proprietà di Windows Form e WPF](#)
- [Errori in fase di progettazione in Progettazione Windows Form](#)
- [Migrazione e interoperabilità](#)

Procedura dettagliata: hosting di controlli Windows Form in WPF

12/02/2020 • 3 minutes to read • [Edit Online](#)

WPF offre numerosi controlli con un insieme di funzionalità completo. Tuttavia, a volte può essere opportuno usare Windows Forms controlli nelle pagine WPF. Ad esempio, è possibile che si disponga di un investimento sostanziale nei controlli di Windows Forms esistenti o che si disponga di un controllo Windows Forms che fornisce funzionalità univoca.

In questa procedura dettagliata viene illustrato come ospitare un controllo `System.Windows.Forms.MaskedTextBox` di Windows Forms in una pagina di WPF utilizzando il codice.

Per un listato di codice completo delle attività illustrate in questa procedura dettagliata, vedere la pagina relativa all'[hosting di un controllo Windows Forms in WPF di esempio](#).

Prerequisites

Per completare la procedura dettagliata, è necessario Visual Studio.

Hosting del controllo Windows Forms

Per ospitare il controllo MaskedTextBox

1. Creare un progetto di applicazione WPF denominato `HostingWfInWpf`.
2. Aggiungere riferimenti agli assembly indicati di seguito.
 - `WindowsFormsIntegration`
 - `System.Windows.Forms`
3. Aprire `MainWindow.xaml` in WPF Designer.
4. Denominare l'elemento `Grid` `grid1`.

```
<Grid Name="grid1">  
  </Grid>
```

5. Nella visualizzazione progettazione o nella visualizzazione XAML Selezionare l'elemento `Window`.
6. Nella Finestra Proprietà fare clic sulla scheda **eventi**.
7. Fare doppio clic sull'evento `Loaded`.
8. Inserire il codice seguente per gestire l'evento `Loaded`.

```

private void Window_Loaded(object sender, RoutedEventArgs e)
{
    // Create the interop host control.
    System.Windows.Forms.Integration.WindowsFormsHost host =
        new System.Windows.Forms.Integration.WindowsFormsHost();

    // Create the MaskedTextBox control.
    MaskedTextBox mtbDate = new MaskedTextBox("00/00/0000");

    // Assign the MaskedTextBox control as the host control's child.
    host.Child = mtbDate;

    // Add the interop host control to the Grid
    // control's collection of child controls.
    this.grid1.Children.Add(host);
}

```

```

Private Sub Window_Loaded(ByVal sender As Object, ByVal e As RoutedEventArgs)
    ' Create the interop host control.
    Dim host As New System.Windows.Forms.Integration.WindowsFormsHost()

    ' Create the MaskedTextBox control.
    Dim mtbDate As New MaskedTextBox("00/00/0000")

    ' Assign the MaskedTextBox control as the host control's child.
    host.Child = mtbDate

    ' Add the interop host control to the Grid
    ' control's collection of child controls.
    Me.grid1.Children.Add(host)

End Sub

```

9. Nella parte superiore del file aggiungere la seguente `Imports` o `using` istruzione.

```
using System.Windows.Forms;
```

```
Imports System.Windows.Forms
```

10. Premere **F5** per compilare ed eseguire l'applicazione.

Vedere anche

- [ElementHost](#)
- [WindowsFormsHost](#)
- [Progettare XAML in Visual Studio](#)
- [Procedura dettagliata: Hosting di controlli Windows Form in WPF tramite XAML](#)
- [Procedura dettagliata: Hosting di controlli Windows Form compositi in WPF](#)
- [Procedura dettaglia: hosting di un controllo WPF composito in Windows Form](#)
- [Controlli Windows Form e controlli WPF equivalenti](#)
- [Esempio di hosting di un controllo Windows Forms in WPF](#)

Procedura dettagliata: hosting di controlli Windows Form in WPF tramite XAML

10/02/2020 • 2 minutes to read • [Edit Online](#)

WPF offre numerosi controlli con un insieme di funzionalità completo. Tuttavia, a volte può essere opportuno usare Windows Forms controlli nelle pagine WPF. Ad esempio, è possibile che si disponga di un investimento sostanziale nei controlli di Windows Forms esistenti o che si disponga di un controllo Windows Forms che fornisce funzionalità univoca.

In questa procedura dettagliata viene illustrato come ospitare un controllo [System.Windows.Forms.MaskedTextBox](#) di Windows Forms in una pagina WPF utilizzando XAML.

Per un listato di codice completo delle attività illustrate in questa procedura dettagliata, vedere [Hosting di un controllo Windows Forms in WPF usando l'esempio XAML](#).

Prerequisites

Per completare la procedura dettagliata, è necessario Visual Studio.

Hosting del controllo Windows Forms

Per ospitare il controllo MaskedTextBox

1. Creare un progetto di applicazione WPF denominato `HostingWfInWpfWithXaml`.
2. Aggiungere riferimenti agli assembly indicati di seguito.
 - WindowsFormsIntegration
 - System.Windows.Forms
3. Aprire MainWindow.xaml in WPF Designer.
4. Nell'elemento [Window](#) aggiungere il mapping dello spazio dei nomi seguente. Il mapping dello spazio dei nomi `wf` stabilisce un riferimento all'assembly che contiene il controllo Windows Forms.

```
xmlns:wf="clr-namespace:System.Windows.Forms;assembly=System.Windows.Forms"
```

5. Nell'elemento [Grid](#) aggiungere il codice XAML seguente.

Il controllo [MaskedTextBox](#) viene creato come elemento figlio del controllo [WindowsFormsHost](#).

```
<Grid>
    <WindowsFormsHost>
        <wf:MaskedTextBox x:Name="mtbDate" Mask="00/00/0000"/>
    </WindowsFormsHost>
</Grid>
```

6. Premere F5 per compilare ed eseguire l'applicazione.

Vedere anche

- [ElementHost](#)
- [WindowsFormsHost](#)
- [Progettare XAML in Visual Studio](#)
- [Procedura dettagliata: hosting di controlli Windows Form in WPF](#)
- [Procedura dettagliata: Hosting di controlli Windows Form compositi in WPF](#)
- [Procedura dettaglia: hosting di un controllo WPF composito in Windows Form](#)
- [Controlli Windows Form e controlli WPF equivalenti](#)
- [Hosting di un controllo Windows Forms in WPF usando l'esempio XAML](#)

Procedura dettagliata: hosting di controlli Windows Form compositi in WPF

31/01/2020 • 23 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) fornisce un ambiente completo per la creazione di applicazioni. Tuttavia, quando si ha un investimento sostanziale nel codice Windows Forms, può essere più efficace riutilizzare almeno parte del codice nell'applicazione WPF anziché riscriverla da zero. Lo scenario più comune è quello in cui si dispone di controlli di Windows Forms esistenti. In alcuni casi, è possibile che non si abbia accesso al codice sorgente di questi controlli. WPF fornisce una procedura semplice per l'hosting di tali controlli in un'applicazione WPF. Ad esempio, è possibile usare WPF per la maggior parte della programmazione durante l'hosting dei controlli [DataGridView](#) specializzati.

Questa procedura dettagliata illustra un'applicazione che ospita un controllo Windows Forms composito per eseguire l'immissione di dati in un'applicazione WPF. Il controllo composito è compresso in una DLL. Questa procedura generale può essere estesa ad applicazioni e controlli più complessi. Questa procedura dettagliata è progettata per essere pressoché identica per quanto riguarda aspetto e funzionalità per [la procedura dettagliata: hosting di un controllo composito WPF in Windows Forms](#). La principale differenza è che lo scenario di hosting è invertito.

La procedura guidata è suddivisa in due sezioni. Nella prima sezione viene brevemente descritta l'implementazione del controllo Windows Forms composito. La seconda sezione illustra in dettaglio come ospitare il controllo composito in un'applicazione WPF, ricevere eventi dal controllo e accedere ad alcune delle proprietà del controllo.

Le attività illustrate nella procedura dettagliata sono le seguenti:

- Implementazione del controllo Windows Forms composito.
- Implementazione dell'applicazione host WPF.

Per un listato di codice completo delle attività illustrate in questa procedura dettagliata, vedere la pagina relativa all'[hosting di un controllo Windows Forms composito in WPF di esempio](#).

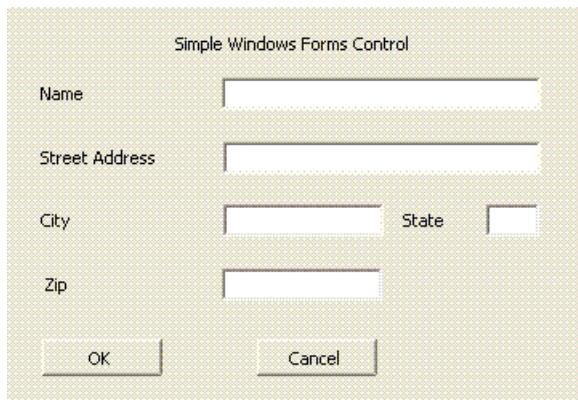
Prerequisiti

Per completare la procedura dettagliata, è necessario Visual Studio.

Implementazione del controllo Windows Forms composito

Il Windows Forms controllo composito usato in questo esempio è un semplice form di immissione dati. Questo form accetta nome e indirizzo dell'utente e quindi usa un evento personalizzato per restituire le informazioni all'host. La figura seguente mostra il controllo sottoposto a rendering.

Nell'immagine seguente viene illustrato un controllo Windows Forms composito:



Creazione del progetto

Per avviare il progetto:

1. Avviare Visual Studio e aprire la finestra di dialogo **nuovo progetto**.
2. Nella categoria finestra selezionare il modello **libreria di controlli Windows Forms**.
3. Assegnare il nome `MyControls` al nuovo progetto.
4. Per il percorso specificare una cartella di primo livello con un nome appropriato, ad esempio `WpfHostingWindowsFormsControl`. Successivamente, si immetterà l'applicazione host in questa cartella.
5. Fare clic su **OK** per creare il progetto. Il progetto predefinito contiene un singolo controllo denominato `UserControl1`.
6. In Esplora soluzioni rinominare `UserControl1` `MyControl1`.

Il progetto dovrebbe includere riferimenti alle DLL di sistema seguenti. Se alcune di queste DLL non sono incluse per impostazione predefinita, aggiungerle al progetto.

- System
- System.Data
- System.Drawing
- System.Windows.Forms
- System.Xml

Aggiunta di controlli al form

Per aggiungere controlli al form:

- Aprire `MyControl1` nella finestra di progettazione.

Aggiungere cinque controlli **Label** e i controlli **TextBox** corrispondenti, ridimensionati e disposti come sono nell'illustrazione precedente, nel form. Nell'esempio, i controlli **TextBox** sono denominati:

- `txtName`
- `txtAddress`
- `txtCity`
- `txtState`
- `txtZip`

Aggiungere due controlli **Button** con etichetta **OK** e **Annulla**. Nell'esempio i nomi dei pulsanti sono `btnOK` e

`btnCancel` rispettivamente.

Implementazione del codice di supporto

Aprire il form nella visualizzazione codice. Il controllo restituisce i dati raccolti al relativo host generando l'evento `OnButtonClick` personalizzato. I dati sono contenuti nell'oggetto argomento dell'evento. Il codice seguente mostra la dichiarazione di evento e delegato.

Aggiungere il codice seguente alla classe `MyControl1`.

```
public delegate void MyControlEventHandler(object sender, MyControlEventArgs args);
public event MyControlEventHandler OnButtonClick;
```

```
Public Delegate Sub MyControlEventHandler(ByVal sender As Object, ByVal args As MyControlEventArgs)
Public Event OnButtonClick As MyControlEventHandler
```

La classe `MyControlEventArgs` contiene le informazioni da restituire all'host.

Aggiungere la classe seguente al form.

```

public class MyControlEvents : EventArgs
{
    private string _Name;
    private string _StreetAddress;
    private string _City;
    private string _State;
    private string _Zip;
    private bool _IsOK;

    public MyControlEvents(bool result,
                          string name,
                          string address,
                          string city,
                          string state,
                          string zip)
    {
        _IsOK = result;
        _Name = name;
        _StreetAddress = address;
        _City = city;
        _State = state;
        _Zip = zip;
    }

    public string MyName
    {
        get { return _Name; }
        set { _Name = value; }
    }
    public string MyStreetAddress
    {
        get { return _StreetAddress; }
        set { _StreetAddress = value; }
    }
    public string MyCity
    {
        get { return _City; }
        set { _City = value; }
    }
    public string MyState
    {
        get { return _State; }
        set { _State = value; }
    }
    public string MyZip
    {
        get { return _Zip; }
        set { _Zip = value; }
    }
    public bool IsOK
    {
        get { return _IsOK; }
        set { _IsOK = value; }
    }
}

```

```

Public Class MyControlEvents
    Inherits EventArgs
    Private _Name As String
    Private _StreetAddress As String
    Private _City As String
    Private _State As String
    Private _Zip As String
    Private _IsOK As Boolean

```

```

    Public Sub New(ByVal result As Boolean, ByVal name As String, ByVal address As String, ByVal city As
String, ByVal state As String, ByVal zip As String)
        _IsOK = result
        _Name = name
        _StreetAddress = address
        _City = city
        _State = state
        _Zip = zip

    End Sub

    Public Property MyName() As String
        Get
            Return _Name
        End Get
        Set
            _Name = value
        End Set
    End Property

    Public Property MyStreetAddress() As String
        Get
            Return _StreetAddress
        End Get
        Set
            _StreetAddress = value
        End Set
    End Property

    Public Property MyCity() As String
        Get
            Return _City
        End Get
        Set
            _City = value
        End Set
    End Property

    Public Property MyState() As String
        Get
            Return _State
        End Get
        Set
            _State = value
        End Set
    End Property

    Public Property MyZip() As String
        Get
            Return _Zip
        End Get
        Set
            _Zip = value
        End Set
    End Property

    Public Property IsOK() As Boolean
        Get
            Return _IsOK
        End Get
        Set
            _IsOK = value
        End Set
    End Property
End Class

```

Quando l'utente fa clic sul pulsante **OK** o **Annulla**, i gestori eventi **Click** creano un oggetto **MyControlEventArgs** che contiene i dati e genera l'evento di **OnButtonClick**. L'unica differenza tra i due gestori è la proprietà **IsOK** dell'argomento dell'evento. Questa proprietà consente all'host di determinare su quale pulsante è stato fatto clic. È impostato su **true** per il pulsante **OK** e **false** per il pulsante **Annulla**. Il codice seguente illustra i gestori dei due pulsanti.

Aggiungere il codice seguente alla classe **MyControl1**.

```
private void btnOK_Click(object sender, System.EventArgs e)
{
    MyControlEventArgs retvals = new MyControlEventArgs(true,
        txtName.Text,
        txtAddress.Text,
        txtCity.Text,
        txtState.Text,
        txtZip.Text);
    OnButtonClick(this, retvals);
}

private void btnCancel_Click(object sender, System.EventArgs e)
{
    MyControlEventArgs retvals = new MyControlEventArgs(false,
        txtName.Text,
        txtAddress.Text,
        txtCity.Text,
        txtState.Text,
        txtZip.Text);
    OnButtonClick(this, retvals);
}
```

```
Private Sub btnOK_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles btnOK.Click
    Dim retvals As New MyControlEventArgs(True, txtName.Text, txtAddress.Text, txtCity.Text, txtState.Text,
    txtZip.Text)
    RaiseEvent OnButtonClick(Me, retvals)
End Sub

Private Sub btnCancel_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles btnCancel.Click
    Dim retvals As New MyControlEventArgs(False, txtName.Text, txtAddress.Text, txtCity.Text, txtState.Text,
    txtZip.Text)
    RaiseEvent OnButtonClick(Me, retvals)
End Sub
```

Assegnazione di un nome sicuro all'assembly e compilazione dell'assembly

Per fare in modo che a questo assembly venga fatto riferimento da un'applicazione WPF, è necessario che disponga di un nome sicuro. Per creare un nome sicuro, creare un file di chiave con sn. exe e aggiungerlo al progetto.

1. Aprire il prompt dei comandi di Visual Studio. A tale scopo, fare clic sul menu **Start**, quindi selezionare **tutti i programmi/Microsoft Visual Studio 2010/strumenti di Visual Studio/prompt dei comandi di Visual Studio**. Verrà avviata una finestra della console con variabili di ambiente personalizzate.
2. Al prompt dei comandi usare il comando **cd** per passare alla cartella del progetto.
3. Generare un file di chiave denominato MyControls.snk eseguendo il comando seguente.

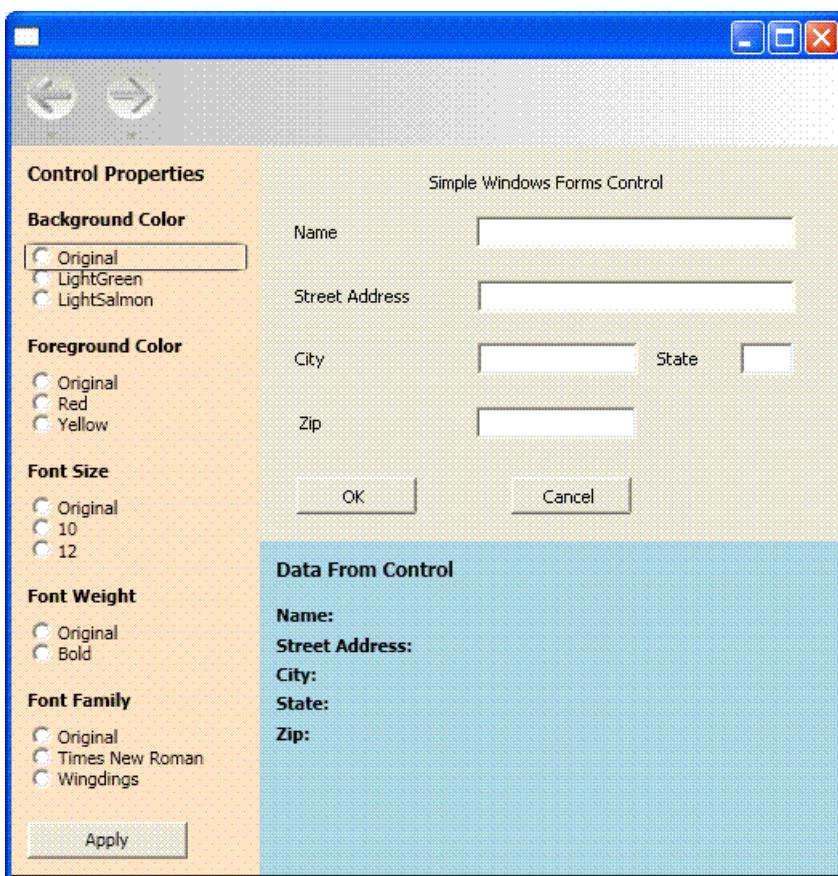
```
Sn.exe -k MyControls.snk
```

4. Per includere il file di chiave nel progetto, fare clic con il pulsante destro del mouse sul nome del progetto in Esplora soluzioni e quindi scegliere **Proprietà**. In Creazione progetti fare clic sulla scheda **firma**, selezionare la casella di controllo **Firma assembly** e quindi passare al file di chiave.
5. Compila la soluzione. La compilazione genererà una DLL denominata MyControls.dll.

Implementazione dell'applicazione host WPF

L'applicazione host WPF usa il controllo [WindowsFormsHost](#) per ospitare [MyControl1](#). L'applicazione gestisce l'evento [OnButtonClick](#) per ricevere i dati dal controllo. Dispone inoltre di una raccolta di pulsanti di opzione che consentono di modificare alcune delle proprietà del controllo dall'applicazione WPF. La figura seguente illustra l'applicazione finita.

Nell'immagine seguente viene illustrata l'applicazione completa, incluso il controllo incorporato nell'applicazione WPF:



Creazione del progetto

Per avviare il progetto:

1. Aprire Visual Studio e selezionare **nuovo progetto**.
2. Nella categoria finestra selezionare il modello **applicazione WPF**.
3. Assegnare il nome [WpfHost](#) al nuovo progetto.
4. Come percorso, specificare la stessa cartella di livello superiore che contiene il progetto MyControls.
5. Fare clic su **OK** per creare il progetto.

È anche necessario aggiungere riferimenti alla DLL che contiene [MyControl1](#) e altri assembly.

1. Fare clic con il pulsante destro del mouse sul nome del progetto in Esplora soluzioni e scegliere **Aggiungi riferimento**.
2. Fare clic sulla scheda **Sfoglia** e selezionare la cartella che contiene i controlli. dll. In questa procedura dettagliata la cartella è MyControls\bin\Debug.
3. Selezionare controllo. dll e quindi fare clic su **OK**.
4. Aggiungere un riferimento all'assembly WindowsFormsIntegration, denominato WindowsFormsIntegration.dll.

Implementazione del layout di base

Il interfaccia utente dell'applicazione host viene implementato nel file MainWindow. XAML. Questo file contiene Extensible Application Markup Language (XAML) markup che definisce il layout e ospita il controllo Windows Forms. L'applicazione è suddivisa in tre aree:

- Il pannello **Proprietà controllo**, che contiene una raccolta di pulsanti di opzione che è possibile utilizzare per modificare varie proprietà del controllo ospitato.
- **Dati dal pannello di controllo**, che contiene diversi elementi **TextBlock** che visualizzano i dati restituiti dal controllo ospitato.
- Il controllo ospitato stesso.

Il layout di base è illustrato nel codice XAML seguente. Il markup necessario per ospitare **MyControl1** viene omesso da questo esempio, ma verrà discusso più avanti.

Sostituire il codice XAML in MainWindow.xaml con il seguente. Se si utilizza Visual Basic, impostare la classe su **x:Class="MainWindow"**.

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        x:Class="WpfHost.MainWindow"
        xmlns:mcl="clr-namespace:MyControls;assembly=MyControls"
        Loaded="Init">
    <DockPanel>
        <DockPanel.Resources>
            <Style x:Key="inlineText" TargetType="{x:Type Inline}">
                <Setter Property="FontWeight" Value="Normal"/>
            </Style>
            <Style x:Key="titleText" TargetType="{x:Type TextBlock}">
                <Setter Property="DockPanel.Dock" Value="Top"/>
                <Setter Property="FontWeight" Value="Bold"/>
                <Setter Property="Margin" Value="10,5,10,0"/>
            </Style>
        </DockPanel.Resources>

        <StackPanel Orientation="Vertical"
                   DockPanel.Dock="Left"
                   Background="Bisque"
                   Width="250">

            <TextBlock Margin="10,10,10,10"
                      FontWeight="Bold"
                      FontSize="12">Control Properties</TextBlock>
            <TextBlock Style="{StaticResource titleText}">Background Color</TextBlock>
            <StackPanel Margin="10,10,10,10">
                <RadioButton Name="rdbtnOriginalBackColor"
                            IsChecked="True"
                            Click="BackColorChanged">Original</RadioButton>
                <RadioButton Name="rdbtnBackGreen"
                            Click="BackColorChanged">LightGreen</RadioButton>
                <RadioButton Name="rdbtnBackSalmon"
                            Click="BackColorChanged">LightSalmon</RadioButton>
            </StackPanel>
        </StackPanel>
    </DockPanel>
</Window>
```

```

        </StackPanel>

        <TextBlock Style="{StaticResource titleText}">Foreground Color</TextBlock>
        <StackPanel Margin="10,10,10,10">
            <RadioButton Name="rdbtnOriginalForeColor"
                IsChecked="True"
                Click="ForeColorChanged">Original</RadioButton>
            <RadioButton Name="rdbtnForeRed"
                Click="ForeColorChanged">Red</RadioButton>
            <RadioButton Name="rdbtnForeYellow"
                Click="ForeColorChanged">Yellow</RadioButton>
        </StackPanel>

        <TextBlock Style="{StaticResource titleText}">Font Family</TextBlock>
        <StackPanel Margin="10,10,10,10">
            <RadioButton Name="rdbtnOriginalFamily"
                IsChecked="True"
                Click="FontChanged">Original</RadioButton>
            <RadioButton Name="rdbtnTimes"
                Click="FontChanged">Times New Roman</RadioButton>
            <RadioButton Name="rdbtnWingdings"
                Click="FontChanged">Wingdings</RadioButton>
        </StackPanel>

        <TextBlock Style="{StaticResource titleText}">Font Size</TextBlock>
        <StackPanel Margin="10,10,10,10">
            <RadioButton Name="rdbtnOriginalSize"
                IsChecked="True"
                Click="FontSizeChanged">Original</RadioButton>
            <RadioButton Name="rdbtnTen"
                Click="FontSizeChanged">10</RadioButton>
            <RadioButton Name="rdbtnTwelve"
                Click="FontSizeChanged">12</RadioButton>
        </StackPanel>

        <TextBlock Style="{StaticResource titleText}">Font Style</TextBlock>
        <StackPanel Margin="10,10,10,10">
            <RadioButton Name="rdbtnNormalStyle"
                IsChecked="True"
                Click="StyleChanged">Original</RadioButton>
            <RadioButton Name="rdbtnItalic"
                Click="StyleChanged">Italic</RadioButton>
        </StackPanel>

        <TextBlock Style="{StaticResource titleText}">Font Weight</TextBlock>
        <StackPanel Margin="10,10,10,10">
            <RadioButton Name="rdbtnOriginalWeight"
                IsChecked="True"
                Click="WeightChanged">
                Original
            </RadioButton>
            <RadioButton Name="rdbtnBold"
                Click="WeightChanged">Bold</RadioButton>
        </StackPanel>
    </StackPanel>

    <WindowsFormsHost Name="wfh"
        DockPanel.Dock="Top"
        Height="300">
        <mcl:MyControl1 Name="mc"/>
    </WindowsFormsHost>

    <StackPanel Orientation="Vertical"
        Height="Auto"
        Background="LightBlue">
        <TextBlock Margin="10,10,10,10"
            FontWeight="Bold"
            FontSize="12">Data From Control</TextBlock>
        <TextBlock Style="{StaticResource titleText}">

```

```

<TextBlock Style="{StaticResource titleText}" />
    Name: <Span Name="txtName" Style="{StaticResource inlineText}" />
</TextBlock>
<TextBlock Style="{StaticResource titleText}">
    Street Address: <Span Name="txtAddress" Style="{StaticResource inlineText}" />
</TextBlock>
<TextBlock Style="{StaticResource titleText}">
    City: <Span Name="txtCity" Style="{StaticResource inlineText}" />
</TextBlock>
<TextBlock Style="{StaticResource titleText}">
    State: <Span Name="txtState" Style="{StaticResource inlineText}" />
</TextBlock>
<TextBlock Style="{StaticResource titleText}">
    Zip: <Span Name="txtZip" Style="{StaticResource inlineText}" />
</TextBlock>
</StackPanel>
</DockPanel>
</Window>

```

Il primo elemento **StackPanel** contiene diversi set di controlli **RadioButton** che consentono di modificare diverse proprietà predefinite del controllo ospitato. Seguito da un elemento **WindowsFormsHost** che ospita **MyControl1**. L'elemento **StackPanel** finale contiene diversi elementi **TextBlock** che visualizzano i dati restituiti dal controllo ospitato. L'ordinamento degli elementi e le impostazioni degli attributi **Dock** e **Height** incorporano il controllo ospitato nella finestra senza gap o distorsione.

Hosting del controllo

La versione modificata seguente del codice XAML precedente è incentrata sugli elementi necessari per ospitare **MyControl1**.

```

<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        x:Class="WpfHost.MainWindow"
        xmlns:mcl="clr-namespace:MyControls;assembly=MyControls"
        Loaded="Init">

```

```

<WindowsFormsHost Name="wfh"
                  DockPanel.Dock="Top"
                  Height="300">
    <mcl:MyControl1 Name="mc"/>
</WindowsFormsHost>

```

L'attributo di mapping dello spazio dei nomi **xmlns** crea un riferimento allo spazio dei nomi **MyControls** che contiene il controllo ospitato. Questo mapping consente di rappresentare **MyControl1** in XAML **<mcl:MyControl1>**.

Due elementi nel codice XAML gestiscono l'hosting:

- **WindowsFormsHost** rappresenta l'elemento **WindowsFormsHost** che consente di ospitare un controllo Windows Forms in un'applicazione WPF.
- **mcl:MyControl1**, che rappresenta **MyControl1**, viene aggiunto alla raccolta figlio dell'elemento **WindowsFormsHost**. Di conseguenza, viene eseguito il rendering di questo controllo Windows Forms come parte della finestra di WPF ed è possibile comunicare con il controllo dall'applicazione.

Implementazione del file code-behind

Il file code-behind, **MainWindow.xaml.vb** o **MainWindow.xaml.cs**, contiene il codice procedurale che implementa la funzionalità della Interfaccia utente descritta nella sezione precedente. Le attività principali sono:

- Associazione di un gestore eventi all'evento **OnButtonClick** di **MyControl1**.

- Modifica di diverse proprietà di `MyControl1`, in base alla modalità di impostazione della raccolta di pulsanti di opzione.
- Visualizzare i dati raccolti dal controllo.

Inizializzazione dell'applicazione

Il codice di inizializzazione è contenuto in un gestore eventi per l'evento `Loaded` della finestra e connette un gestore eventi all'evento `OnButtonClick` del controllo.

In `MainWindow.xaml.vb` o `MainWindow.xaml.cs` aggiungere il codice seguente alla classe `MainWindow`.

```
private Application app;
private Window myWindow;
FontWeight initFontWeight;
Double initFontSize;
FontStyle initFontStyle;
SolidColorBrush initBackBrush;
SolidColorBrush initForeBrush;
FontFamily initFontFamily;
bool UIIsReady = false;

private void Init(object sender, EventArgs e)
{
    app = System.Windows.Application.Current;
    myWindow = (Window)app.MainWindow;
    myWindow.SizeToContent = SizeToContent.WidthAndHeight;
    wfh.TabIndex = 10;
    initFontSize = wfh.FontSize;
    initFontWeight = wfh.FontWeight;
    initFontFamily = wfh.FontFamily;
    initFontStyle = wfh.FontStyle;
    initBackBrush = (SolidColorBrush)wfh.Background;
    initForeBrush = (SolidColorBrush)wfh.Foreground;
    (wfh.Child as MyControl1).OnButtonClick += new MyControl1.MyControlEventHandler(Pane1_OnButtonClick);
    UIIsReady = true;
}
```

```

Private app As Application
Private myWindow As Window
Private initFontWeight As FontWeight
Private initFontSize As [Double]
Private initFontStyle As FontStyle
Private initBackBrush As SolidColorBrush
Private initForeBrush As SolidColorBrush
Private initFontFamily As FontFamily
Private UIIsReady As Boolean = False

Private Sub Init(ByVal sender As Object, ByVal e As RoutedEventArgs)
    app = System.Windows.Application.Current
    myWindow = CType(app.MainWindow, Window)
    myWindow.SizeToContent = SizeToContent.WidthAndHeight
    wfh.TabIndex = 10
    initFontSize = wfh.FontSize
    initFontWeight = wfh.FontWeight
    initFontFamily = wfh.FontFamily
    initFontStyle = wfh.FontStyle
    initBackBrush = CType(wfh.Background, SolidColorBrush)
    initForeBrush = CType(wfh.Foreground, SolidColorBrush)

    Dim mc As MyControl1 = wfh.Child

    AddHandler mc.OnButtonClick, AddressOf Pane1_OnButtonClick
    UIIsReady = True
End Sub

```

Poiché la XAML descritta in precedenza ha aggiunto `MyControl1` alla raccolta di elementi figlio dell'elemento `WindowsFormsHost`, è possibile eseguire il cast del `Child` dell'elemento `WindowsFormsHost` per ottenere il riferimento a `MyControl1`. È quindi possibile utilizzare tale riferimento per allegare un gestore eventi a `OnButtonClick`.

Oltre a fornire un riferimento al controllo stesso, `WindowsFormsHost` espone alcune proprietà del controllo, che è possibile modificare dall'applicazione. Il codice di inizializzazione assegna tali valori a variabili globali private per l'uso successivo nell'applicazione.

Per poter accedere facilmente ai tipi nella DLL `MyControls`, aggiungere la `Imports` o l'istruzione `using` seguente all'inizio del file.

```
Imports MyControls
```

```
using MyControls;
```

Gestione dell'evento `OnButtonClick`

`MyControl1` genera l'evento `OnButtonClick` quando l'utente fa clic su uno dei pulsanti del controllo.

Aggiungere il codice seguente alla classe `MainWindow`.

```

//Handle button clicks on the Windows Form control
private void Panel1_OnButtonClick(object sender, MyControlEventArgs args)
{
    txtName.Inlines.Clear();
    txtAddress.Inlines.Clear();
    txtCity.Inlines.Clear();
    txtState.Inlines.Clear();
    txtZip.Inlines.Clear();

    if (args.IsOK)
    {
        txtName.Inlines.Add( " " + args.MyName );
        txtAddress.Inlines.Add( " " + args.MyStreetAddress );
        txtCity.Inlines.Add( " " + args.MyCity );
        txtState.Inlines.Add( " " + args.MyState );
        txtZip.Inlines.Add( " " + args.MyZip );
    }
}

```

```

'Handle button clicks on the Windows Form control
Private Sub Panel1_OnButtonClick(ByVal sender As Object, ByVal args As MyControlEventArgs)
    txtName.Inlines.Clear()
    txtAddress.Inlines.Clear()
    txtCity.Inlines.Clear()
    txtState.Inlines.Clear()
    txtZip.Inlines.Clear()

    If args.IsOK Then
        txtName.Inlines.Add(" " + args.MyName)
        txtAddress.Inlines.Add(" " + args.MyStreetAddress)
        txtCity.Inlines.Add(" " + args.MyCity)
        txtState.Inlines.Add(" " + args.MyState)
        txtZip.Inlines.Add(" " + args.MyZip)
    End If
End Sub

```

I dati nelle caselle di testo vengono compressi nell'oggetto `MyControlEventArgs`. Se l'utente fa clic sul pulsante **OK**, il gestore dell'evento estraie i dati e li visualizza nel pannello riportato di seguito `MyControl1`.

Modifica delle proprietà del controllo

L'elemento `WindowsFormsHost` espone diverse proprietà predefinite del controllo ospitato. Di conseguenza, è possibile modificare l'aspetto del controllo in modo che corrisponda maggiormente allo stile dell'applicazione. Gli insiemi di pulsanti di opzione nel pannello di sinistra consentono all'utente di modificare varie proprietà di colore e tipo di carattere. Ogni set di pulsanti dispone di un gestore per l'evento `Click`, che rileva le selezioni dei pulsanti di opzione dell'utente e modifica la proprietà corrispondente nel controllo.

Aggiungere il codice seguente alla classe `MainWindow`.

```

private void BackColorChanged(object sender, RoutedEventArgs e)
{
    if (sender == rdbtnBackGreen)
        wfh.Background = new SolidColorBrush(Colors.LightGreen);
    else if (sender == rdbtnBackSalmon)
        wfh.Background = new SolidColorBrush(Colors.LightSalmon);
    else if (UIIsReady == true)
        wfh.Background = initBackBrush;
}

private void ForeColorChanged(object sender, RoutedEventArgs e)
{
    if (sender == rdbtnForeRed)
        wfh.Foreground = new SolidColorBrush(Colors.Red);
    else if (sender == rdbtnForeYellow)
        wfh.Foreground = new SolidColorBrush(Colors.Yellow);
    else if (UIIsReady == true)
        wfh.Foreground = initForeBrush;
}

private void FontChanged(object sender, RoutedEventArgs e)
{
    if (sender == rdbtnTimes)
        wfh.FontFamily = new FontFamily("Times New Roman");
    else if (sender == rdbtnWingdings)
        wfh.FontFamily = new FontFamily("Wingdings");
    else if (UIIsReady == true)
        wfh.FontFamily = initFontFamily;
}

private void FontSizeChanged(object sender, RoutedEventArgs e)
{
    if (sender == rdbtnTen)
        wfh.FontSize = 10;
    else if (sender == rdbtnTwelve)
        wfh.FontSize = 12;
    else if (UIIsReady == true)
        wfh.FontSize = initFontSize;
}

private void StyleChanged(object sender, RoutedEventArgs e)
{
    if (sender == rdbtnItalic)
        wfh.FontStyle = FontStyles.Italic;
    else if (UIIsReady == true)
        wfh.FontStyle = initFontStyle;
}

private void WeightChanged(object sender, RoutedEventArgs e)
{
    if (sender == rdbtnBold)
        wfh.FontWeight = FontWeights.Bold;
    else if (UIIsReady == true)
        wfh.FontWeight = initFontWeight;
}

```

```

Private Sub BackColorChanged(ByVal sender As Object, ByVal e As RoutedEventArgs)
    If sender.Equals(rdbtnBackGreen) Then
        wfh.Background = New SolidColorBrush(Colors.LightGreen)
    ElseIf sender.Equals(rdbtnBackSalmon) Then
        wfh.Background = New SolidColorBrush(Colors.LightSalmon)
    ElseIf UIIsReady = True Then
        wfh.Background = initBackBrush
    End If
End Sub

Private Sub ForeColorChanged(ByVal sender As Object, ByVal e As RoutedEventArgs)
    If sender.Equals(rdbtnForeRed) Then
        wfh.Foreground = New SolidColorBrush(Colors.Red)
    ElseIf sender.Equals(rdbtnForeYellow) Then
        wfh.Foreground = New SolidColorBrush(Colors.Yellow)
    ElseIf UIIsReady = True Then
        wfh.Foreground = initForeBrush
    End If
End Sub

Private Sub FontChanged(ByVal sender As Object, ByVal e As RoutedEventArgs)
    If sender.Equals(rdbtnTimes) Then
        wfh.FontFamily = New FontFamily("Times New Roman")
    ElseIf sender.Equals(rdbtnWingdings) Then
        wfh.FontFamily = New FontFamily("Wingdings")
    ElseIf UIIsReady = True Then
        wfh.FontFamily = initFontFamily
    End If
End Sub

Private Sub FontSizeChanged(ByVal sender As Object, ByVal e As RoutedEventArgs)
    If sender.Equals(rdbtnTen) Then
        wfh.FontSize = 10
    ElseIf sender.Equals(rdbtnTwelve) Then
        wfh.FontSize = 12
    ElseIf UIIsReady = True Then
        wfh.FontSize = initFontSize
    End If
End Sub

Private Sub StyleChanged(ByVal sender As Object, ByVal e As RoutedEventArgs)
    If sender.Equals(rdbtnItalic) Then
        wfh.FontStyle = FontStyles.Italic
    ElseIf UIIsReady = True Then
        wfh.FontStyle = initFontStyle
    End If
End Sub

Private Sub WeightChanged(ByVal sender As Object, ByVal e As RoutedEventArgs)
    If sender.Equals(rdbtnBold) Then
        wfh.FontWeight = FontWeights.Bold
    ElseIf UIIsReady = True Then
        wfh.FontWeight = initFontWeight
    End If
End Sub

```

Compilare ed eseguire l'applicazione. Aggiungere testo nel controllo Windows Forms composito, quindi fare clic su **OK**. Il testo viene visualizzato nelle etichette. Fare clic sui vari pulsanti di opzione per vedere l'effetto sul controllo.

Vedere anche

- [ElementHost](#)
- [WindowsFormsHost](#)
- [Progettare XAML in Visual Studio](#)
- [Procedura dettagliata: hosting di controlli Windows Form in WPF](#)
- [Procedura dettaglia: hosting di un controllo WPF composito in Windows Form](#)

Procedura dettagliata: hosting di un controllo ActiveX in WPF

31/01/2020 • 5 minutes to read • [Edit Online](#)

Per consentire una migliore interazione con i browser, è possibile utilizzare i controlli Microsoft ActiveX nell'applicazione basata su WPF. In questa procedura dettagliata viene illustrato come è possibile ospitare Microsoft Windows Media Player come controllo in una pagina di WPF.

Le attività illustrate nella procedura dettagliata sono le seguenti:

- Creazione del progetto.
- Creazione del controllo ActiveX.
- Hosting del controllo ActiveX in una pagina WPF.

Al termine di questa procedura dettagliata, si apprenderà come usare i controlli Microsoft ActiveX nell'applicazione basata su WPF.

Prerequisiti

Per completare la procedura dettagliata, è necessario disporre dei componenti seguenti:

- Microsoft Windows Media Player installato nel computer in cui è installato Visual Studio.
- Visual Studio 2010.

Creazione del progetto

Per creare e impostare il progetto

1. Creare un progetto di applicazione WPF denominato `HostingAxInWpf`.
2. Aggiungere un progetto libreria di controlli Windows Forms alla soluzione e denominare il progetto `WmpAxLib`.
3. Nel progetto `WmpAxLib` aggiungere un riferimento all'assembly di Windows Media Player, denominato `wmp.dll`.
4. Aprire la **casella degli strumenti**.
5. Fare clic con il pulsante destro del mouse nella **casella degli strumenti**, quindi **scegliere Scegli elementi**.
6. Fare clic sulla scheda **componenti com**, selezionare il controllo **Media Player Windows**, quindi fare clic su **OK**.

Il controllo Media Player Windows viene aggiunto alla **casella degli strumenti**.

7. In Esplora soluzioni fare clic con il pulsante destro del mouse sul file **UserControl1** e quindi scegliere **Rinomina**.
8. Modificare il nome in `WmpAxControl1.vb` o `WmpAxControl1.cs`, a seconda del linguaggio.
9. Se viene richiesto di rinominare tutti i riferimenti, fare clic su **Sì**.

Creazione del controllo ActiveX

Visual Studio genera automaticamente una classe wrapper [AxHost](#) per un controllo Microsoft ActiveX quando il controllo viene aggiunto a un'area di progettazione. La procedura seguente consente di creare un assembly gestito denominato AxInterop.WMPLib.dll.

Per creare il controllo ActiveX

1. Aprire WmpAxControl.vb o WmpAxControl.cs nel Progettazione Windows Form.
2. Dalla **casella degli strumenti** aggiungere il controllo Windows Media Player nell'area di progettazione.
3. Nella Finestra Proprietà impostare il valore della proprietà [Dock](#) del controllo Media Player di Windows su [Fill](#).
4. Compilare il progetto libreria di controlli WmpAxLib.

Hosting del controllo ActiveX in una pagina WPF

Per ospitare il controllo ActiveX

1. Nel progetto HostingAxInWpf aggiungere un riferimento all'assembly di interoperabilità ActiveX generato. Questo assembly è denominato AxInterop.WMPLib.dll ed è stato aggiunto alla cartella debug del progetto WmpAxLib quando è stato importato il controllo Media Player di Windows.
2. Aggiungere un riferimento all'assembly WindowsFormsIntegration, denominato WindowsFormsIntegration.dll.
3. Aggiungere un riferimento all'assembly Windows Forms, denominato System.Windows.Forms.dll.
4. Aprire MainWindow.xaml in WPF Designer.
5. Denominare l'elemento [Grid](#) `grid1`.

```
<Grid Name="grid1">  
  </Grid>
```

6. Nella visualizzazione progettazione o nella visualizzazione XAML Selezionare l'elemento [Window](#).
7. Nella Finestra Proprietà fare clic sulla scheda **eventi**.
8. Fare doppio clic sull'evento [Loaded](#).
9. Inserire il codice seguente per gestire l'evento [Loaded](#).

Questo codice crea un'istanza del controllo [WindowsFormsHost](#) e aggiunge un'istanza del controllo [AxWindowsMediaPlayer](#) come figlio.

```

private void Window_Loaded(object sender, RoutedEventArgs e)
{
    // Create the interop host control.
    System.Windows.Forms.Integration.WindowsFormsHost host =
        new System.Windows.Forms.Integration.WindowsFormsHost();

    // Create the ActiveX control.
    WmpAxLib.AxWindowsMediaPlayer axWmp = new WmpAxLib.AxWindowsMediaPlayer();

    // Assign the ActiveX control as the host control's child.
    host.Child = axWmp;

    // Add the interop host control to the Grid
    // control's collection of child controls.
    this.grid1.Children.Add(host);

    // Play a .wav file with the ActiveX control.
    axWmp.URL = @"C:\Windows\Media\tada.wav";
}

```

```

Private Sub Window_Loaded(ByVal sender As Object, ByVal e As RoutedEventArgs)

    ' Create the interop host control.
    Dim host As New System.Windows.Forms.Integration.WindowsFormsHost()

    ' Create the ActiveX control.
    Dim axWmp As New AxWMPLib.AxWindowsMediaPlayer()

    ' Assign the ActiveX control as the host control's child.
    host.Child = axWmp

    ' Add the interop host control to the Grid
    ' control's collection of child controls.
    Me.grid1.Children.Add(host)

    ' Play a .wav file with the ActiveX control.
    axWmp.URL = "C:\Windows\Media\tada.wav"

End Sub

```

10. Premere F5 per compilare ed eseguire l'applicazione.

Vedere anche

- [ElementHost](#)
- [WindowsFormsHost](#)
- [Progettare XAML in Visual Studio](#)
- [Procedura dettagliata: Hosting di controlli Windows Form compositi in WPF](#)
- [Procedura dettaglia: hosting di un controllo WPF composito in Windows Form](#)

Procedura: Abilitare stili di visualizzazione in un'applicazione ibrida

31/01/2020 • 3 minutes to read • [Edit Online](#)

In questo argomento viene illustrato come abilitare gli stili di visualizzazione in un controllo Windows Forms ospitato in un'applicazione basata su WPF.

Se l'applicazione chiama il metodo [EnableVisualStyles](#), la maggior parte dei controlli Windows Forms utilizzerà automaticamente gli stili di visualizzazione. Per altre informazioni, vedere [Rendering dei controlli con stili visivi](#).

Per un listato di codice completo delle attività illustrate in questo argomento, vedere l'articolo relativo all'[Abilitazione degli stili di visualizzazione in un'applicazione ibrida](#).

Attivazione degli stili di visualizzazione Windows Form

Per attivare gli stili di visualizzazione Windows Form

1. Creare un progetto di applicazione WPF denominato `HostingWfWithVisualStyles`.
2. In Esplora soluzioni aggiungere riferimenti agli assembly seguenti.
 - `WindowsFormsIntegration`
 - `System.Windows.Forms`
3. Nella casella degli strumenti fare doppio clic sull'icona [Grid](#) per inserire un elemento [Grid](#) nell'area di progettazione.
4. Nella Finestra Proprietà impostare i valori delle proprietà [Height](#) e [Width](#) su **auto**.
5. Nella visualizzazione progettazione o nella visualizzazione XAML Selezionare il [Window](#).
6. Nella Finestra Proprietà fare clic sulla scheda **eventi**.
7. Fare doppio clic sull'evento [Loaded](#).
8. In `MainWindow.xaml`, `MainWindow.xaml.cs` o `MainWindow.xaml.vb` inserire il codice seguente per gestire l'evento [Loaded](#).

```

private void Window_Loaded(object sender, RoutedEventArgs e)
{
    // Comment out the following line to disable visual
    // styles for the hosted Windows Forms control.
    System.Windows.Forms.Application.EnableVisualStyles();

    // Create a WindowsFormsHost element to host
    // the Windows Forms control.
    System.Windows.Forms.Integration.WindowsFormsHost host =
        new System.Windows.Forms.Integration.WindowsFormsHost();

    // Create a Windows Forms tab control.
    System.Windows.Forms.TabControl tc = new System.Windows.Forms.TabControl();
    tc.TabPages.Add("Tab1");
    tc.TabPages.Add("Tab2");

    // Assign the Windows Forms tab control as the hosted control.
    host.Child = tc;

    // Assign the host element to the parent Grid element.
    this.grid1.Children.Add(host);
}

```

```

Private Sub Window_Loaded(ByVal sender As Object, ByVal e As RoutedEventArgs)
    ' Comment out the following line to disable visual
    ' styles for the hosted Windows Forms control.
    System.Windows.Forms.Application.EnableVisualStyles()

    ' Create a WindowsFormsHost element to host
    ' the Windows Forms control.
    Dim host As New System.Windows.Forms.Integration.WindowsFormsHost()

    ' Create a Windows Forms tab control.
    Dim tc As New System.Windows.Forms.TabControl()
    tc.TabPages.Add("Tab1")
    tc.TabPages.Add("Tab2")

    ' Assign the Windows Forms tab control as the hosted control.
    host.Child = tc

    ' Assign the host element to the parent Grid element.
    Me.grid1.Children.Add(host)

End Sub

```

9. Premere F5 per compilare ed eseguire l'applicazione.

Il controllo Windows Forms viene disegnato con stili di visualizzazione.

Disattivazione degli stili di visualizzazione Windows Form

Per disabilitare gli stili di visualizzazione, è sufficiente rimuovere la chiamata al metodo [EnableVisualStyles](#).

Per disattivare gli stili di visualizzazione Windows Form

1. Aprire MainWindow.xaml.vb o MainWindow.xaml.cs nell'editor di codice.
2. Impostare come commento la chiamata al metodo [EnableVisualStyles](#).
3. Premere F5 per compilare ed eseguire l'applicazione.

Il controllo Windows Forms viene disegnato con lo stile di sistema predefinito.

Vedere anche

- [EnableVisualStyles](#)
- [System.Windows.Forms.VisualStyles](#)
- [WindowsFormsHost](#)
- [Rendering dei controlli con stili visivi](#)
- [Procedura dettagliata: hosting di controlli Windows Form in WPF](#)

Procedura dettagliata: disposizione di controlli Windows Form in WPF

10/02/2020 • 16 minutes to read • [Edit Online](#)

Questa procedura dettagliata illustra come usare le funzionalità di layout WPF per disporre i controlli Windows Forms in un'applicazione ibrida.

Le attività illustrate nella procedura dettagliata sono le seguenti:

- Creazione del progetto.
- Uso delle impostazioni di layout predefinite.
- Ridimensionamento in base al contenuto.
- Uso del posizionamento assoluto.
- Specifica esplicita delle dimensioni.
- Impostazione delle proprietà di layout.
- Informazioni sulle limitazioni di z order.
- Ancoraggio.
- Impostazione della visibilità.
- Hosting di un controllo che non si adatta.
- Scalabilità.
- Rotazione.
- Impostazione della spaziatura interna e dei margini.
- Uso di contenitori di layout dinamici.

Per un listato di codice completo delle attività illustrate in questa procedura dettagliata, vedere [disposizione di Windows Forms controlli in WPF di esempio](#).

Al termine, sarà possibile comprendere Windows Forms funzionalità di layout nelle applicazioni basate su WPF.

Prerequisites

Per completare la procedura dettagliata, è necessario Visual Studio.

Creazione del progetto

Per creare e configurare il progetto, attenersi alla procedura seguente:

1. Creare un progetto di applicazione WPF denominato `WpfLayoutHostingWf`.
2. In Esplora soluzioni aggiungere riferimenti agli assembly seguenti:
 - WindowsFormsIntegration
 - System.Windows.Forms
 - System.Drawing
3. Fare doppio clic su *MainWindow.xaml* per aprirlo nella visualizzazione XAML.
4. Nell'elemento `Window` aggiungere il seguente Windows Forms mapping dello spazio dei nomi.

```
xmlns:wf="clr-namespace:System.Windows.Forms;assembly=System.Windows.Forms"
```

5. Nell'elemento **Grid** impostare la proprietà **ShowGridLines** su **true** e definire cinque righe e tre colonne.

```
<Grid ShowGridLines="true">
  <Grid.RowDefinitions>
    <RowDefinition/>
    <RowDefinition/>
    <RowDefinition/>
    <RowDefinition/>
    <RowDefinition/>
  </Grid.RowDefinitions>

  <Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition/>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>
```

Uso delle impostazioni di layout predefinite

Per impostazione predefinita, l'elemento **WindowsFormsHost** gestisce il layout per il controllo Windows Forms ospitato.

Per usare le impostazioni di layout predefinite, seguire questa procedura:

1. Copiare il codice XAML seguente nell'elemento **Grid**:

```
<!-- Default layout. -->
<Canvas Grid.Row="0" Grid.Column="0">
  <WindowsFormsHost Background="Yellow">
    <wf:Button Text="Windows Forms control" FlatStyle="Flat"/>
  </WindowsFormsHost>
</Canvas>
```

2. Premere F5 per compilare ed eseguire l'applicazione. Il controllo Windows Forms **System.Windows.Forms.Button** viene visualizzato nella **Canvas**. Il controllo ospitato viene ridimensionato in base al relativo contenuto e l'elemento **WindowsFormsHost** viene dimensionato per contenere il controllo ospitato.

Ridimensionamento in base al contenuto

L'elemento **WindowsFormsHost** garantisce che il controllo ospitato venga ridimensionato per visualizzare correttamente il contenuto.

Per ridimensionare il contenuto, attenersi alla seguente procedura:

1. Copiare il codice XAML seguente nell'elemento **Grid**:

```

<!-- Sizing to content. -->
<Canvas Grid.Row="1" Grid.Column="0">
    <WindowsFormsHost Background="Orange">
        <wf:Button Text="Windows Forms control with more content" FlatStyle="Flat"/>
    </WindowsFormsHost>
</Canvas>

<Canvas Grid.Row="2" Grid.Column="0">
    <WindowsFormsHost FontSize="24" Background="Yellow">
        <wf:Button Text="Windows Forms control" FlatStyle="Flat"/>
    </WindowsFormsHost>
</Canvas>

```

- Premere F5 per compilare ed eseguire l'applicazione. I due nuovi controlli pulsante vengono ridimensionati in modo da visualizzare la stringa di testo più lunga e le dimensioni del carattere più grandi e gli elementi [WindowsFormsHost](#) vengono ridimensionati per contenere i controlli ospitati.

Uso del posizionamento assoluto

È possibile usare il posizionamento assoluto per inserire l'elemento [WindowsFormsHost](#) in un punto qualsiasi dell'interfaccia utente.

Per usare il posizionamento assoluto, attenersi alla procedura seguente:

- Copiare il codice XAML seguente nell'elemento [Grid](#):

```

<!-- Absolute positioning. -->
<Canvas Grid.Row="3" Grid.Column="0">
    <WindowsFormsHost Canvas.Top="20" Canvas.Left="20" Background="Yellow">
        <wf:Button Text="Windows Forms control with absolute positioning" FlatStyle="Flat"/>
    </WindowsFormsHost>
</Canvas>

```

- Premere F5 per compilare ed eseguire l'applicazione. L'elemento [WindowsFormsHost](#) viene inserito a 20 pixel dal lato superiore della cella della griglia e a 20 pixel da sinistra.

Specifiche esplicativa delle dimensioni

È possibile specificare le dimensioni dell'elemento [WindowsFormsHost](#) usando le proprietà [Width](#) e [Height](#).

Per specificare le dimensioni in modo esplicito, attenersi alla procedura seguente:

- Copiare il codice XAML seguente nell'elemento [Grid](#):

```

<!-- Explicit sizing. -->
<Canvas Grid.Row="4" Grid.Column="0">
    <WindowsFormsHost Width="50" Height="70" Background="Yellow">
        <wf:Button Text="Windows Forms control" FlatStyle="Flat"/>
    </WindowsFormsHost>
</Canvas>

```

- Premere F5 per compilare ed eseguire l'applicazione. L'elemento [WindowsFormsHost](#) è impostato su una dimensione di 50 pixel in larghezza per 70 pixel di altezza, minore rispetto alle impostazioni predefinite del layout. Il contenuto del controllo Windows Forms viene ridisposto di conseguenza.

Impostazione delle proprietà di layout

Impostare sempre le proprietà correlate al layout nel controllo ospitato usando le proprietà dell'elemento

[WindowsFormsHost](#). L'impostazione diretta delle proprietà di layout nel controllo ospitato darà risultati imprevisti.

L'impostazione delle proprietà correlate al layout nel controllo ospitato in XAML non ha alcun effetto.

Per visualizzare gli effetti dell'impostazione delle proprietà nel controllo ospitato, attenersi alla seguente procedura:

1. Copiare il codice XAML seguente nell'elemento [Grid](#):

```
<!-- Setting hosted control properties directly. -->
<Canvas Grid.Row="0" Grid.Column="1">
    <WindowsFormsHost Width="160" Height="50" Background="Yellow">
        <wf:Button Name="button1" Click="button1_Click" Text="Click me" FlatStyle="Flat" BackColor="Green"/>
    </WindowsFormsHost>
</Canvas>
```

2. In **Esplora soluzioni** fare doppio clic su *MainWindow.xaml.vb* o *MainWindow.xaml.cs* per aprirlo nell'editor di codice.

3. Copiare il codice seguente nella definizione della classe [MainWindow](#):

```
private void button1_Click(object sender, EventArgs e )
{
    System.Windows.Forms.Button b = sender as System.Windows.Forms.Button;

    b.Top = 20;
    b.Left = 20;
}
```

```
Private Sub button1_Click(ByVal sender As Object, ByVal e As EventArgs)
    Dim b As System.Windows.Forms.Button = sender

    b.Top = 20
    b.Left = 20

End Sub
```

4. Premere F5 per compilare ed eseguire l'applicazione.

5. Fare clic sul pulsante **fare clic su me**. Il gestore dell'evento [button1_Click](#) imposta le proprietà [Top](#) e [Left](#) sul controllo ospitato. In questo modo il controllo ospitato viene riposizionato all'interno dell'elemento [WindowsFormsHost](#). L'host mantiene la stessa area dello schermo, ma il controllo ospitato viene ritagliato. Al contrario, il controllo ospitato deve riempire sempre l'elemento [WindowsFormsHost](#).

Informazioni sulle limitazioni di z order

Gli elementi [WindowsFormsHost](#) visibili vengono sempre disegnati sopra gli altri elementi WPF e non sono interessati dall'ordine z. Per visualizzare questo comportamento dell'ordine z, eseguire le operazioni seguenti:

1. Copiare il codice XAML seguente nell'elemento [Grid](#):

```
<!-- Z-order demonstration. -->
<Canvas Grid.Row="1" Grid.Column="1">
    <WindowsFormsHost Canvas.Top="20" Canvas.Left="20" Background="Yellow">
        <wf:Button Text="Windows Forms control" FlatStyle="Flat"/>
    </WindowsFormsHost>
    <Label Content="A WPF label" FontSize="24"/>
</Canvas>
```

- Premere F5 per compilare ed eseguire l'applicazione. L'elemento [WindowsFormsHost](#) viene disegnato sull'elemento label.

Docking

[WindowsFormsHost](#) elemento supporta l'ancoraggio WPF. Impostare la proprietà associata [Dock](#) per ancorare il controllo ospitato in un elemento [DockPanel](#).

Per ancorare un controllo ospitato, attenersi alla procedura seguente:

- Copiare il codice XAML seguente nell'elemento [Grid](#):

```
<!-- Docking a WindowsFormsHost element. -->
<DockPanel LastChildFill="false" Grid.Row="2" Grid.Column="1">
    <WindowsFormsHost DockPanel.Dock="Right" Canvas.Top="20" Canvas.Left="20" Background="Yellow">
        <wf:Button Text="Windows Forms control" FlatStyle="Flat"/>
    </WindowsFormsHost>
</DockPanel>
```

- Premere F5 per compilare ed eseguire l'applicazione. L'elemento [WindowsFormsHost](#) è ancorato al lato destro dell'elemento [DockPanel](#).

Impostazione della visibilità

È possibile rendere invisibile il controllo Windows Forms o comprimerlo impostando la proprietà [Visibility](#) sull'elemento [WindowsFormsHost](#). Quando un controllo non è visibile, non viene visualizzato, ma occupa spazio del layout. Quando un controllo è compresso, non viene visualizzato né occupa spazio del layout.

Per impostare la visibilità di un controllo ospitato, attenersi alla procedura seguente:

- Copiare il codice XAML seguente nell'elemento [Grid](#):

```
<!-- Setting Visibility to hidden and collapsed. -->
<StackPanel Grid.Row="3" Grid.Column="1">
    <Button Name="button2" Click="button2_Click" Content="Click to make invisible" Background="OrangeRed"/>
    <WindowsFormsHost Name="host1" Background="Yellow">
        <wf:Button Text="Windows Forms control" FlatStyle="Flat"/>
    </WindowsFormsHost>
    <Button Name="button3" Click="button3_Click" Content="Click to collapse" Background="OrangeRed"/>
</StackPanel>
```

- In *MainWindow.xaml.vb* o *MainWindow.xaml.cs* copiare il codice seguente nella definizione della classe:

```
private void button2_Click(object sender, EventArgs e)
{
    this.host1.Visibility = Visibility.Hidden;
}

private void button3_Click(object sender, EventArgs e)
{
    this.host1.Visibility = Visibility.Collapsed;
}
```

```

Private Sub button2_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Me.host1.Visibility = Windows.Visibility.Hidden
End Sub

Private Sub button3_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Me.host1.Visibility = Windows.Visibility.Collapsed
End Sub

```

3. Premere F5 per compilare ed eseguire l'applicazione.
4. Fare clic sul pulsante **fare clic per rendere invisibile** per rendere invisibile l'elemento [WindowsFormsHost](#).
5. Fare clic sul pulsante **fare clic per comprimere** per nascondere completamente l'elemento [WindowsFormsHost](#) dal layout. Quando il controllo Windows Forms viene compresso, gli elementi circostanti vengono ridisposti per occuparne lo spazio.

Hosting di un controllo che non si adatta

Alcuni controlli Windows Forms hanno una dimensione fissa e non si adattano per riempire lo spazio disponibile nel layout. Ad esempio, il controllo [MonthCalendar](#) Visualizza un mese in uno spazio fisso.

Per ospitare un controllo che non si estende, attenersi alla procedura seguente:

1. Copiare il codice XAML seguente nell'elemento [Grid](#):

```

<!-- Hosting a control that does not stretch. -->
<!-- The MonthCalendar has a discrete size. -->
<StackPanel Grid.Row="4" Grid.Column="1">
    <Label Content="A WPF element" Background="OrangeRed"/>
    <WindowsFormsHost Background="Yellow">
        <wf:MonthCalendar/>
    </WindowsFormsHost>
    <Label Content="Another WPF element" Background="OrangeRed"/>
</StackPanel>

```

2. Premere F5 per compilare ed eseguire l'applicazione. L'elemento [WindowsFormsHost](#) viene centrato nella riga della griglia, ma non viene allungato per riempire lo spazio disponibile. Se la finestra è sufficientemente grande, è possibile che vengano visualizzati due o più mesi dal controllo [MonthCalendar](#) host, che però sono centrati nella riga. Il motore di layout di WPF centra gli elementi che non possono essere ridimensionati per riempire lo spazio disponibile.

Scalabilità

Diversamente dagli elementi WPF, la maggior parte dei controlli Windows Forms non è continuamente scalabile. Per fornire scalabilità personalizzata, è necessario eseguire l'override del metodo [WindowsFormsHost.ScaleChild](#).

Per ridimensionare un controllo ospitato usando il comportamento predefinito, attenersi alla procedura seguente:

1. Copiare il codice XAML seguente nell'elemento [Grid](#):

```

<!-- Scaling transformation. -->
<StackPanel Grid.Row="0" Grid.Column="2">

    <StackPanel.RenderTransform>
        <ScaleTransform CenterX="0" CenterY="0" ScaleX="0.5" ScaleY="0.5" />
    </StackPanel.RenderTransform>

    <Label Content="A WPF UIElement" Background="OrangeRed"/>

    <WindowsFormsHost Background="Yellow">
        <wf:Button Text="Windows Forms control" FlatStyle="Flat"/>
    </WindowsFormsHost>

    <Label Content="Another WPF UIElement" Background="OrangeRed"/>

</StackPanel>

```

- Premere F5 per compilare ed eseguire l'applicazione. Il controllo ospitato e gli elementi che lo circondano vengono ridimensionati in base a un fattore di 0,5. Tuttavia il tipo di carattere del controllo ospitato non viene ridimensionato.

Rotazione

Diversamente dagli elementi WPF, i controlli Windows Forms non supportano la rotazione. L'elemento [WindowsFormsHost](#) non viene ruotato con altri elementi WPF quando viene applicata una trasformazione di rotazione. Qualsiasi valore di rotazione diverso da 180 gradi genera l'evento [LayoutError](#).

Per visualizzare l'effetto della rotazione in un'applicazione ibrida, attenersi alla procedura seguente:

- Copiare il codice XAML seguente nell'elemento [Grid](#):

```

<!-- Rotation transformation. -->
<StackPanel Grid.Row="1" Grid.Column="2">

    <StackPanel.RenderTransform>
        <RotateTransform CenterX="200" CenterY="50" Angle="180" />
    </StackPanel.RenderTransform>

    <Label Content="A WPF element" Background="OrangeRed"/>

    <WindowsFormsHost Background="Yellow">
        <wf:Button Text="Windows Forms control" FlatStyle="Flat"/>
    </WindowsFormsHost>

    <Label Content="Another WPF element" Background="OrangeRed"/>

</StackPanel>

```

- Premere F5 per compilare ed eseguire l'applicazione. Il controllo ospitato non viene ruotato, ma i relativi elementi circostanti vengono ruotati di 180 gradi. Potrebbe essere necessario ridimensionare la finestra per vedere gli elementi.

Impostazione della spaziatura interna e dei margini

La spaziatura interna e i margini nel layout WPF sono simili a spaziatura interna e margini in Windows Forms. È sufficiente impostare le proprietà [Padding](#) e [Margin](#) sull'elemento [WindowsFormsHost](#).

Per impostare spaziatura interna e margini per un controllo ospitato, attenersi alla seguente procedura:

- Copiare il codice XAML seguente nell'elemento [Grid](#):

```

<!-- Padding. -->
<Canvas Grid.Row="2" Grid.Column="2">
    <WindowsFormsHost Padding="0, 20, 0, 0" Background="Yellow">
        <wf:Button Text="Windows Forms control with padding" FlatStyle="Flat"/>
    </WindowsFormsHost>
</Canvas>

```

```

<!-- Margin. -->
<Canvas Grid.Row="3" Grid.Column="2">
    <WindowsFormsHost Margin="20, 20, 0, 0" Background="Yellow">
        <wf:Button Text="Windows Forms control with margin" FlatStyle="Flat"/>
    </WindowsFormsHost>
</Canvas>

```

- Premere F5 per compilare ed eseguire l'applicazione. Le impostazioni di spaziatura interna e margini vengono applicate ai controlli Windows Forms ospitati nello stesso modo in cui vengono applicati in Windows Forms.

Uso di contenitori di layout dinamici

Windows Forms fornisce due contenitori di layout dinamici, [FlowLayoutPanel](#) e [TableLayoutPanel](#). È anche possibile usare questi contenitori in layout WPF.

Per usare un contenitore di layout dinamico, attenersi alla procedura seguente:

- Copiare il codice XAML seguente nell'elemento [Grid](#):

```

<!-- Flow layout. -->
<DockPanel Grid.Row="4" Grid.Column="2">
    <WindowsFormsHost Name="flowLayoutHost" Background="Yellow">
        <wf:FlowLayoutPanel/>
    </WindowsFormsHost>
</DockPanel>

```

- In *MainWindow.xaml.vb* o *MainWindow.xaml.cs* copiare il codice seguente nella definizione della classe:

```

private void InitializeFlowLayoutPanel()
{
    System.Windows.Forms.FlowLayoutPanel flp =
        this.flowLayoutPanelHost.Child as System.Windows.Forms.FlowLayoutPanel;

    flp.WrapContents = true;

    const int numButtons = 6;

    for (int i = 0; i < numButtons; i++)
    {
        System.Windows.Forms.Button b = new System.Windows.Forms.Button();
        b.Text = "Button";
        b.BackColor = System.Drawing.Color.AliceBlue;
        b.FlatStyle = System.Windows.Forms.FlatStyle.Flat;

        flp.Controls.Add(b);
    }
}

```

```

Private Sub InitializeFlowLayoutPanel()
    Dim flp As System.Windows.Forms.FlowLayoutPanel = Me.flowLayoutPanelHost.Child

    flp.WrapContents = True

    Const numButtons As Integer = 6

    Dim i As Integer
    For i = 0 To numButtons
        Dim b As New System.Windows.Forms.Button()
        b.Text = "Button"
        b.BackColor = System.Drawing.Color.AliceBlue
        b.FlatStyle = System.Windows.Forms.FlatStyle.Flat

        flp.Controls.Add(b)
    Next i

End Sub

```

3. Aggiungere una chiamata al metodo `InitializeFlowLayoutPanel` nel costruttore:

```

public MainWindow()
{
    InitializeComponent();

    this.InitializeFlowLayoutPanel();
}

```

```

Public Sub New()
    InitializeComponent()

    Me.InitializeFlowLayoutPanel()

End Sub

```

4. Premere F5 per compilare ed eseguire l'applicazione. L'elemento `WindowsFormsHost` riempie l'`DockPanel`. `FlowLayoutPanel` dispone i relativi controlli figlio nel `FlowDirection` predefinito.

Vedere anche

- [ElementHost](#)
- [WindowsFormsHost](#)
- [Progettare XAML in Visual Studio](#)
- [Considerazioni sul layout per l'elemento WindowsFormsHost](#)
- [Disposizione di controlli Windows Forms nell'esempio WPF](#)
- [Procedura dettagliata: Hosting di controlli Windows Form compositi in WPF](#)
- [Procedura dettagliata: hosting di un controllo WPF composito in Windows Form](#)

Procedura dettagliata: associazione ai dati in applicazioni ibride

10/02/2020 • 10 minutes to read • [Edit Online](#)

Il binding di un'origine dati a un controllo è essenziale per consentire agli utenti di accedere ai dati sottostanti, indipendentemente dal fatto che si utilizzino Windows Forms o WPF. Questa procedura dettagliata illustra come è possibile usare data binding in applicazioni ibride che includono controlli sia Windows Forms che WPF.

Le attività illustrate nella procedura dettagliata sono le seguenti:

- Creazione del progetto.
- Definizione del modello dati.
- Specifica del layout del form.
- Specifica delle associazioni dati.
- Visualizzazione dei dati usando l'interoperabilità.
- Aggiunta dell'origine dati al progetto.
- Associazione all'origine dati.

Per un listato di codice completo delle attività illustrate in questa procedura dettagliata, vedere [esempio di associazione dati in applicazioni ibride](#).

Al termine, si conosceranno le funzionalità di associazione dati nelle applicazioni ibride.

Prerequisites

Per completare questa procedura dettagliata, è necessario disporre dei componenti seguenti:

- Visual Studio.
- Accesso al database di esempio Northwind in esecuzione in Microsoft SQL Server.

Creazione del progetto

Per creare e impostare il progetto

1. Creare un progetto di applicazione WPF denominato `WPFWithWFAndDatabinding`.
2. In Esplora soluzioni aggiungere riferimenti agli assembly seguenti.
 - `WindowsFormsIntegration`
 - `System.Windows.Forms`
3. Aprire `MainWindow.xaml` in WPF Designer.
4. Nell'elemento `Window` aggiungere il mapping degli spazi dei nomi Windows Forms seguente.

```
xmlns:wf="clr-namespace:System.Windows.Forms;assembly=System.Windows.Forms"
```

5. Denominare l'elemento predefinito `Grid` `mainGrid` assegnando la proprietà `Name`.

```
<Grid x:Name="mainGrid">
```

Definizione del modello dati

L'elenco principale dei clienti viene visualizzato in un controllo [ListBox](#). Nell'esempio di codice seguente viene definito un oggetto [DataTemplate](#) denominato [ListItemsTemplate](#) che controlla la struttura ad albero visuale del controllo [ListBox](#). Questo [DataTemplate](#) viene assegnato alla proprietà [ItemTemplate](#) del controllo [ListBox](#).

Per definire il modello di dati

- Copiare il codice XAML seguente nella dichiarazione dell'elemento [Grid](#).

```
<Grid.Resources>
    <DataTemplate x:Key="ListItemsTemplate">
        <TextBlock Text="{Binding Path=ContactName}"/>
    </DataTemplate>
</Grid.Resources>
```

Specifica del layout del form

Il layout del form è definito da una griglia con tre righe e tre colonne. per identificare ogni colonna della tabella Customers sono disponibili controlli [Label](#).

Per impostare il layout Grid

- Copiare il codice XAML seguente nella dichiarazione dell'elemento [Grid](#).

```
<Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
</Grid.RowDefinitions>

<Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto"/>
    <ColumnDefinition Width="Auto"/>
    <ColumnDefinition Width="Auto"/>
</Grid.ColumnDefinitions>
```

Per impostare i controlli Label

- Copiare il codice XAML seguente nella dichiarazione dell'elemento [Grid](#).

```
<StackPanel Orientation="Vertical" Grid.Row="0" Grid.Column="1">
    <Label Margin="20,38,5,2">First Name:</Label>
    <Label Margin="20,0,5,2">Company Name:</Label>
    <Label Margin="20,0,5,2">Phone:</Label>
    <Label Margin="20,0,5,2">Address:</Label>
    <Label Margin="20,0,5,2">City:</Label>
    <Label Margin="20,0,5,2">Region:</Label>
    <Label Margin="20,0,5,2">Postal Code:</Label>
</StackPanel>
```

Specifica delle associazioni dati

L'elenco principale dei clienti viene visualizzato in un controllo [ListBox](#). Il [ListItemsTemplate](#) collegato associa un controllo [TextBlock](#) al campo [ContactName](#) dal database.

I dettagli di ogni record cliente vengono visualizzati in diversi controlli **TextBox**.

Per specificare le associazioni dati

- Copiare il codice XAML seguente nella dichiarazione dell'elemento **Grid**.

La classe **Binding** associa i controlli **TextBox** ai campi appropriati nel database.

```
<StackPanel Orientation="Vertical" Grid.Row="0" Grid.Column="0">
    <Label Margin="20,5,5,0">List of Customers:</Label>
    <ListBox x:Name="listBox1" Height="200" Width="200" HorizontalAlignment="Left"
        ItemTemplate="{StaticResource ListItemsTemplate}" IsSynchronizedWithCurrentItem="True"
        Margin="20,5,5,5"/>
</StackPanel>

<StackPanel Orientation="Vertical" Grid.Row="0" Grid.Column="2">
    <TextBox Margin="5,38,5,2" Width="200" Text="{Binding Path=ContactName}"/>
    <TextBox Margin="5,0,5,2" Width="200" Text="{Binding Path=CompanyName}"/>
    <TextBox Margin="5,0,5,2" Width="200" Text="{Binding Path=Phone}"/>
    <TextBox Margin="5,0,5,2" Width="200" Text="{Binding Path=Address}"/>
    <TextBox Margin="5,0,5,2" Width="200" Text="{Binding Path=City}"/>
    <TextBox Margin="5,0,5,2" Width="30" HorizontalAlignment="Left" Text="{Binding Path=Region}"/>
    <TextBox Margin="5,0,5,2" Width="50" HorizontalAlignment="Left" Text="{Binding Path=PostalCode}"/>
</StackPanel>
```

Visualizzazione dei dati utilizzando l'interoperabilità

Gli ordini corrispondenti al cliente selezionato vengono visualizzati in un controllo

System.Windows.Forms.DataGridView denominato `dataGridView1`. Il controllo `dataGridView1` viene associato all'origine dati nel file code-behind. Un controllo **WindowsFormsHost** è l'elemento padre di questo controllo Windows Forms.

Per visualizzare i dati nel controllo **DataGridView**

- Copiare il codice XAML seguente nella dichiarazione dell'elemento **Grid**.

```
<WindowsFormsHost Grid.Row="1" Grid.Column="0" Grid.ColumnSpan="3" Margin="20,5,5,5" Height="300">
    <wf:DataGridView x:Name="dataGridView1"/>
</WindowsFormsHost>
```

Aggiunta dell'origine dati al progetto

Con Visual Studio è possibile aggiungere facilmente un'origine dati al progetto. Con questa procedura vengono aggiunti dati fortemente tipizzati al progetto. Vengono anche aggiunte altre classi di supporto, ad esempio adattatori di tabelle.

Per aggiungere l'origine dati

1. Scegliere **Aggiungi nuova origine dati** dal menu **dati** .
2. Nella **Configurazione guidata origine dati** creare una connessione al database Northwind utilizzando un set di dati. Per altre informazioni, vedere [How to: Connect to Data in a Database](#).
3. Quando viene richiesta la **Configurazione guidata origine dati**, salvare la stringa di connessione come `NorthwindConnectionString` .
4. Quando viene richiesto di scegliere gli oggetti di database, selezionare le tabelle `Customers` e `Orders` e denominare il set di dati generato `NorthwindDataSet` .

Associazione all'origine dati

Il componente [System.Windows.Forms.BindingSource](#) fornisce un'interfaccia uniforme per l'origine dati dell'applicazione. L'associazione all'origine dati è implementata nel file code-behind.

Per associare l'origine dati

1. Aprire il file code-behind, denominato MainWindow.xaml.vb o MainWindow.xaml.cs.
2. Copiare il codice seguente nella definizione della classe `MainWindow`.

Questo codice dichiara il componente [BindingSource](#) e le classi helper associate che si connettono al database.

```
private System.Windows.Forms.BindingSource nwBindingSource;
private NorthwindDataSet nwDataSet;
private NorthwindDataSetTableAdapters.CustomersTableAdapter customersTableAdapter =
    new NorthwindDataSetTableAdapters.CustomersTableAdapter();
private NorthwindDataSetTableAdapters.OrdersTableAdapter ordersTableAdapter =
    new NorthwindDataSetTableAdapters.OrdersTableAdapter();
```

```
Private nwBindingSource As System.Windows.Forms.BindingSource
Private nwDataSet As NorthwindDataSet
Private customersTableAdapter As New NorthwindDataSetTableAdapters.CustomersTableAdapter()
Private ordersTableAdapter As New NorthwindDataSetTableAdapters.OrdersTableAdapter()
```

3. Copiare il seguente codice nel costruttore.

Questo codice crea e inizializza il componente [BindingSource](#).

```
public MainWindow()
{
    InitializeComponent();

    // Create a DataSet for the Customers data.
    this.nwDataSet = new NorthwindDataSet();
    this.nwDataSet.DataSetName = "nwDataSet";

    // Create a BindingSource for the Customers data.
    this.nwBindingSource = new System.Windows.Forms.BindingSource();
    this.nwBindingSource.DataMember = "Customers";
    this.nwBindingSource.DataSource = this.nwDataSet;
}
```

```
Public Sub New()
    InitializeComponent()

    ' Create a DataSet for the Customers data.
    Me.nwDataSet = New NorthwindDataSet()
    Me.nwDataSet.DataSetName = "nwDataSet"

    ' Create a BindingSource for the Customers data.
    Me.nwBindingSource = New System.Windows.Forms.BindingSource()
    Me.nwBindingSource.DataMember = "Customers"
    Me.nwBindingSource.DataSource = Me.nwDataSet

End Sub
```

4. Aprire MainWindow.xaml.

5. Nella visualizzazione progettazione o nella visualizzazione XAML Selezionare l'elemento **Window**.
6. Nella Finestra Proprietà fare clic sulla scheda **eventi** .
7. Fare doppio clic sull'evento **Loaded**.
8. Copiare il codice seguente nel gestore eventi **Loaded**.

Questo codice assegna il componente **BindingSource** come contesto dati e popola gli oggetti **customers** e **Orders** adapter.

```

private void Window_Loaded(object sender, RoutedEventArgs e)
{
    // Fill the Customers table adapter with data.
    this.customersTableAdapter.ClearBeforeFill = true;
    this.customersTableAdapter.Fill(this.nwDataSet.Customers);

    // Fill the Orders table adapter with data.
    this.ordersTableAdapter.Fill(this.nwDataSet.Orders);

    // Assign the BindingSource to
    // the data context of the main grid.
    this.mainGrid.DataContext = this.nwBindingSource;

    // Assign the BindingSource to
    // the data source of the list box.
    this.listBox1.ItemsSource = this.nwBindingSource;

    // Because this is a master/details form, the DataGridView
    // requires the foreign key relating the tables.
    this.dataGridView1.DataSource = this.nwBindingSource;
    this.dataGridView1.DataMember = "FK_Orders_Customers";

    // Handle the currency management aspect of the data models.
    // Attach an event handler to detect when the current item
    // changes via the WPF ListBox. This event handler synchronizes
    // the list collection with the BindingSource.
    //

    BindingListCollectionView cv = CollectionViewSource.GetDefaultView(
        this.nwBindingSource) as BindingListCollectionView;

    cv.CurrentChanged += new EventHandler(WPF_CurrentChanged);
}

```

```

Private Sub Window_Loaded( _
    ByVal sender As Object, _
    ByVal e As RoutedEventArgs)

    ' Fill the Customers table adapter with data.
    Me.customersTableAdapter.ClearBeforeFill = True
    Me.customersTableAdapter.Fill(Me.nwDataSet.Customers)

    ' Fill the Orders table adapter with data.
    Me.ordersTableAdapter.Fill(Me.nwDataSet.Orders)

    ' Assign the BindingSource to
    ' the data context of the main grid.
    Me.mainGrid.DataContext = Me.nwBindingSource

    ' Assign the BindingSource to
    ' the data source of the list box.
    Me.listBox1.ItemsSource = Me.nwBindingSource

    ' Because this is a master/details form, the DataGridView
    ' requires the foreign key relating the tables.
    Me.dataGridView1.DataSource = Me.nwBindingSource
    Me.dataGridView1.DataMember = "FK_Orders_Customers"

    ' Handle the currency management aspect of the data models.
    ' Attach an event handler to detect when the current item
    ' changes via the WPF ListBox. This event handler synchronizes
    ' the list collection with the BindingSource.
    '

    Dim cv As BindingListCollectionView = _
        CollectionViewSource.GetDefaultView(Me.nwBindingSource)

    AddHandler cv.CurrentChanged, AddressOf WPF_CurrentChanged

End Sub

```

9. Copiare il codice seguente nella definizione della classe `MainWindow`.

Questo metodo gestisce l'evento `CurrentChanged` e aggiorna l'elemento corrente del data binding.

```

// This event handler updates the current item
// of the data binding.
void WPF_CurrentChanged(object sender, EventArgs e)
{
    BindingListCollectionView cv = sender as BindingListCollectionView;
    this.nwBindingSource.Position = cv.CurrentPosition;
}

```

```

' This event handler updates the current item
' of the data binding.
Private Sub WPF_CurrentChanged(ByVal sender As Object, ByVal e As EventArgs)
    Dim cv As BindingListCollectionView = sender
    Me.nwBindingSource.Position = cv.CurrentPosition
End Sub

```

10. Premere F5 per compilare ed eseguire l'applicazione.

Vedere anche

- [ElementHost](#)

- WindowsFormsHost
- Progettare XAML in Visual Studio
- Esempio di associazione dati in applicazioni ibride
- Procedura dettagliata: Hosting di controlli Windows Form compositi in WPF
- Procedura dettaglia: hosting di un controllo WPF composito in Windows Form

Procedura dettagliata: ospitare un controllo composito 3D WPF in Windows Forms

31/01/2020 • 6 minutes to read • [Edit Online](#)

In questa procedura dettagliata viene illustrato come creare un controllo WPF composito e ospitarlo in controlli Windows Forms e form usando il controllo [ElementHost](#).

In questa procedura dettagliata verrà implementato un WPF [UserControl](#) contenente due controlli figlio. Il [UserControl](#) visualizza un cono tridimensionale (3D). Il rendering degli oggetti 3D è molto più semplice con il WPF rispetto a Windows Forms. È pertanto consigliabile ospitare una classe WPF [UserControl](#) per creare grafica 3D in Windows Forms.

Le attività illustrate nella procedura dettagliata sono le seguenti:

- Creazione della [UserControlWPF](#).
- Creazione del progetto host Windows Forms.
- Hosting della [UserControlWPF](#).

Prerequisiti

Per completare la procedura dettagliata, è necessario disporre dei componenti seguenti:

- Visual Studio 2017

Creare UserControl

1. Creare un progetto **libreria di controlli utente WPF** denominato `HostingWpfUserControlInWf`.
2. Aprire `UserControl1.xaml` in WPF Designer.
3. Sostituire il codice generato con il codice seguente:

```
<UserControl x:Class="HostingWpfUserControlInWf.UserControl1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    >

    <Grid>

        <!-- Place a Label control at the top of the view. -->
        <Label
            HorizontalAlignment="Center"
            TextBlock.TextAlignment="Center"
            FontSize="20"
            Foreground="Red"
            Content="Model: Cone"/>

        <!-- Viewport3D is the rendering surface. -->
        <Viewport3D Name="myViewport" >

            <!-- Add a camera. -->
            <Viewport3D.Camera>
                <PerspectiveCamera
                    FarPlaneDistance="20"
                    LookDirection="0,0,1"
```

```

        UpDirection="0,1,0"
        NearPlaneDistance="1"
        Position="0,0,-3"
        FieldOfView="45" />
    </Viewport3D.Camera>

    <!-- Add models. -->
    <Viewport3D.Children>

        <ModelVisual3D>
            <ModelVisual3D.Content>

                <Model3DGroup>
                    <Model3DGroup.Children>

                        <!-- Lights, MeshGeometry3D and DiffuseMaterial objects are added to the
ModelVisual3D. -->
                        <DirectionalLight Color="#FFFFFF" Direction="3,-4,5" />

                        <!-- Define a red cone. -->
                        <GeometryModel3D>

                            <GeometryModel3D.Geometry>
                                <MeshGeometry3D>

                                    Positions="0.293893 -0.5 0.404509  0.475528 -0.5 0.154509  0 0.5 0  0.475528 -0.5 0.154509  0 0.5 0
0 0.5 0  0.475528 -0.5 0.154509  0.475528 -0.5 -0.154509  0 0.5 0  0.475528 -0.5 -0.154509  0 0.5 0
0.5 0  0.475528 -0.5 -0.154509  0.293893 -0.5 -0.404509  0 0.5 0  0.293893 -0.5 -0.404509  0 0.5 0
0 0.5 0  0.293893 -0.5 -0.404509  0 -0.5 -0.5  0 0.5 0  0 -0.5 -0.5  0 0.5 0  0 0.5 0  0 -0.5 -0.5 -
0.293893 -0.5 -0.404509  0 0.5 0  -0.293893 -0.5 -0.404509  0 0.5 0  0 0.5 0  -0.293893 -0.5 -0.404509
-0.475528 -0.5 -0.154509  0 0.5 0  -0.475528 -0.5 -0.154509  0 0.5 0  0 0.5 0  -0.475528 -0.5 -0.154509
-0.475528 -0.5 0.154509  0 0.5 0  -0.475528 -0.5 0.154509  0 0.5 0  0 0.5 0  -0.475528 -0.5 0.154509 -
0.293892 -0.5 0.404509  0 0.5 0  -0.293892 -0.5 0.404509  0 0.5 0  0 0.5 0  -0.293892 -0.5 0.404509  0 -
0.5 0.5 0  0 -0.5 0.5  0 0.5 0  0 0.5 0  0 -0.5 0.5  0 0.5 0  0 0.5 0  0 0.5 0  "
                                    Normals="0.7236065,0.4472139,0.5257313  0.2763934,0.4472138,0.8506507  0.5308242,0.4294462,0.7306172
0.2763934,0.4472138,0.8506507  0,0.4294458,0.9030925  0.5308242,0.4294462,0.7306172
0.2763934,0.4472138,0.8506507  -0.2763934,0.4472138,0.8506507  0,0.4294458,0.9030925 -
0.2763934,0.4472138,0.8506507  -0.5308242,0.4294462,0.7306172  0,0.4294458,0.9030925 -
0.2763934,0.4472138,0.8506507  -0.7236065,0.4472139,0.5257313  -0.5308242,0.4294462,0.7306172 -
0.7236065,0.4472139,0.5257313  -0.858892,0.429446,0.279071  -0.5308242,0.4294462,0.7306172 -
0.7236065,0.4472139,0.5257313  -0.8944269,0.4472139,0  -0.858892,0.429446,0.279071 -
0.8944269,0.4472139,0  -0.858892,0.429446,-0.279071  -0.858892,0.429446,0.279071  -0.8944269,0.4472139,0
-0.7236065,0.4472139,-0.5257313  -0.858892,0.429446,-0.279071  -0.7236065,0.4472139,-0.5257313 -
0.5308242,0.4294462,-0.7306172  -0.858892,0.429446,-0.279071  -0.7236065,0.4472139,-0.5257313 -
0.2763934,0.4472138,-0.8506507  -0.5308242,0.4294462,-0.7306172  -0.2763934,0.4472138,-0.8506507
0,0.4294458,-0.9030925  -0.5308242,0.4294462,-0.7306172  -0.2763934,0.4472138,-0.8506507
0.2763934,0.4472138,-0.8506507  0,0.4294458,-0.9030925  0.2763934,0.4472138,-0.8506507
0.5308249,0.4294459,-0.7306169  0,0.4294458,-0.9030925  0.2763934,0.4472138,-0.8506507
0.7236068,0.4472141,-0.5257306  0.5308249,0.4294459,-0.7306169  0.7236068,0.4472141,-0.5257306
0.8588922,0.4294461,-0.27907  0.5308249,0.4294459,-0.7306169  0.7236068,0.4472141,-0.5257306
0.8944269,0.4472139,0  0.8588922,0.4294461,-0.27907  0.8944269,0.4472139,0  0.858892,0.429446,0.279071
0.8588922,0.4294461,-0.27907  0.8944269,0.4472139,0  0.7236065,0.4472139,0.5257313
0.858892,0.429446,0.279071  0.7236065,0.4472139,0.5257313  0.5308242,0.4294462,0.7306172
0.858892,0.429446,0.279071  "
                                            TriangleIndices="0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 " />
                            </GeometryModel3D.Geometry>

                            <GeometryModel3D.Material>
                                <DiffuseMaterial>
                                    <DiffuseMaterial.Brush>
                                        <SolidColorBrush
                                            Color="Red"
                                            Opacity="1.0"/>
                                    </DiffuseMaterial.Brush>
                                </DiffuseMaterial>
                            </GeometryModel3D.Material>
                        </GeometryModel3D>
                    </Model3DGroup.Children>
                </Model3DGroup>
            </ModelVisual3D.Content>
        </ModelVisual3D>
    </Viewport3D.Children>
</Viewport3D>
```

```

        </Model3DGroup.Children>
    </Model3DGroup>

    </ModelVisual3D.Content>

    </ModelVisual3D>

    </Viewport3D.Children>

    </Viewport3D>
</Grid>

</UserControl>

```

Questo codice definisce una [System.Windows.Controls.UserControl](#) che contiene due controlli figlio. Il primo controllo figlio è un controllo [System.Windows.Controls.Label](#); il secondo è un controllo [Viewport3D](#) che visualizza un cono 3D.

Creare il progetto host

1. Aggiungere un progetto di **App Windows Forms (.NET Framework)** denominato [WpfUserControlHost](#) alla soluzione.
2. In **Esplora soluzioni** aggiungere un riferimento all'assembly WindowsFormsIntegration, denominato WindowsFormsIntegration.dll.
3. Aggiungere i riferimenti agli assembly WPF seguenti:
 - PresentationCore
 - PresentationFramework
 - WindowsBase
4. Aggiungere un riferimento al progetto [HostingWpfUserControlInWf](#).
5. In Esplora soluzioni impostare il progetto [WpfUserControlHost](#) come progetto di avvio.

Ospitare UserControl

1. Nella Progettazione Windows Form aprire Form1.
2. Nella Finestra Proprietà fare clic su **eventi**, quindi fare doppio clic sull'evento [Load](#) per creare un gestore eventi.
Viene aperto l'editor di codice per il gestore dell'evento [Form1_Load](#) appena generato.
3. Sostituire il codice in Form1.cs con il codice seguente.

Il gestore dell'evento [Form1_Load](#) crea un'istanza di [UserControl1](#) e aggiunge alla raccolta dei controlli figlio del controllo [ElementHost](#). Il controllo [ElementHost](#) viene aggiunto alla raccolta di controlli figlio del form.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

using System.Windows.Forms.Integration;

namespace WpfUserControlHost
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // Create the ElementHost control for hosting the
            // WPF UserControl.
            ElementHost host = new ElementHost();
            host.Dock = DockStyle.Fill;

            // Create the WPF UserControl.
            HostingWpfUserControlInWf.UserControl1 uc =
                new HostingWpfUserControlInWf.UserControl1();

            // Assign the WPF UserControl to the ElementHost control's
            // Child property.
            host.Child = uc;

            // Add the ElementHost control to the form's
            // collection of child controls.
            this.Controls.Add(host);
        }
    }
}
```

```

Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Data
Imports System.Drawing
Imports System.Text
Imports System.Windows.Forms

Imports System.Windows.Forms.Integration

Public Class Form1
    Inherits Form

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        ' Create the ElementHost control for hosting the
        ' WPF UserControl.
        Dim host As New ElementHost()
        host.Dock = DockStyle.Fill

        ' Create the WPF UserControl.
        Dim uc As New HostingWpfUserControlInWf.UserControl1()

        ' Assign the WPF UserControl to the ElementHost control's
        ' Child property.
        host.Child = uc

        ' Add the ElementHost control to the form's
        ' collection of child controls.
        Me.Controls.Add(host)
    End Sub

End Class

```

4. Premere **F5** per compilare ed eseguire l'applicazione.

Vedere anche

- [ElementHost](#)
- [WindowsFormsHost](#)
- [Progettare XAML in Visual Studio](#)
- [Procedura dettaglia: hosting di un controllo WPF composito in Windows Form](#)
- [Procedura dettagliata: Hosting di controlli Windows Form composti in WPF](#)
- [Hosting di un controllo composito WPF nell'esempio Windows Forms](#)

Procedura dettagliata: hosting di controlli composti di WPF in Windows Form

31/01/2020 • 29 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) fornisce un ambiente completo per la creazione di applicazioni. Tuttavia, quando si ha un investimento sostanziale nel codice Windows Forms, può essere più efficace estendere la Windows Forms Application esistente con WPF anziché riscrivere da zero. Uno scenario comune è quando si desidera incorporare uno o più controlli implementati con WPF all'interno del Windows Forms Application. Per ulteriori informazioni sulla personalizzazione dei controlli WPF, vedere [personalizzazione del controllo](#).

Questa procedura dettagliata illustra un'applicazione che ospita un controllo WPF composito per eseguire l'immissione di dati in una Windows Forms Application. Il controllo composito è compresso in una DLL. Questa procedura generale può essere estesa ad applicazioni e controlli più complessi. Questa procedura dettagliata è progettata per essere pressoché identica per quanto riguarda aspetto e funzionalità per [la procedura dettagliata: hosting di un controllo Windows Forms composito in WPF](#). La principale differenza è che lo scenario di hosting è invertito.

La procedura guidata è suddivisa in due sezioni. Nella prima sezione viene brevemente descritta l'implementazione del controllo WPF composito. La seconda sezione illustra in dettaglio come ospitare il controllo composito in una Windows Forms Application, ricevere eventi dal controllo e accedere ad alcune delle proprietà del controllo.

Le attività illustrate nella procedura dettagliata sono le seguenti:

- Implementazione del controllo composito WPF.
- Implementazione dell'applicazione host Windows Forms.

Per un listato di codice completo delle attività illustrate in questa procedura dettagliata, vedere [hosting di un controllo composito WPF nell'esempio Windows Forms](#).

Prerequisiti

Per completare la procedura dettagliata, è necessario Visual Studio.

Implementazione del controllo composito WPF

Il WPF controllo composito usato in questo esempio è un semplice form di immissione dati che accetta il nome e l'indirizzo dell'utente. Quando l'utente fa clic su uno dei due pulsanti per indicare che l'attività è completata, il controllo genera un evento personalizzato per restituire tali informazioni all'host. La figura seguente mostra il controllo sottoposto a rendering.

Nell'immagine seguente viene illustrato un controllo composito WPF:



Creazione del progetto

Per avviare il progetto:

1. Avviare Visual Studio e aprire la finestra di dialogo **nuovo progetto** .
2. In Visual C# e nella Categoria Windows selezionare il modello **libreria di controlli utente WPF** .
3. Assegnare il nome `MyControls` al nuovo progetto.
4. Per il percorso specificare una cartella di primo livello con un nome appropriato, ad esempio `WindowsFormsHostingWpfControl` . Successivamente, si immetterà l'applicazione host in questa cartella.
5. Fare clic su **OK** per creare il progetto. Il progetto predefinito contiene un singolo controllo denominato `UserControl1` .
6. In Esplora soluzioni rinominare `UserControl1` `MyControl1` .

Il progetto dovrebbe includere riferimenti alle DLL di sistema seguenti. Se alcune di queste DLL non sono incluse per impostazione predefinita, aggiungerle al progetto.

- PresentationCore
- PresentationFramework
- System
- WindowsBase

Creazione dell'interfaccia utente

Il interfaccia utente per il controllo composito viene implementato con Extensible Application Markup Language (XAML). Il Interfaccia utente di controllo composto è costituito da cinque elementi di `TextBox`. Ogni elemento `TextBox` dispone di un elemento `TextBlock` associato che funge da etichetta. Sono presenti due `Button` elementi nella parte inferiore, **OK** e **Cancel**. Quando l'utente fa clic su uno di questi pulsanti, il controllo genera un evento personalizzato per restituire le informazioni all'host.

Layout di base

I vari elementi di Interfaccia utente sono contenuti in un elemento `Grid`. È possibile utilizzare `Grid` per disporre il contenuto del controllo composito nello stesso modo in cui si utilizzerà un elemento `Table` in HTML. WPF dispone anche di un elemento `Table`, ma `Grid` è più leggero e più adatto per le semplici attività di layout.

Il codice XAML seguente illustra il layout di base. Questo codice XAML definisce la struttura complessiva del controllo specificando il numero di colonne e righe nell'elemento `Grid`.

In `MyControl1.xaml` sostituire il codice XAML esistente con il seguente.

```
<Grid xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      x:Class="MyControls.MyControl1"
      Background="#DCDCDC"
      Width="375"
      Height="250"
      Name="rootElement"
      Loaded="Init">

    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition Width="Auto"/>
    </Grid.ColumnDefinitions>

    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>
```

```
</Grid>
```

Aggiunta di elementi **TextBlock** e **TextBox** alla griglia

Per inserire un elemento Interfaccia utente nella griglia, impostare gli attributi [RowProperty](#) e [ColumnProperty](#) dell'elemento sul numero di riga e di colonna appropriato. Ricordare che la numerazione di righe e colonne è in base zero. È possibile fare in modo che un elemento si estenda su più colonne impostando il relativo attributo [ColumnSpanProperty](#). Per ulteriori informazioni sugli elementi di [Grid](#), vedere [Create a Grid element](#).

Il codice XAML seguente Mostra gli elementi **TextBox** e **TextBlock** del controllo composito con i rispettivi attributi [RowProperty](#) e [ColumnProperty](#), impostati per inserire correttamente gli elementi nella griglia.

In MyControl1. XAML aggiungere il codice XAML seguente all'interno dell'elemento **Grid**.

```

<TextBlock Grid.Column="0"
    Grid.Row="0"
    Grid.ColumnSpan="4"
    Margin="10,5,10,0"
    HorizontalAlignment="Center"
    Style="{StaticResource titleText}">Simple WPF Control</TextBlock>

<TextBlock Grid.Column="0"
    Grid.Row="1"
    Style="{StaticResource inlineText}"
    Name="nameLabel">Name</TextBlock>
<TextBox Grid.Column="1"
    Grid.Row="1"
    Grid.ColumnSpan="3"
    Name="txtName"/>

<TextBlock Grid.Column="0"
    Grid.Row="2"
    Style="{StaticResource inlineText}"
    Name="addressLabel">Street Address</TextBlock>
<TextBox Grid.Column="1"
    Grid.Row="2"
    Grid.ColumnSpan="3"
    Name="txtAddress"/>

<TextBlock Grid.Column="0"
    Grid.Row="3"
    Style="{StaticResource inlineText}"
    Name="cityLabel">City</TextBlock>
<TextBox Grid.Column="1"
    Grid.Row="3"
    Width="100"
    Name="txtCity"/>

<TextBlock Grid.Column="2"
    Grid.Row="3"
    Style="{StaticResource inlineText}"
    Name="stateLabel">State</TextBlock>
<TextBox Grid.Column="3"
    Grid.Row="3"
    Width="50"
    Name="txtState"/>

<TextBlock Grid.Column="0"
    Grid.Row="4"
    Style="{StaticResource inlineText}"
    Name="zipLabel">Zip</TextBlock>
<TextBox Grid.Column="1"
    Grid.Row="4"
    Width="100"
    Name="txtZip"/>

```

Applicazione di stili agli elementi dell'interfaccia utente

Molti degli elementi nel form di immissione dati hanno un aspetto simile, che significa che hanno impostazioni identiche per alcune delle loro proprietà. Anziché impostare separatamente gli attributi di ogni elemento, il codice XAML precedente usa **Style** elementi per definire le impostazioni delle proprietà standard per le classi di elementi. Questo approccio riduce la complessità del controllo e consente di modificare l'aspetto di più elementi tramite un unico attributo di stile.

Gli elementi **Style** sono contenuti nella proprietà **Resources** dell'elemento **Grid**, in modo che possano essere usati da tutti gli elementi nel controllo. Se uno stile è denominato, lo si applica a un elemento aggiungendo un **Style** elemento impostato sul nome dello stile. Gli stili che non sono denominati diventano lo stile predefinito per l'elemento. Per ulteriori informazioni sugli stili di WPF, vedere applicazione di stili e modelli.

Il codice XAML seguente Mostra gli elementi **Style** per il controllo composito. Per visualizzare come gli stili vengono applicati agli elementi, vedere il codice XAML precedente. Ad esempio, l'ultimo elemento **TextBlock** ha lo stile `inlineText` e l'ultimo elemento **TextBox** usa lo stile predefinito.

In MyControl1. XAML aggiungere il codice XAML seguente subito dopo l'elemento di avvio **Grid**.

```
<Grid.Resources>
    <Style x:Key="inlineText" TargetType="{x:Type TextBlock}">
        <Setter Property="Margin" Value="10,5,10,0"/>
        <Setter Property="FontWeight" Value="Normal"/>
        <Setter Property="FontSize" Value="12"/>
    </Style>
    <Style x:Key="titleText" TargetType="{x:Type TextBlock}">
        <Setter Property="DockPanel.Dock" Value="Top"/>
        <Setter Property="FontWeight" Value="Bold"/>
        <Setter Property="FontSize" Value="14"/>
        <Setter Property="Margin" Value="10,5,10,0"/>
    </Style>
    <Style TargetType="{x:Type Button}">
        <Setter Property="Margin" Value="10,5,10,0"/>
        <Setter Property="Width" Value="60"/>
    </Style>
    <Style TargetType="{x:Type TextBox}">
        <Setter Property="Margin" Value="10,5,10,0"/>
    </Style>
</Grid.Resources>
```

Aggiunta dei pulsanti OK e Cancel

Gli elementi finali del controllo composito sono gli elementi **Button OK e Cancel**, che occupano le prime due colonne dell'ultima riga del **Grid**. Questi elementi usano un gestore eventi comune, `ButtonClicked` e lo stile **Button** predefinito definito nel codice XAML precedente.

In MyControl1. XAML aggiungere il codice XAML seguente dopo l'ultimo elemento **TextBox**. Il XAML parte del controllo composito è ora completo.

```
<Button Grid.Row="5"
        Grid.Column="0"
        Name="btnOK"
        Click="ButtonClicked">OK</Button>
<Button Grid.Row="5"
        Grid.Column="1"
        Name="btnCancel"
        Click="ButtonClicked">Cancel</Button>
```

Implementazione del file code-behind

Il file code-behind, MyControl1.xaml.cs, implementa tre attività essenziali:

1. Gestisce l'evento che si verifica quando l'utente fa clic su uno dei pulsanti.
2. Recupera i dati dagli elementi **TextBox** e li impacchetta in un oggetto argomento dell'evento personalizzato.
3. Genera l'evento `OnButtonClick` personalizzato, che invia una notifica all'host che l'utente ha terminato e passa i dati all'host.

Il controllo espone inoltre un numero di proprietà di colore e tipo di carattere che consentono di modificare l'aspetto. A differenza della classe **WindowsFormsHost**, che viene usata per ospitare un controllo Windows Forms, la classe **ElementHost** espone solo la proprietà **Background** del controllo. Per mantenere la somiglianza tra questo esempio di codice e l'esempio illustrato in [procedura dettagliata: hosting di un controllo Windows Forms composito in WPF](#), il controllo espone direttamente le proprietà rimanenti.

Struttura di base del file code-behind

Il file code-behind è costituito da un singolo spazio dei nomi, `MyControls`, che conterrà due classi, `MyControl1` e `MyControlEventArgs`.

```
namespace MyControls
{
    public partial class MyControl1 : Grid
    {
        //...
    }
    public class MyControlEventArgs : EventArgs
    {
        //...
    }
}
```

La prima classe, `MyControl1`, è una classe parziale contenente il codice che implementa la funzionalità della Interfaccia utente definita in `MyControl1.xaml`. Quando `MyControl1.xaml` viene analizzato, il XAML viene convertito nella stessa classe parziale e le due classi parziali vengono unite per formare il controllo compilato. Per questo motivo, il nome della classe nel file code-behind deve corrispondere al nome della classe assegnato a `MyControl1.xaml` e deve ereditare dall'elemento radice del controllo. La seconda classe, `MyControlEventArgs`, è una classe di argomenti di evento utilizzata per restituire i dati all'host.

Aprire `MyControl1.xaml.cs`. Modificare la dichiarazione di classe esistente in modo che abbia il nome seguente ed erediti da `Grid`.

```
public partial class MyControl1 : Grid
```

Inizializzazione del controllo

Il codice seguente implementa alcune attività di base:

- Dichiara un evento privato, `OnButtonClick` e il delegato associato, `MyControlEventHandler`.
- Crea diverse variabili globali private che archiviano i dati dell'utente. Questi dati vengono esposti tramite le proprietà corrispondenti.
- Implementa un gestore, `Init`, per l'evento `Loaded` del controllo. Questo gestore inizializza le variabili globali assegnando loro i valori definiti in `MyControl1.xaml`. A tale scopo, usa il `Name` assegnato a un elemento di `TextBlock` tipico, `nameLabel`, per accedere alle impostazioni delle proprietà di tale elemento.

Eliminare il costruttore esistente e aggiungere il codice seguente alla classe `MyControl1`.

```

public delegate void MyControlEventHandler(object sender, MyControlEventArgs args);
public event MyControlEventHandler OnButtonClick;
private FontWeight _fontWeight;
private double _fontSize;
private FontFamily _fontFamily;
private FontStyle _fontStyle;
private SolidColorBrush _foreground;
private SolidColorBrush _background;

private void Init(object sender, EventArgs e)
{
    //They all have the same style, so use nameLabel to set initial values.
    _fontWeight = nameLabel.FontWeight;
    _fontSize = nameLabel.FontSize;
    _fontFamily = nameLabel.FontFamily;
    _fontStyle = nameLabel.FontStyle;
    _foreground = (SolidColorBrush)nameLabel.Foreground;
    _background = (SolidColorBrush)rootElement.Background;
}

```

Gestione degli eventi clic dei pulsanti

L'utente indica che l'attività di immissione dei dati è stata completata facendo clic sul pulsante **OK** o sul pulsante **Annulla**. Entrambi i pulsanti utilizzano lo stesso gestore dell'evento **Click**, **ButtonClicked**. Entrambi i pulsanti hanno un nome, **btnOK** o **btnCancel**, che consente al gestore di determinare su quale pulsante è stato fatto clic esaminando il valore dell'argomento **sender**. Il gestore esegue le operazioni seguenti:

- Crea un **MyControlEventArgs** oggetto che contiene i dati dagli elementi di **TextBox**.
- Se l'utente ha fatto clic sul pulsante **Annulla**, imposta la proprietà **isOk** dell'oggetto **MyControlEventArgs** su **false**.
- Genera l'evento **OnButtonClick** per indicare all'host che l'utente ha terminato e passa di nuovo i dati raccolti.

Aggiungere il codice seguente alla classe **MyControl1**, dopo il metodo **Init**.

```

private void ButtonClicked(object sender, RoutedEventArgs e)
{
    MyControlEventArgs retvals = new MyControlEventArgs(true,
                                                       txtName.Text,
                                                       txtAddress.Text,
                                                       txtCity.Text,
                                                       txtState.Text,
                                                       txtZip.Text);

    if (sender == btnCancel)
    {
        retvals.IsOK = false;
    }
    if (OnButtonClick != null)
        OnButtonClick(this, retvals);
}

```

Creazione di proprietà

Il resto della classe espone semplicemente proprietà che corrispondono alle variabili globali precedentemente illustrate. Quando una proprietà viene modificata, la funzione di accesso impostata modifica l'aspetto del controllo cambiando le proprietà degli elementi corrispondenti e aggiornando le variabili globali sottostanti.

Aggiungere il codice seguente alla classe **MyControl1**.

```

public FontWeight MyControl_FontWeight
{

```

```

        get { return _fontWeight; }
        set
        {
            _fontWeight = value;
            nameLabel.FontWeight = value;
            addressLabel.FontWeight = value;
            cityLabel.FontWeight = value;
            stateLabel.FontWeight = value;
            zipLabel.FontWeight = value;
        }
    }
    public double MyControl_FontSize
    {
        get { return _fontSize; }
        set
        {
            _fontSize = value;
            nameLabel.FontSize = value;
            addressLabel.FontSize = value;
            cityLabel.FontSize = value;
            stateLabel.FontSize = value;
            zipLabel.FontSize = value;
        }
    }
    public FontStyle MyControl_FontStyle
    {
        get { return _fontStyle; }
        set
        {
            _fontStyle = value;
            nameLabel.FontStyle = value;
            addressLabel.FontStyle = value;
            cityLabel.FontStyle = value;
            stateLabel.FontStyle = value;
            zipLabel.FontStyle = value;
        }
    }
    public FontFamily MyControl_FontFamily
    {
        get { return _fontFamily; }
        set
        {
            _fontFamily = value;
            nameLabel.FontFamily = value;
            addressLabel.FontFamily = value;
            cityLabel.FontFamily = value;
            stateLabel.FontFamily = value;
            zipLabel.FontFamily = value;
        }
    }
}

public SolidColorBrush MyControl_Background
{
    get { return _background; }
    set
    {
        _background = value;
        rootElement.Background = value;
    }
}
public SolidColorBrush MyControl_Foreground
{
    get { return _foreground; }
    set
    {
        _foreground = value;
        nameLabel.Foreground = value;
        addressLabel.Foreground = value;
        cityLabel.Foreground = value;
    }
}

```

```
    stateLabel.Foreground = value;
    zipLabel.Foreground = value;
}
}
```

Invio di dati di nuovo all'host

Il componente finale del file è la classe `MyControlEventArgs`, che viene utilizzata per inviare i dati raccolti all'host.

Aggiungere il codice seguente allo spazio dei nomi `MyControls`. L'implementazione è semplice e non verrà ulteriormente illustrata.

```

public class MyControlEventArgs : EventArgs
{
    private string _Name;
    private string _StreetAddress;
    private string _City;
    private string _State;
    private string _Zip;
    private bool _IsOK;

    public MyControlEventArgs(bool result,
                             string name,
                             string address,
                             string city,
                             string state,
                             string zip)
    {
        _IsOK = result;
        _Name = name;
        _StreetAddress = address;
        _City = city;
        _State = state;
        _Zip = zip;
    }

    public string MyName
    {
        get { return _Name; }
        set { _Name = value; }
    }
    public string MyStreetAddress
    {
        get { return _StreetAddress; }
        set { _StreetAddress = value; }
    }
    public string MyCity
    {
        get { return _City; }
        set { _City = value; }
    }
    public string MyState
    {
        get { return _State; }
        set { _State = value; }
    }
    public string MyZip
    {
        get { return _Zip; }
        set { _Zip = value; }
    }
    public bool IsOK
    {
        get { return _IsOK; }
        set { _IsOK = value; }
    }
}

```

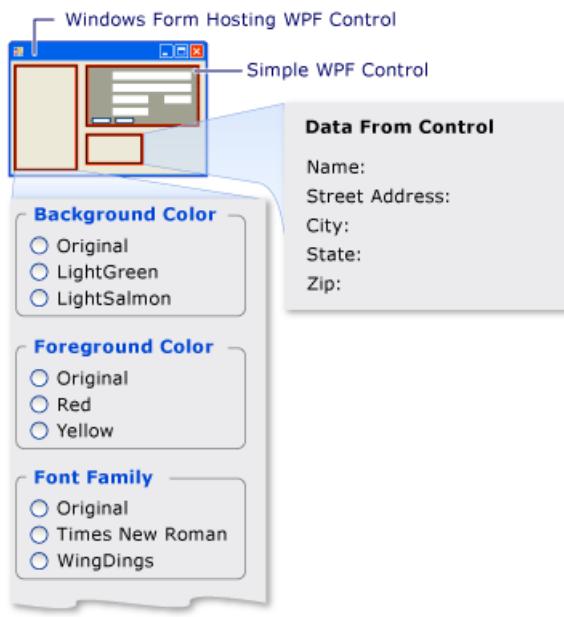
Compila la soluzione. La compilazione genererà una DLL denominata MyControls.dll.

Implementazione dell'applicazione host Windows Forms

L'applicazione host Windows Forms usa un oggetto [ElementHost](#) per ospitare il WPF controllo composito.

L'applicazione gestisce l'evento `OnButtonClick` per ricevere i dati dal controllo composito. L'applicazione presenta anche un insieme di pulsanti di opzione che è possibile utilizzare per modificare l'aspetto del controllo. La figura seguente mostra l'applicazione.

Nell'immagine seguente viene illustrato un controllo composito WPF ospitato in un Windows Forms Application



Creazione del progetto

Per avviare il progetto:

1. Avviare Visual Studio e aprire la finestra di dialogo **nuovo progetto**.
2. In Visual C# e nella Categoria Windows selezionare il modello di **applicazione Windows Forms**.
3. Assegnare il nome `WFHost` al nuovo progetto.
4. Come percorso, specificare la stessa cartella di livello superiore che contiene il progetto MyControls.
5. Fare clic su **OK** per creare il progetto.

È anche necessario aggiungere riferimenti alla DLL che contiene `MyControl1` e altri assembly.

1. Fare clic con il pulsante destro del mouse sul nome del progetto in Esplora soluzioni e scegliere **Aggiungi riferimento**.
2. Fare clic sulla scheda **Sfoglia** e selezionare la cartella che contiene i controlli. dll. In questa procedura dettagliata la cartella è `MyControls\bin\Debug`.
3. Selezionare controllo. dll e quindi fare clic su **OK**.
4. Aggiungere riferimenti agli assembly indicati di seguito.
 - PresentationCore
 - PresentationFramework
 - System.Xaml
 - WindowsBase
 - WindowsFormsIntegration

Implementazione dell'interfaccia utente per l'applicazione

L'interfaccia utente per l'applicazione Windows Form contiene vari controlli per interagire con il controllo composito WPF.

1. Aprire Form1 in Progettazione Windows Form.

2. Ingrandire il form per cui inserire i controlli.
3. Nell'angolo in alto a destra del form aggiungere un controllo **System.Windows.Forms.Panel** per mantenere il controllo composito WPF.
4. Aggiungere i controlli di **System.Windows.Forms.GroupBox** seguenti al form.

NAME	TESTO
groupBox1	Colore di sfondo
groupBox2	Colore primo piano
groupBox3	Dimensioni carattere
groupBox4	Famiglia di caratteri
groupBox5	Stile carattere
groupBox6	Spessore carattere
groupBox7	Dati dal controllo

5. Aggiungere i controlli di **System.Windows.Forms.RadioButton** seguenti ai controlli **System.Windows.Forms.GroupBox**.

GROUPBOX	NAME	TESTO
groupBox1	radioBackgroundOriginal	Originale
groupBox1	radioBackgroundLightGreen	LightGreen
groupBox1	radioBackgroundLightSalmon	LightSalmon
groupBox2	radioForegroundOriginal	Originale
groupBox2	radioForegroundRed	Red
groupBox2	radioForegroundYellow	Yellow
groupBox3	radioSizeOriginal	Originale
groupBox3	radioSizeTen	10
groupBox3	radioSizeTwelve	12
groupBox4	radioFamilyOriginal	Originale
groupBox4	radioFamilyTimes	Times New Roman
groupBox4	radioFamilyWingDings	WingDings
groupBox5	radioStyleOriginal	Normale

GROUPBOX	NAME	TESTO
groupBox5	radioStyleItalic	Corsivo
groupBox6	radioWeightOriginal	Originale
groupBox6	radioWeightBold	Grassetto

6. Aggiungere i controlli di [System.Windows.Forms.Label](#) seguenti all'ultima [System.Windows.Forms.GroupBox](#). Questi controlli visualizzano i dati restituiti dal controllo WPF composito.

GROUPBOX	NAME	TESTO
groupBox7	lblName	Nome:
groupBox7	lblAddress	Indirizzo:
groupBox7	lblCity	Città:
groupBox7	lblState	Stato:
groupBox7	lblZip	CAP:

Inizializzazione del form

In genere si implementa il codice host nel gestore dell'evento [Load](#) del modulo. Il codice seguente illustra il gestore dell'evento [Load](#), un gestore per l'evento [Loaded](#) del controllo WPF composito e le dichiarazioni per diverse variabili globali usate in un secondo momento.

Nella Progettazione Windows Form fare doppio clic sul form per creare un gestore dell'evento [Load](#). Nella parte superiore di Form1.cs aggiungere le istruzioni [using](#) seguenti.

```
using System.Windows;
using System.Windows.Forms.Integration;
using System.Windows.Media;
```

Sostituire il contenuto della classe [Form1](#) esistente con il codice seguente.

```

private ElementHost ctrlHost;
private MyControls.MyControl1 wpfAddressCtrl;
System.Windows.FontWeight initFontWeight;
double initFontSize;
System.Windows.FontStyle initFontStyle;
System.Windows.Media.SolidColorBrush initBackBrush;
System.Windows.Media.SolidColorBrush initForeBrush;
System.Windows.Media.FontFamily initFontFamily;

public Form1()
{
    InitializeComponent();
}

private void Form1_Load(object sender, EventArgs e)
{
    ctrlHost = new ElementHost();
    ctrlHost.Dock = DockStyle.Fill;
    panel1.Controls.Add(ctrlHost);
    wpfAddressCtrl = new MyControls.MyControl1();
    wpfAddressCtrl.InitializeComponent();
    ctrlHost.Child = wpfAddressCtrl;

    wpfAddressCtrl.OnButtonClick +=
        new MyControls.MyControl1.MyControlEventHandler(
            avAddressCtrl_OnButtonClick);
    wpfAddressCtrl.Loaded += new RoutedEventHandler(
        avAddressCtrl_Loaded);
}

void avAddressCtrl_Loaded(object sender, EventArgs e)
{
    initBackBrush = (SolidColorBrush)wpfAddressCtrl.MyControl_Background;
    initForeBrush = wpfAddressCtrl.MyControl_Foreground;
    initFontFamily = wpfAddressCtrl.MyControl_FontFamily;
    initFontSize = wpfAddressCtrl.MyControl_FontSize;
    initFontWeight = wpfAddressCtrl.MyControl_FontWeight;
    initFontStyle = wpfAddressCtrl.MyControl_FontStyle;
}

```

Il metodo `Form1_Load` nel codice precedente illustra la procedura generale per l'hosting di un controllo WPF:

1. Creare un nuovo oggetto `ElementHost`.
2. Impostare la proprietà `Dock` del controllo su `DockStyle.Fill`.
3. Aggiungere il controllo `ElementHost` alla raccolta di `Controls` del controllo `Panel`.
4. Creare un'istanza del controllo WPF.
5. Ospitare il controllo composito nel form assegnando il controllo alla proprietà `Child` del controllo `ElementHost`.

Le due righe rimanenti nel metodo `Form1_Load` alleghino i gestori a due eventi di controllo:

- `OnButtonClick` è un evento personalizzato che viene generato dal controllo composito quando l'utente fa clic sul pulsante **OK** o **Annulla**. Si gestisce l'evento per ottenere la risposta dell'utente e per raccogliere i dati specificati dall'utente.
- `Loaded` è un evento standard generato da un controllo WPF quando viene caricato completamente. L'evento è usato qui poiché nell'esempio è necessario inizializzare diverse variabili globali usando le proprietà del controllo. Al momento dell'evento `Load` del modulo, il controllo non è completamente caricato e tali valori sono ancora impostati su `null`. Prima di poter accedere a tali proprietà, è necessario

attendere fino a quando non si verifica l'evento **Loaded** del controllo.

Il gestore eventi **Loaded** è illustrato nel codice precedente. Il gestore **OnButtonClick** viene illustrato nella sezione successiva.

Gestione dell'evento **OnButtonClick**

L'evento **OnButtonClick** si verifica quando l'utente fa clic sul pulsante **OK** o **Annulla**.

Il gestore eventi controlla il campo **IsOK** dell'argomento dell'evento per determinare il pulsante selezionato. Le variabili di *dati* **lbl** corrispondono ai controlli **Label** descritti in precedenza. Se l'utente fa clic sul pulsante **OK**, i dati dei controlli **TextBox** del controllo vengono assegnati al controllo **Label** corrispondente. Se l'utente fa clic su **Annulla**, i valori **Text** vengono impostati sulle stringhe predefinite.

Aggiungere il seguente codice del gestore dell'evento click del pulsante alla classe **Form1**.

```
void avAddressCtrl1_OnButtonClick(
    object sender,
    MyControls.MyControl1.MyControlEvents args)
{
    if (args.IsOK)
    {
        lblAddress.Text = "Street Address: " + args.MyStreetAddress;
        lblCity.Text = "City: " + args.MyCity;
        lblName.Text = "Name: " + args.MyName;
        lblState.Text = "State: " + args.MyState;
        lblZip.Text = "Zip: " + args.MyZip;
    }
    else
    {
        lblAddress.Text = "Street Address: ";
        lblCity.Text = "City: ";
        lblName.Text = "Name: ";
        lblState.Text = "State: ";
        lblZip.Text = "Zip: ";
    }
}
```

Compilare ed eseguire l'applicazione. Aggiungere il testo nel controllo composito WPF, quindi fare clic su **OK**. Il testo viene visualizzato nelle etichette. A questo punto, il codice non è stato aggiunto ai pulsanti di opzione.

Modifica dell'aspetto del controllo

I controlli **RadioButton** nel form consentono all'utente di modificare i colori di primo piano e di sfondo del controllo composito di WPF, nonché diverse proprietà del tipo di carattere. Il colore di sfondo viene esposto dall'oggetto **ElementHost**. Le proprietà restanti sono esposte come proprietà personalizzate del controllo.

Fare doppio clic su ogni controllo **RadioButton** sul form per creare **CheckedChanged** gestori eventi. Sostituire i gestori eventi **CheckedChanged** con il codice seguente.

```
private void radioBackgroundOriginal_CheckedChanged(object sender, EventArgs e)
{
    wpfAddressCtrl1.MyControl_Background = initBackBrush;
}

private void radioBackgroundLightGreen_CheckedChanged(object sender, EventArgs e)
{
    wpfAddressCtrl1.MyControl_Background = new SolidColorBrush(Colors.LightGreen);
}

private void radioBackgroundLightSalmon_CheckedChanged(object sender, EventArgs e)
{
    wpfAddressCtrl1.MyControl_Background = new SolidColorBrush(Colors.LightSalmon);
}
```

```

private void radioForegroundOriginal_CheckedChanged(object sender, EventArgs e)
{
    wpfAddressCtrl.MyControl_Foreground = initForeBrush;
}

private void radioForegroundRed_CheckedChanged(object sender, EventArgs e)
{
    wpfAddressCtrl.MyControl_Foreground = new System.Windows.Media.SolidColorBrush(Colors.Red);
}

private void radioForegroundYellow_CheckedChanged(object sender, EventArgs e)
{
    wpfAddressCtrl.MyControl_Foreground = new System.Windows.Media.SolidColorBrush(Colors.Yellow);
}

private void radioFamilyOriginal_CheckedChanged(object sender, EventArgs e)
{
    wpfAddressCtrl.MyControl_FontFamily = initFontFamily;
}

private void radioFamilyTimes_CheckedChanged(object sender, EventArgs e)
{
    wpfAddressCtrl.MyControl_FontFamily = new System.Windows.Media.FontFamily("Times New Roman");
}

private void radioFamilyWingDings_CheckedChanged(object sender, EventArgs e)
{
    wpfAddressCtrl.MyControl_FontFamily = new System.Windows.Media.FontFamily("WingDings");
}

private void radioSizeOriginal_CheckedChanged(object sender, EventArgs e)
{
    wpfAddressCtrl.MyControl_FontSize = initFontSize;
}

private void radioSizeTen_CheckedChanged(object sender, EventArgs e)
{
    wpfAddressCtrl.MyControl_FontSize = 10;
}

private void radioSizeTwelve_CheckedChanged(object sender, EventArgs e)
{
    wpfAddressCtrl.MyControl_FontSize = 12;
}

private void radioStyleOriginal_CheckedChanged(object sender, EventArgs e)
{
    wpfAddressCtrl.MyControl_FontStyle = initFontStyle;
}

private void radioStyleItalic_CheckedChanged(object sender, EventArgs e)
{
    wpfAddressCtrl.MyControl_FontStyle = System.Windows.FontStyles.Italic;
}

private void radioWeightOriginal_CheckedChanged(object sender, EventArgs e)
{
    wpfAddressCtrl.MyControl_FontWeight = initFontWeight;
}

private void radioWeightBold_CheckedChanged(object sender, EventArgs e)
{
    wpfAddressCtrl.MyControl_FontWeight = FontWeights.Bold;
}

```

Compilare ed eseguire l'applicazione. Fare clic sui vari pulsanti di opzione per vedere l'effetto sul controllo

composito WPF.

Vedere anche

- [ElementHost](#)
- [WindowsFormsHost](#)
- [Progettare XAML in Visual Studio](#)
- [Procedura dettagliata: Hosting di controlli Windows Form compositi in WPF](#)
- [Procedura dettagliata: hosting di controlli compositi 3D di WPF in Windows Form](#)

Procedura dettagliata: mapping delle proprietà tramite il controllo ElementHost

31/01/2020 • 9 minutes to read • [Edit Online](#)

In questa procedura dettagliata viene illustrato come utilizzare la proprietà [PropertyMap](#) per eseguire il mapping delle proprietà Windows Forms alle proprietà corrispondenti in un elemento WPF ospitato.

Le attività illustrate nella procedura dettagliata sono le seguenti:

- Creazione del progetto.
- Definizione di un nuovo mapping delle proprietà.
- Definizione di un mapping delle proprietà predefinito.
- Estensione di un mapping delle proprietà predefinito.

Per un listato di codice completo delle attività illustrate in questa procedura dettagliata, vedere [mapping delle proprietà tramite l'esempio di controllo ElementHost](#).

Al termine, sarà possibile eseguire il mapping delle proprietà di Windows Forms alle proprietà WPF corrispondenti su un elemento ospitato.

Prerequisiti

Per completare la procedura dettagliata, è necessario disporre dei componenti seguenti:

- Visual Studio 2017

Creazione del progetto

Per creare il progetto

1. Creare un progetto di **App Windows Forms** denominato `PropertyMappingWithElementHost`.
2. In **Esplora soluzioni** aggiungere riferimenti agli assembly WPF seguenti.
 - PresentationCore
 - PresentationFramework
 - WindowsBase
 - WindowsFormsIntegration
3. Copiare il codice seguente nella parte superiore del file di codice `Form1`.

```
using System.Windows;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Forms.Integration;
```

```
Imports System.Windows
Imports System.Windows.Media
Imports System.Windows.Media.Imaging
Imports System.Windows.Forms.Integration
```

4. Aprire `Form1` in Progettazione Windows Form. Fare doppio clic sul form per aggiungere un gestore eventi per l'evento `Load`.
5. Tornare alla Progettazione Windows Form e aggiungere un gestore eventi per l'evento `Resize` del modulo. Per altre informazioni, vedere [procedura: creare gestori eventi tramite la finestra di progettazione](#).
6. Dichiarare un campo `ElementHost` nella classe `Form1`.

```
ElementHost elemHost = null;
```

```
Private elemHost As ElementHost = Nothing
```

Definizione di nuovi mapping di proprietà

Il controllo `ElementHost` fornisce diversi mapping di proprietà predefiniti. Per aggiungere un nuovo mapping della proprietà, chiamare il metodo `Add` sul `PropertyMap` del controllo `ElementHost`.

Per definire nuovi mapping di proprietà

1. Copiare il codice seguente nella definizione della classe `Form1`.

```
// The AddMarginMapping method adds a new property mapping
// for the Margin property.
private void AddMarginMapping()
{
    elemHost.PropertyMap.Add(
        "Margin",
        new PropertyTranslator(OnMarginChange));
}

// The OnMarginChange method implements the mapping
// from the Windows Forms Margin property to the
// Windows Presentation Foundation Margin property.
//
// The provided Padding value is used to construct
// a Thickness value for the hosted element's Margin
// property.
private void OnMarginChange(object h, String propertyName, object value)
{
    ElementHost host = h as ElementHost;
    Padding p = (Padding)value;
    System.Windows.Controls.Button wpfButton =
        host.Child as System.Windows.Controls.Button;

    Thickness t = new Thickness(p.Left, p.Top, p.Right, p.Bottom);

    wpfButton.Margin = t;
}
```

```

' The AddMarginMapping method adds a new property mapping
' for the Margin property.
Private Sub AddMarginMapping()

    elemHost.PropertyMap.Add( _
        "Margin", _
        New PropertyTranslator(AddressOf OnMarginChange))

End Sub

' The OnMarginChange method implements the mapping
' from the Windows Forms Margin property to the
' Windows Presentation Foundation Margin property.
'
' The provided Padding value is used to construct
' a Thickness value for the hosted element's Margin
' property.
Private Sub OnMarginChange( _
    ByVal h As Object, _
    ByVal propertyName As String, _
    ByVal value As Object)

    Dim host As ElementHost = h
    Dim p As Padding = CType(value, Padding)
    Dim wpfButton As System.Windows.Controls.Button = host.Child

    Dim t As New Thickness(p.Left, p.Top, p.Right, p.Bottom)

    wpfButton.Margin = t

End Sub

```

Il metodo `AddMarginMapping` aggiunge un nuovo mapping per la proprietà `Margin`.

Il metodo `OnMarginChange` converte la proprietà `Margin` nella proprietà `Margin` WPF.

2. Copiare il codice seguente nella definizione della classe `Form1`.

```
// The AddRegionMapping method assigns a custom
// mapping for the Region property.
private void AddRegionMapping()
{
    elemHost.PropertyMap.Add(
        "Region",
        new PropertyTranslator(OnRegionChange));
}

// The OnRegionChange method assigns an EllipseGeometry to
// the hosted element's Clip property.
private void OnRegionChange(
    object h,
    String propertyName,
    object value)
{
    ElementHost host = h as ElementHost;
    System.Windows.Controls.Button wpfButton =
        host.Child as System.Windows.Controls.Button;

    wpfButton.Clip = new EllipseGeometry(new Rect(
        0,
        0,
        wpfButton.ActualWidth,
        wpfButton.ActualHeight));
}

// The Form1_Resize method handles the form's Resize event.
// It calls the OnRegionChange method explicitly to
// assign a new clipping geometry to the hosted element.
private void Form1_Resize(object sender, EventArgs e)
{
    this.OnRegionChange(elemHost, "Region", null);
}
```

```

' The AddRegionMapping method assigns a custom
' mapping for the Region property.
Private Sub AddRegionMapping()

    elemHost.PropertyMap.Add( _
        "Region", _
        New PropertyTranslator(AddressOf OnRegionChange))

End Sub

' The OnRegionChange method assigns an EllipseGeometry to
' the hosted element's Clip property.
Private Sub OnRegionChange( _
    ByVal h As Object, _
    ByVal propertyName As String, _
    ByVal value As Object)

    Dim host As ElementHost = h

    Dim wpfButton As System.Windows.Controls.Button = host.Child

    wpfButton.Clip = New EllipseGeometry(New Rect( _
        0, _
        0, _
        wpfButton.ActualWidth, _
        wpfButton.ActualHeight))

End Sub

' The Form1_Resize method handles the form's Resize event.
' It calls the OnRegionChange method explicitly to
' assign a new clipping geometry to the hosted element.
Private Sub Form1_Resize( _
    ByVal sender As Object, _
    ByVal e As EventArgs) Handles MyBase.Resize

    If elemHost IsNot Nothing Then
        Me.OnRegionChange(elemHost, "Region", Nothing)
    End If

End Sub

```

Il metodo `AddRegionMapping` aggiunge un nuovo mapping per la proprietà `Region`.

Il metodo `OnRegionChange` converte la proprietà `Region` nella proprietà `Clip` WPF.

Il metodo `Form1_Resize` gestisce l'evento `Resize` del form e ridimensiona l'area di ridimensionamento per adattarla all'elemento ospitato.

Rimozione di un mapping delle proprietà predefinito

Rimuovere un mapping di proprietà predefinito chiamando il metodo `Remove` sul `PropertyMap` del controllo `ElementHost`.

Per rimuovere un mapping delle proprietà predefinito

- Copiare il codice seguente nella definizione della classe `Form1`.

```
// The RemoveCursorMapping method deletes the default
// mapping for the Cursor property.
private void RemoveCursorMapping()
{
    elemHost.PropertyMap.Remove("Cursor");
}
```

```
' The RemoveCursorMapping method deletes the default
' mapping for the Cursor property.
Private Sub RemoveCursorMapping()
    elemHost.PropertyMap.Remove("Cursor")
End Sub
```

Il metodo `RemoveCursorMapping` Elimina il mapping predefinito per la proprietà [Cursor](#).

Estensione di un mapping delle proprietà predefinito

È possibile usare un mapping delle proprietà predefinito ed estenderlo con un mapping personalizzato.

Per estendere un mapping delle proprietà predefinito

- Copiare il codice seguente nella definizione della classe `Form1`.

```
// The ExtendBackColorMapping method adds a property
// translator if a mapping already exists.
private void ExtendBackColorMapping()
{
    if (elemHost.PropertyMap["BackColor"] != null)
    {
        elemHost.PropertyMap["BackColor"] +=
            new PropertyTranslator(OnBackColorChange);
    }
}

// The OnBackColorChange method assigns a specific image
// to the hosted element's Background property.
private void OnBackColorChange(object h, String propertyName, object value)
{
    ElementHost host = h as ElementHost;
    System.Windows.Controls.Button wpfButton =
        host.Child as System.Windows.Controls.Button;

    ImageBrush b = new ImageBrush(new BitmapImage(
        new Uri(@"file:///C:\WINDOWS\Santa Fe Stucco.bmp")));
    wpfButton.Background = b;
}
```

```

' The ExtendBackColorMapping method adds a property
' translator if a mapping already exists.
Private Sub ExtendBackColorMapping()

    If elemHost.PropertyMap("BackColor") IsNot Nothing Then

        elemHost.PropertyMap("BackColor") = PropertyTranslator.Combine( _
            elemHost.PropertyMap("BackColor"), _
            PropertyTranslator.CreateDelegate( _
                GetType(PropertyTranslator), _
                Me, _
                "OnBackColorChange"))
    End If

End Sub

' The OnBackColorChange method assigns a specific image
' to the hosted element's Background property.
Private Sub OnBackColorChange( _
    ByVal h As Object, _
    ByVal propertyName As String, _
    ByVal value As Object)

    Dim host As ElementHost = h
    Dim wpfButton As System.Windows.Controls.Button = host.Child
    Dim b As New ImageBrush(New BitmapImage( _
        New Uri("file:///C:\WINDOWS\Santa Fe Stucco.bmp")))
    wpfButton.Background = b
End Sub

```

Il metodo `ExtendBackColorMapping` aggiunge un convertitore di proprietà personalizzato al mapping della proprietà `BackColor` esistente.

Il metodo `OnBackColorChange` assegna un'immagine specifica alla proprietà `Background` del controllo ospitato. Il metodo `OnBackColorChange` viene chiamato dopo l'applicazione del mapping di proprietà predefinito.

Inizializzare i mapping delle proprietà

1. Copiare il codice seguente nella definizione della classe `Form1`.

```
private void Form1_Load(object sender, EventArgs e)
{
    // Create the ElementHost control.
    elemHost = new ElementHost();
    elemHost.Dock = DockStyle.Fill;
    this.Controls.Add(elemHost);

    // Create a Windows Presentation Foundation Button element
    // and assign it as the ElementHost control's child.
    System.Windows.Controls.Button wpfButton = new System.Windows.Controls.Button();
    wpfButton.Content = "Windows Presentation Foundation Button";
    elemHost.Child = wpfButton;

    // Map the Margin property.
    this.AddMarginMapping();

    // Remove the mapping for the Cursor property.
    this.RemoveCursorMapping();

    // Add a mapping for the Region property.
    this.AddRegionMapping();

    // Add another mapping for the BackColor property.
    this.ExtendBackColorMapping();

    // Cause the OnMarginChange delegate to be called.
    elemHost.Margin = new Padding(23, 23, 23, 23);

    // Cause the OnRegionChange delegate to be called.
    elemHost.Region = new Region();

    // Cause the OnBackColorChange delegate to be called.
    elemHost.BackColor = System.Drawing.Color.AliceBlue;
}
```

```

Private Sub Form1_Load( _
    ByVal sender As Object, _
    ByVal e As EventArgs) Handles MyBase.Load

    ' Create the ElementHost control.
    elemHost = New ElementHost()
    elemHost.Dock = DockStyle.Fill
    Me.Controls.Add(elemHost)

    ' Create a Windows Presentation Foundation Button element
    ' and assign it as the ElementHost control's child.
    Dim wpfButton As New System.Windows.Controls.Button()
    wpfButton.Content = "Windows Presentation Foundation Button"
    elemHost.Child = wpfButton

    ' Map the Margin property.
    Me.AddMarginMapping()

    ' Remove the mapping for the Cursor property.
    Me.RemoveCursorMapping()

    ' Add a mapping for the Region property.
    Me.AddRegionMapping()

    ' Add another mapping for the BackColor property.
    Me.ExtendBackColorMapping()

    ' Cause the OnMarginChange delegate to be called.
    elemHost.Margin = New Padding(23, 23, 23, 23)

    ' Cause the OnRegionChange delegate to be called.
    elemHost.Region = New [Region]()

    ' Cause the OnBackColorChange delegate to be called.
    elemHost.BackColor = System.Drawing.Color.AliceBlue

End Sub

```

Il metodo `Form1_Load` gestisce l'evento `Load` ed esegue l'inizializzazione seguente.

- Crea un elemento WPF `Button`.
- Chiama i metodi definiti in precedenza nella procedura dettagliata per impostare i mapping delle proprietà.
- Assegna i valori iniziali alle proprietà mappate.

2. Premere F5 per compilare ed eseguire l'applicazione.

Vedere anche

- [ElementHost.PropertyMap](#)
- [WindowsFormsHost.PropertyMap](#)
- [WindowsFormsHost](#)
- [Mapping di proprietà di Windows Form e WPF](#)
- [Progettare XAML in Visual Studio](#)
- [Procedura dettaglia: hosting di un controllo WPF composito in Windows Form](#)

Procedura dettagliata: mapping di proprietà tramite l'elemento WindowsFormsHost

31/01/2020 • 12 minutes to read • [Edit Online](#)

In questa procedura dettagliata viene illustrato come utilizzare la proprietà [PropertyMap](#) per eseguire il mapping delle proprietà WPF alle proprietà corrispondenti in un controllo Windows Forms ospitato.

Le attività illustrate nella procedura dettagliata sono le seguenti:

- Creazione del progetto.
- Definizione del layout dell'applicazione.
- Definizione di un nuovo mapping delle proprietà.
- Definizione di un mapping delle proprietà predefinito.
- Sostituzione di un mapping delle proprietà predefinito.
- Estensione di un mapping delle proprietà predefinito.

Per un listato di codice completo delle attività illustrate in questa procedura dettagliata, vedere [mapping delle proprietà tramite l'esempio WindowsFormsHost](#).

Al termine, sarà possibile eseguire il mapping delle proprietà WPF alle proprietà corrispondenti in un controllo Windows Forms ospitato.

Prerequisiti

Per completare la procedura dettagliata, è necessario disporre dei componenti seguenti:

- Visual Studio 2017

Creare e configurare il progetto

1. Creare un progetto di **applicazione WPF** denominato `PropertyMappingWithWfhSample`.
2. In **Esplora soluzioni** aggiungere un riferimento all'assembly WindowsFormsIntegration, denominato WindowsFormsIntegration.dll.
3. In **Esplora soluzioni** aggiungere riferimenti agli assembly System.Drawing e System.Windows.Forms.

Definizione del layout dell'applicazione

L'applicazione basata su WPF usa l'elemento [WindowsFormsHost](#) per ospitare un controllo di Windows Forms.

Per definire il layout dell'applicazione

1. Aprire Window1.xaml in WPF Designer.
2. Sostituire il codice esistente con quello seguente.

```
<Window x:Class="PropertyMappingWithWfh.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="PropertyMappingWithWfh" Height="300" Width="300"
    Loaded="WindowLoaded">
    <DockPanel Name="panel1" LastChildFill="True">
        <WindowsFormsHost Name="wfHost" DockPanel.Dock="Left" SizeChanged="Window1_SizeChanged"
            FontSize="20" />
    </DockPanel>
</Window>
```

3. Aprire Window1.xaml.cs nell'editor di codice.
4. Nella parte superiore del file importare gli spazi dei nomi seguenti.

```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;
using System.Windows.Forms.Integration;
```

```
Imports System.Drawing
Imports System.Drawing.Drawing2D
Imports System.Windows.Forms
Imports System.Windows.Forms.Integration
```

Definizione di un nuovo mapping delle proprietà

L'elemento [WindowsFormsHost](#) fornisce diversi mapping di proprietà predefiniti. Per aggiungere un nuovo mapping della proprietà, chiamare il metodo [Add](#) sul [PropertyMap](#) dell'elemento del [WindowsFormsHost](#).

Per definire un nuovo mapping delle proprietà

- Copiare il codice seguente nella definizione della classe `Window1`.

```
// The AddClipMapping method adds a custom
// mapping for the Clip property.
private void AddClipMapping()
{
    wfHost.PropertyMap.Add(
        "Clip",
        new PropertyTranslator(OnClipChange));
}

// The OnClipChange method assigns an elliptical clipping
// region to the hosted control's Region property.
private void OnClipChange(object h, String propertyName, object value)
{
    WindowsFormsHost host = h as WindowsFormsHost;
    System.Windows.Forms.CheckBox cb = host.Child as System.Windows.Forms.CheckBox;

    if (cb != null)
    {
        cb.Region = this.CreateClipRegion();
    }
}

// The Window1_SizeChanged method handles the window's
// SizeChanged event. It calls the OnClipChange method explicitly
// to assign a new clipping region to the hosted control.
private void Window1_SizeChanged(object sender, SizeChangedEventArgs e)
{
    this.OnClipChange(wfHost, "Clip", null);
}

// The CreateClipRegion method creates a Region from an
// elliptical GraphicsPath.
private Region CreateClipRegion()
{
    GraphicsPath path = new GraphicsPath();

    path.StartFigure();

    path.AddEllipse(new System.Drawing.Rectangle(
        0,
        0,
        (int)wfHost.ActualWidth,
        (int)wfHost.ActualHeight ) );

    path.CloseFigure();

    return( new Region(path) );
}
```

```

' The AddClipMapping method adds a custom mapping
' for the Clip property.
Private Sub AddClipMapping()

    wfHost.PropertyMap.Add( _
        "Clip", _
        New PropertyTranslator(AddressOf OnClipChange))

End Sub

' The OnClipChange method assigns an elliptical clipping
' region to the hosted control's Region property.
Private Sub OnClipChange( _
    ByVal h As Object, _
    ByVal propertyName As String, _
    ByVal value As Object)

    Dim host As WindowsFormsHost = h

    Dim cb As System.Windows.Forms.CheckBox = host.Child

    If cb IsNot Nothing Then
        cb.Region = Me.CreateClipRegion()
    End If

End Sub

' The Window1_SizeChanged method handles the window's
' SizeChanged event. It calls the OnClipChange method explicitly
' to assign a new clipping region to the hosted control.
Private Sub Window1_SizeChanged( _
    ByVal sender As Object, _
    ByVal e As SizeChangedEventArgs)

    Me.OnClipChange(wfHost, "Clip", Nothing)

End Sub

' The CreateClipRegion method creates a Region from an
' elliptical GraphicsPath.
Private Function CreateClipRegion() As [Region]
    Dim path As New GraphicsPath()

    path.StartFigure()

    path.AddEllipse(New System.Drawing.Rectangle( _
        0, _
        0, _
        wfHost.ActualWidth, _
        wfHost.ActualHeight))

    path.CloseFigure()

    Return New [Region](path)
End Function

```

Il metodo `AddClipMapping` aggiunge un nuovo mapping per la proprietà `Clip`.

Il metodo `OnClipChange` converte la proprietà `Clip` nella proprietà `Region` Windows Forms.

Il metodo `Window1_SizeChanged` gestisce l'evento `SizeChanged` della finestra e ridimensiona l'area di ridimensionamento per adattarla alla finestra dell'applicazione.

Rimozione di un mapping delle proprietà predefinito

Rimuovere un mapping di proprietà predefinito chiamando il metodo [Remove](#) sul [PropertyMap](#) dell'elemento del [WindowsFormsHost](#).

Per rimuovere un mapping delle proprietà predefinite

- Copiare il codice seguente nella definizione della classe [Window1](#).

```
// The RemoveCursorMapping method deletes the default
// mapping for the Cursor property.
private void RemoveCursorMapping()
{
    wfHost.PropertyMap.Remove("Cursor");
}
```

```
' The RemoveCursorMapping method deletes the default
' mapping for the Cursor property.
Private Sub RemoveCursorMapping()
    wfHost.PropertyMap.Remove("Cursor")
End Sub
```

Il metodo [RemoveCursorMapping](#) Elimina il mapping predefinito per la proprietà [Cursor](#).

Sostituzione di un mapping delle proprietà predefinite

Sostituire un mapping di proprietà predefinito rimuovendo il mapping predefinito e chiamando il metodo [Add](#) sul [PropertyMap](#) dell'elemento del [WindowsFormsHost](#).

Per sostituire un mapping delle proprietà predefinite

- Copiare il codice seguente nella definizione della classe [Window1](#).

```
// The ReplaceFlowDirectionMapping method replaces the
// default mapping for the FlowDirection property.
private void ReplaceFlowDirectionMapping()
{
    wfHost.PropertyMap.Remove("FlowDirection");

    wfHost.PropertyMap.Add(
        "FlowDirection",
        new PropertyTranslator(OnFlowDirectionChange));
}

// The OnFlowDirectionChange method translates a
// Windows Presentation Foundation FlowDirection value
// to a Windows Forms RightToLeft value and assigns
// the result to the hosted control's RightToLeft property.
private void OnFlowDirectionChange(object h, String propertyName, object value)
{
    WindowsFormsHost host = h as WindowsFormsHost;
    System.Windows.FlowDirection fd = (System.Windows.FlowDirection)value;
    System.Windows.Forms.CheckBox cb = host.Child as System.Windows.Forms.CheckBox;

    cb.RightToLeft = (fd == System.Windows.FlowDirection.RightToLeft) ?
        RightToLeft.Yes : RightToLeft.No;
}

// The cb_CheckedChanged method handles the hosted control's
// CheckedChanged event. If the Checked property is true,
// the flow direction is set to RightToLeft, otherwise it is
// set to LeftToRight.
private void cb_CheckedChanged(object sender, EventArgs e)
{
    System.Windows.Forms.CheckBox cb = sender as System.Windows.Forms.CheckBox;

    wfHost.FlowDirection = ( cb.CheckState == CheckState.Checked ) ?
        System.Windows.FlowDirection.RightToLeft :
        System.Windows.FlowDirection.LeftToRight;
}
```

```

' The ReplaceFlowDirectionMapping method replaces the
' default mapping for the FlowDirection property.
Private Sub ReplaceFlowDirectionMapping()

    wfHost.PropertyMap.Remove("FlowDirection")

    wfHost.PropertyMap.Add( _
        "FlowDirection", _
        New PropertyTranslator(AddressOf OnFlowDirectionChange))
End Sub

' The OnFlowDirectionChange method translates a
' Windows Presentation Foundation FlowDirection value
' to a Windows Forms RightToLeft value and assigns
' the result to the hosted control's RightToLeft property.
Private Sub OnFlowDirectionChange( _
    ByVal h As Object, _
    ByVal propertyName As String, _
    ByVal value As Object)

    Dim host As WindowsFormsHost = h

    Dim fd As System.Windows.FlowDirection = _
        CType(value, System.Windows.FlowDirection)

    Dim cb As System.Windows.Forms.CheckBox = host.Child

    cb.RightToLeft = IIf(fd = System.Windows.FlowDirection.RightToLeft, _
        RightToLeft.Yes, _
        RightToLeft.No)
End Sub

' The cb_CheckedChanged method handles the hosted control's
' CheckedChanged event. If the Checked property is true,
' the flow direction is set to RightToLeft, otherwise it is
' set to LeftToRight.
Private Sub cb_CheckedChanged( _
    ByVal sender As Object, _
    ByVal e As EventArgs)

    Dim cb As System.Windows.Forms.CheckBox = sender

    wfHost.FlowDirection = IIf(cb.CheckState = CheckState.Checked, _
        System.Windows.FlowDirection.RightToLeft, _
        System.Windows.FlowDirection.LeftToRight)
End Sub

```

Il metodo `ReplaceFlowDirectionMapping` sostituisce il mapping predefinito per la proprietà `FlowDirection`.

Il metodo `OnFlowDirectionChange` converte la proprietà `FlowDirection` nella proprietà `RightToLeft` Windows Forms.

Il metodo `cb_CheckedChanged` gestisce l'evento `CheckedChanged` sul controllo `CheckBox`. Assegna la proprietà `FlowDirection` in base al valore della proprietà `CheckState`.

Estensione di un mapping delle proprietà predefinito

È possibile usare un mapping delle proprietà predefinito ed estenderlo con un mapping personalizzato.

Per estendere un mapping delle proprietà predefinito

- Copiare il codice seguente nella definizione della classe `Window1`.

```
// The ExtendBackgroundMapping method adds a property
// translator if a mapping already exists.
private void ExtendBackgroundMapping()
{
    if (wfHost.PropertyMap["Background"] != null)
    {
        wfHost.PropertyMap["Background"] += new PropertyTranslator(OnBackgroundChange);
    }
}

// The OnBackgroundChange method assigns a specific image
// to the hosted control's BackgroundImage property.
private void OnBackgroundChange(object h, String propertyName, object value)
{
    WindowsFormsHost host = h as WindowsFormsHost;
    System.Windows.Forms.CheckBox cb = host.Child as System.Windows.Forms.CheckBox;
    ImageBrush b = value as ImageBrush;

    if (b != null)
    {
        cb.BackgroundImage = new System.Drawing.Bitmap(@"C:\WINDOWS\Santa Fe Stucco.bmp");
    }
}
```

```
' The ExtendBackgroundMapping method adds a property
' translator if a mapping already exists.
Private Sub ExtendBackgroundMapping()
    If wfHost.PropertyMap("Background") IsNot Nothing Then

        wfHost.PropertyMap("Background") = PropertyTranslator.Combine( _
            wfHost.PropertyMap("Background"), _
            PropertyTranslator.CreateDelegate( _
                GetType(PropertyTranslator), _
                Me, _
                "OnBackgroundChange"))
    End If

End Sub

' The OnBackgroundChange method assigns a specific image
' to the hosted control's BackgroundImage property.
Private Sub OnBackgroundChange(ByVal h As Object, ByVal propertyName As String, ByVal value As Object)
    Dim host As WindowsFormsHost = h
    Dim cb As System.Windows.Forms.CheckBox = host.Child
    Dim b As ImageBrush = value

    If Not (b Is Nothing) Then
        cb.BackgroundImage = New System.Drawing.Bitmap("C:\WINDOWS\Santa Fe Stucco.bmp")
    End If

End Sub
```

Il metodo `ExtendBackgroundMapping` aggiunge un convertitore di proprietà personalizzato al mapping della proprietà `Background` esistente.

Il metodo `OnBackgroundChange` assegna un'immagine specifica alla proprietà `BackgroundImage` del controllo ospitato. Il metodo `OnBackgroundChange` viene chiamato dopo l'applicazione del mapping di proprietà predefinito.

Inizializzazione dei mapping delle proprietà

Configurare i mapping delle proprietà chiamando i metodi descritti in precedenza nel gestore eventi [Loaded](#).

Per inizializzare i mapping delle proprietà

1. Copiare il codice seguente nella definizione della classe `Window1`.

```
// The WindowLoaded method handles the Loaded event.  
// It enables Windows Forms visual styles, creates  
// a Windows Forms checkbox control, and assigns the  
// control as the child of the WindowsFormsHost element.  
// This method also modifies property mappings on the  
// WindowsFormsHost element.  
private void WindowLoaded(object sender, RoutedEventArgs e)  
{  
    System.Windows.Forms.Application.EnableVisualStyles();  
  
    // Create a Windows Forms checkbox control and assign  
    // it as the WindowsFormsHost element's child.  
    System.Windows.Forms.CheckBox cb = new System.Windows.Forms.CheckBox();  
    cb.Text = "Windows Forms checkbox";  
    cb.Dock = DockStyle.Fill;  
    cb.TextAlign = ContentAlignment.MiddleCenter;  
    cb.CheckedChanged += new EventHandler(cb_CheckedChanged);  
    wfHost.Child = cb;  
  
    // Replace the default mapping for the FlowDirection property.  
    this.ReplaceFlowDirectionMapping();  
  
    // Remove the mapping for the Cursor property.  
    this.RemoveCursorMapping();  
  
    // Add the mapping for the Clip property.  
    this.AddClipMapping();  
  
    // Add another mapping for the Background property.  
    this.ExtendBackgroundMapping();  
  
    // Cause the OnFlowDirectionChange delegate to be called.  
    wfHost.FlowDirection = System.Windows.FlowDirection.LeftToRight;  
  
    // Cause the OnClipChange delegate to be called.  
    wfHost.Clip = new RectangleGeometry();  
  
    // Cause the OnBackgroundChange delegate to be called.  
    wfHost.Background = new ImageBrush();  
}
```

```

' The WindowLoaded method handles the Loaded event.
' It enables Windows Forms visual styles, creates
' a Windows Forms checkbox control, and assigns the
' control as the child of the WindowsFormsHost element.
' This method also modifies property mappings on the
' WindowsFormsHost element.
Private Sub WindowLoaded( _
 ByVal sender As Object, _
 ByVal e As RoutedEventArgs)

    System.Windows.Forms.Application.EnableVisualStyles()

    ' Create a Windows Forms checkbox control and assign
    ' it as the WindowsFormsHost element's child.
    Dim cb As New System.Windows.Forms.CheckBox()
    cb.Text = "Windows Forms checkbox"
    cb.Dock = DockStyle.Fill
    cb.TextAlign = ContentAlignment.MiddleCenter
    AddHandler cb.CheckedChanged, AddressOf cb_CheckedChanged
    wfHost.Child = cb

    ' Replace the default mapping for the FlowDirection property.
    Me.ReplaceFlowDirectionMapping()

    ' Remove the mapping for the Cursor property.
    Me.RemoveCursorMapping()

    ' Add the mapping for the Clip property.
    Me.AddClipMapping()

    ' Add another mapping for the Background property.
    Me.ExtendBackgroundMapping()

    ' Cause the OnFlowDirectionChange delegate to be called.
    wfHost.FlowDirection = System.Windows.FlowDirection.LeftToRight

    ' Cause the OnClipChange delegate to be called.
    wfHost.Clip = New RectangleGeometry()

    ' Cause the OnBackgroundChange delegate to be called.
    wfHost.Background = New ImageBrush()

End Sub

```

Il metodo `WindowLoaded` gestisce l'evento `Loaded` ed esegue l'inizializzazione seguente.

- Crea una Windows Forms controllo `CheckBox`.
- Chiama i metodi definiti in precedenza nella procedura dettagliata per impostare i mapping delle proprietà.
- Assegna i valori iniziali alle proprietà mappate.

2. Premere **F5** per compilare ed eseguire l'applicazione. Fare clic sulla casella di controllo per visualizzare l'effetto del mapping del `FlowDirection`. Quando si seleziona la casella di controllo, il layout inverte l'orientamento da sinistra a destra.

Vedere anche

- [WindowsFormsHost.PropertyMap](#)
- [ElementHost.PropertyMap](#)
- [WindowsFormsHost](#)
- [Mapping di proprietà di Windows Form e WPF](#)

- [Progettare XAML in Visual Studio](#)
- [Procedura dettagliata: hosting di controlli Windows Form in WPF](#)

Procedura dettagliata: localizzazione di un'applicazione ibrida

31/01/2020 • 8 minutes to read • [Edit Online](#)

Questa procedura dettagliata illustra come localizzare WPF elementi in un'applicazione ibrida basata su Windows Forms.

Le attività illustrate nella procedura dettagliata sono le seguenti:

- Creazione del progetto host Windows Forms.
- Aggiunta di contenuto localizzabile.
- Abilitazione della localizzazione.
- Assegnazione di identificatori di risorsa.
- Utilizzo dello strumento LocBaml per produrre un assembly satellite.

Per un listato di codice completo delle attività illustrate in questa procedura dettagliata, vedere [esempio di localizzazione di un'applicazione ibrida](#).

Al termine, sarà disponibile un'applicazione ibrida localizzata.

Prerequisiti

Per completare la procedura dettagliata, è necessario disporre dei componenti seguenti:

- Visual Studio 2017

Creazione del progetto host Windows Form

Il primo passaggio consiste nel creare il progetto Windows Forms Application e aggiungere un elemento WPF con contenuto che verrà localizzato.

Per creare il progetto host

1. Creare un progetto di **applicazione WPF** denominato `LocalizingWpfInWf`. (**File > nuovo > progetto > Visual C# o Visual Basic > desktop classico > applicazione WPF**).
2. Aggiungere un WPFelemento **UserControl** chiamato `SimpleControl` al progetto.
3. Utilizzare il controllo **ElementHost** per inserire un elemento `SimpleControl` nel form. Per ulteriori informazioni, vedere [procedura dettagliata: hosting di un controllo composito 3D WPF in Windows Forms](#).

Aggiunta di contenuto localizzabile

Successivamente, si aggiungerà un controllo Label Windows Forms e si imposterà il contenuto dell'elemento WPF su una stringa localizzabile.

Per aggiungere contenuto localizzabile

1. In **Esplora soluzioni** fare doppio clic su **SimpleControl.xaml** per aprirlo in WPF Designer.
2. Impostare il contenuto del controllo **Button** usando il codice seguente.

```

<UserControl x:Class="LocalizingWpfInWf.SimpleControl"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    >

    <Canvas>
        <Button Content="Hello"/>
    </Canvas>
</UserControl>

```

3. In **Esplora soluzioni** fare doppio clic su **Form1** per aprirlo nella progettazione Windows Form.
 4. Aprire la **casella degli strumenti** e fare doppio clic su **etichetta** per aggiungere un controllo etichetta al modulo. Impostare il valore della proprietà **Text** su **"Hello"**.
 5. Premere **F5** per compilare ed eseguire l'applicazione.
- Sia l'elemento **SimpleControl** che il controllo Label visualizzano il testo **"Hello"**.

Abilitazione della localizzazione

Progettazione Windows Form fornisce le impostazioni per abilitare la localizzazione in un assembly satellite.

Per abilitare la localizzazione

1. In **Esplora soluzioni** fare doppio clic su **Form1.cs** per aprirlo nella progettazione Windows Form.
2. Nella finestra **Proprietà** impostare il valore della proprietà **localizzabile** del modulo su **true**.
3. Nella finestra **Proprietà** impostare il valore della proprietà **Language** su **spagnolo (Spagna)**.
4. In Progettazione Windows Form selezionare il controllo etichetta.
5. Nella finestra **Proprietà** impostare il valore della proprietà **Text** su **"Hola"**.

Un nuovo file di risorse denominato Form1.es-ES.resx verrà aggiunto al progetto.

6. In **Esplora soluzioni** fare clic con il pulsante destro del mouse su **Form1.cs** e scegliere **Visualizza codice** per aprirlo nell'editor di codice.
7. Copiare il codice seguente nel costruttore **Form1**, prima della chiamata a **InitializeComponent**.

```

public Form1()
{
    System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("es-ES");

    InitializeComponent();
}

```

8. In **Esplora soluzioni** fare clic con il pulsante destro del mouse su **LocalizingWpfInWf** e scegliere **Scarica progetto**.

Il nome del progetto è contrassegnato come non **disponibile**.

9. Fare clic con il pulsante destro del mouse su **LocalizingWpfInWf.csproj** scegliere **Modifica LocalizingWpfInWf.csproj**.

Il file di progetto verrà aperto nell'editor di codice.

10. Copiare la riga seguente nella prima **PropertyGroup** nel file di progetto.

```
<UICulture>en-US</UICulture>
```

11. Salvare e chiudere il file di progetto.
12. In **Esplo**ra soluzioni fare clic con il pulsante destro del mouse su **LocalizingWpfInWF** e scegliere **Ricarica progetto**.

Assegnazione di identificatori di risorsa

È possibile eseguire il mapping del contenuto localizzabile agli assembly di risorse usando identificatori di risorsa. Quando si specifica l'opzione `updateuid`, l'applicazione MsBuild.exe assegna automaticamente gli identificatori di risorsa.

Per assegnare identificatori di risorsa

1. Dal menu Start aprire il Prompt dei comandi per gli sviluppatori per Visual Studio.
2. Per assegnare identificatori di risorsa al contenuto localizzabile, usare il comando seguente.

```
msbuild -t:updateuid LocalizingWpfInWF.csproj
```

3. In **Esplo**ra soluzioni fare doppio clic su **SimpleControl.xaml** per aprirlo nell'editor di codice. Si noterà che il comando `msbuild` ha aggiunto l'attributo `Uid` a tutti gli elementi. In tal modo si semplifica la localizzazione mediante l'assegnazione di identificatori di risorsa.

```
<UserControl x:Uid="UserControl_1" x:Class="LocalizingWpfInWF.SimpleControl"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    >

    <Canvas x:Uid="Canvas_1">
        <Button x:Uid="Button_1" Content="Hello"/>
    </Canvas>
</UserControl>
```

4. Premere **F6** per compilare la soluzione.

Utilizzo dello strumento LocBaml per produrre un assembly satellite

Il contenuto localizzato viene archiviato in un *assembly satellitedi sole risorse*. Utilizzare lo strumento da riga di comando LocBaml.exe per generare un assembly localizzato per il contenuto del WPF.

Per produrre un assembly satellite

1. Copiare LocBaml.exe nella cartella obj\Debug del progetto. Per altre informazioni, vedere [localizzare un'applicazione](#).
2. Nella finestra del prompt dei comandi usare il comando seguente per estrarre stringhe di risorsa in un file temporaneo.

```
LocBaml /parse LocalizingWpfInWF.g.en-US.resources /out:temp.csv
```

3. Aprire il file Temp.csv con Visual Studio o un altro editor di testo. Sostituire la stringa "Hello" con la relativa traduzione spagnola, "Hola".
4. Salvare il file temp.csv.

5. Per generare il file di risorse localizzato, usare il comando seguente.

```
LocBaml /generate /trans:temp.csv LocalizingWpfInWf.g.en-US.resources /out:.. /cul:es-ES
```

Il file LocalizingWpfInWf.g.es-ES.resources viene creato nella cartella obj\Debug.

6. Per compilare l'assembly satellite localizzato, usare il comando seguente.

```
A1.exe /out:LocalizingWpfInWf.resources.dll /culture:es-ES /embed:LocalizingWpfInWf.Form1.es-ES.resources /embed:LocalizingWpfInWf.g.es-ES.resources
```

Il file LocalizingWpfInWf.resources.dll viene creato nella cartella obj\Debug.

7. Copiare il file LocalizingWpfInWf.resources.dll nella cartella bin\Debug\es-ES del progetto. Sostituire il file esistente.
8. Eseguire LocalizingWpfInWf.exe, situato nella cartella bin\Debug del progetto. Non ricompilare l'applicazione altrimenti l'assembly satellite verrà sovrascritto.

Nell'applicazione vengono visualizzate le stringhe localizzate invece delle stringhe in inglese.

Vedere anche

- [ElementHost](#)
- [WindowsFormsHost](#)
- [Localizzare un'applicazione](#)
- [Procedura dettagliata: localizzazione di Windows Forms](#)
- [Progettare XAML in Visual Studio](#)

Interoperatività di WPF e Win32

28/01/2020 • 21 minutes to read • [Edit Online](#)

In questo argomento viene fornita una panoramica dell'interoperabilità di WPF e del codice Win32. Windows Presentation Foundation (WPF) fornisce un ambiente completo per la creazione di applicazioni. Tuttavia, quando si ha un investimento sostanziale nel codice Win32, potrebbe essere più efficace riutilizzare parte del codice.

Nozioni di base sull'interoperatività di WPF e Win32

Esistono due tecniche di base per l'interoperatività tra WPF e il codice Win32.

- Ospitare WPF contenuto in una finestra Win32. Con questa tecnica, è possibile usare le funzionalità grafiche avanzate di WPF all'interno del Framework di una finestra e di un'applicazione Win32 standard.
- Ospita una finestra Win32 nel contenuto WPF. Con questa tecnica, è possibile usare un controllo Win32 personalizzato esistente nel contesto di altri WPF contenuto e passare i dati attraverso i limiti.

Questo argomento illustra concettualmente queste due tecniche. Per un'illustrazione più orientata al codice dell'hosting WPF in Win32, vedere [procedura dettagliata: hosting di contenuto WPF in Win32](#). Per un'illustrazione più orientata al codice dell'hosting di Win32 in WPF, vedere [procedura dettagliata: hosting di un controllo Win32 in WPF](#).

Progetti di interoperatività WPF

WPF API sono codice gestito, ma la maggior parte dei programmi Win32 esistenti sono scritti in C++ non gestiti. Non è possibile chiamare WPF API da un vero programma non gestito. Tuttavia, usando l'opzione `/c1r` con il compilatore visuale C++ Microsoft, è possibile creare un programma misto gestito e non gestito in cui è possibile combinare facilmente le chiamate API gestite e non gestite.

Una complicazione a livello di progetto è che non è possibile compilare Extensible Application Markup Language (XAML) file C++ in un progetto. Per ovviare a tale problema, ci sono diverse tecniche di divisione dei progetti.

- Creare una C# dll contenente tutte le pagine di XAML come assembly compilato e quindi fare in modo che C++ il file eseguibile includa tale DLL come riferimento.
- Creare un C# eseguibile per il contenuto del WPF e fare in riferimento C++ a una dll che contiene il contenuto Win32.
- Usare [Load](#) per caricare qualsiasi XAML in fase di esecuzione, anziché compilare il XAML.
- Non usare XAML e scrivere tutti i WPF nel codice, costruendo l'albero degli elementi da [Application](#).

Usare l'approccio che meglio soddisfa le esigenze.

NOTE

Se non è stato usato C++/CLI in precedenza, è possibile notare alcune parole chiave "nuove", ad esempio `gcnew` e `nullptr` negli esempi di codice di interoperabilità. Queste parole chiave sostituiscono la sintassi di doppio sottolineatura precedente (`__gc`) e forniscono una sintassi più naturale per il C++ codice gestito in. Per ulteriori informazioni sulle funzionalità C++ gestite di/CLI, vedere [estensioni dei componenti per le piattaforme di runtime](#).

Modo d'uso degli handle di finestra (HWND) in WPF

Per sfruttare al meglio le caratteristiche di "interoperatività HWND" di WPF, è necessario capire in che modo WPF usa gli handle di finestra (HWND). Per qualsiasi HWND, non è possibile combinare WPF rendering con il rendering DirectX o il rendering GDI/GDI+. Questo comporta diverse implicazioni. In primo luogo, per combinare questi modelli di rendering, è necessario creare una soluzione di interoperatività e usare segmenti designati di interoperatività per ogni modello di rendering che si sceglie di usare. Il comportamento di rendering crea inoltre una restrizione di "spazio aereo" per le operazioni che la soluzione di interoperatività può completare. Il concetto di "spazio aereo" è illustrato più dettagliatamente nell'argomento [Cenni preliminari sulle aree di tecnologia](#).

Tutti gli elementi WPF nella schermata sono supportati da un oggetto HWND. Quando si crea un WPF [Window](#), WPF crea un HWND di primo livello e utilizza un [HwndSource](#) per inserire il [Window](#) e il relativo contenuto WPF all'interno di HWND. Il resto del contenuto WPF nell'applicazione condivide tale oggetto HWND singolo. Un'eccezione è costituita da menu, caselle combinate a discesa e altri popup. Poiché questi elementi creano una propria finestra di primo livello, è possibile che un menu WPF superi potenzialmente il bordo dell'oggetto HWND della finestra che lo contiene. Quando si utilizza [HwndHost](#) per inserire un HWND all'interno di WPF, WPF informa Win32 come posizionare il nuovo HWND figlio rispetto alla WPF [Window](#) HWND.

Un concetto correlato a HWND è la trasparenza all'interno di ogni oggetto HWND e tra tali oggetti. Questo concetto è illustrato anche nell'argomento [Cenni preliminari sulle aree di tecnologia](#).

Hosting di contenuto WPF in una finestra Microsoft Win32

La chiave per ospitare un WPF in una finestra Win32 è la classe [HwndSource](#). Questa classe esegue il wrapping del contenuto WPF in una finestra Win32, in modo che il contenuto del WPF possa essere incorporato nel interfaccia utente come finestra figlio. L'approccio seguente combina Win32 e WPF in un'unica applicazione.

1. Implementare il contenuto WPF (elemento radice del contenuto) come classe gestita. In genere, la classe eredita da una delle classi che possono contenere più elementi figlio e/o usata come elemento radice, ad esempio [DockPanel](#) o [Page](#). Nei passaggi successivi questa classe è detta classe contenuto WPF e le istanze della classe sono dette oggetti contenuto WPF.
2. Implementare un'applicazione Windows con C++/cli. Se si inizia con un' C++ applicazione non gestita esistente, in genere è possibile abilitarla per chiamare il codice gestito modificando le impostazioni del progetto in modo da includere il flag del compilatore `/c1r` (l'ambito completo di ciò che potrebbe essere necessario per supportare la compilazione `/c1r` non è descritto in questo argomento).
3. Impostare il modello di threading su apartment a thread singolo (STA, Single Threaded Apartment). WPF usa questo modello di threading.
4. Gestire la notifica WM_CREATE nella procedura di finestra.
5. All'interno del gestore (o di una funzione chiamata dal gestore) eseguire queste operazioni:
 - a. Creare un nuovo oggetto [HwndSource](#) con HWND della finestra padre come parametro di `parent`
.
 - b. Creare un'istanza della classe contenuto WPF.
 - c. Assegnare un riferimento all'oggetto WPF contenuto alla proprietà [HwndSource](#) oggetto [RootVisual](#).
 - d. La proprietà [HwndSource](#) oggetto [Handle](#) contiene l'handle di finestra (HWND). Per ottenere un HWND utilizzabile nella parte non gestita dell'applicazione, eseguire il cast di `Handle.ToPointer()` a un HWND.
6. Implementare una classe gestita che include un campo statico contenente un riferimento all'oggetto contenuto WPF. Questa classe consente di ottenere un riferimento all'oggetto WPF contenuto dal codice

Win32, ma ancora più importante impedisce che la [HwndSource](#) venga inavvertitamente sottoposta a Garbage Collection.

7. Ricevere notifiche dall'oggetto contenuto WPF associando un gestore a uno o più eventi dell'oggetto contenuto WPF.
8. Comunicare con l'oggetto contenuto WPF usando il riferimento archiviato nel campo statico per impostare proprietà, chiamare metodi e così via.

NOTE

Tutte le attività di definizione della classe contenuto WPF, o alcune di esse, per il primo passaggio in XAML possono essere eseguite usando la classe parziale predefinita della classe contenuto, se si genera un assembly separato e quindi vi si fa riferimento. Sebbene in genere si includa un oggetto [Application](#) come parte della compilazione del XAML in un assembly, non si finisce di utilizzare tale [Application](#) come parte dell'interoperabilità, è sufficiente utilizzare una o più classi radice per i file XAML a cui fa riferimento l'applicazione e fare riferimento alle relative classi parziali. La parte restante della procedura è essenzialmente simile a quella appena descritta.

Ognuno di questi passaggi viene illustrato tramite il codice nell'argomento [Procedura dettagliata: hosting di contenuto WPF in Win32](#).

Hosting di una finestra Microsoft Win32 in WPF

La chiave per ospitare una finestra Win32 all'interno di altri WPF contenuto è la classe [HwndHost](#). Questa classe esegue il wrapping della finestra in un elemento WPF che può essere aggiunto a un albero degli elementi WPF. [HwndHost](#) supporta anche le API che consentono di eseguire attività quali l'elaborazione dei messaggi per la finestra ospitata. La procedura di base è la seguente:

1. Creare un albero degli elementi per un'applicazione WPF (tramite codice o markup). Trovare un punto appropriato e ammissibile nell'albero degli elementi in cui è possibile aggiungere l'implementazione del [HwndHost](#) come elemento figlio. Nei restanti passaggi questo elemento viene chiamato elemento di riserva.
2. Derivare da [HwndHost](#) per creare un oggetto che include il contenuto Win32.
3. In tale classe host, eseguire l'override del metodo [HwndHost BuildWindowCore](#). Restituire l'oggetto HWND della finestra ospitata. Può essere necessario eseguire il wrapping dei controlli effettivi come finestra figlio della finestra restituita. Il wrapping dei controlli in una finestra host offre un modo semplice per consentire al contenuto WPF di ricevere notifiche dai controlli. Questa tecnica consente di correggere alcuni problemi Win32 relativi alla gestione dei messaggi in corrispondenza del limite del controllo ospitato.
4. Eseguire l'override dei metodi di [HwndHost DestroyWindowCore](#) e [WndProc](#). Lo scopo è quello di eseguire la pulizia e rimuovere i riferimenti al contenuto ospitato, in particolare se sono stati creati riferimenti a oggetti non gestiti.
5. Nel file code-behind creare un'istanza della classe di hosting del controllo e impostarla come figlio dell'elemento di riserva. In genere si usa un gestore eventi, ad esempio [Loaded](#), oppure si usa il costruttore della classe parziale. È però anche possibile aggiungere il contenuto di interoperatività tramite un comportamento di runtime.
6. Elaborare i messaggi della finestra selezionati, ad esempio le notifiche dei controlli. Ci sono due approcci. Entrambi offrono un accesso identico al flusso di messaggi, quindi la scelta dipende per lo più dalle esigenze di programmazione.
 - Implementare l'elaborazione del messaggio per tutti i messaggi (non solo per i messaggi di arresto) nell'override del metodo [HwndHost WndProc](#).

- Fare in modo che l'elemento host WPF elabori i messaggi gestendo l'evento [MessageHook](#).
Questo evento viene generato per ogni messaggio inviato alla procedura di finestra principale della finestra ospitata.
- Non è possibile elaborare messaggi da finestre che non sono in corso di elaborazione utilizzando [WndProc](#).

7. Comunicare con la finestra ospitata usando platform invoke per chiamare la funzione [SendMessage](#) non gestita.

Questi passaggi consentono di creare un'applicazione che funziona con l'input del mouse. È possibile aggiungere il supporto di tabulazione per la finestra ospitata implementando l'interfaccia [IKeyboardInputSink](#).

Ognuno di questi passaggi viene illustrato tramite il codice nell'argomento [Procedura dettagliata: hosting di un controllo Win32 in WPF](#).

Handle di finestra (HWND) in WPF

È possibile considerare [HwndHost](#) come un controllo speciale. Tecnicamente, [HwndHost](#) è una classe derivata [FrameworkElement](#), non una classe derivata [Control](#), ma può essere considerata un controllo per l'interoperabilità. [HwndHost](#) astrae la natura Win32 sottostante del contenuto ospitato in modo che il resto del WPF consideri il contenuto ospitato come un altro oggetto simile a un controllo, che deve eseguire il rendering e l'elaborazione dell'input. [HwndHost](#) in genere si comporta in modo analogo a qualsiasi altra WPF [FrameworkElement](#), anche se esistono alcune importanti differenze nell'output (disegno e grafica) e input (mouse e tastiera) in base alle limitazioni di ciò che gli HWND sottostanti sono in grado di supportare.

Differenze rilevanti nel comportamento di output

- [FrameworkElement](#), ovvero la classe di base [HwndHost](#), presenta alcune proprietà che implicano modifiche all'interfaccia utente. Sono incluse proprietà come [FrameworkElement.FlowDirection](#), che modifica il layout degli elementi all'interno di tale elemento come padre. Tuttavia, la maggior parte di queste proprietà non è mappata ai possibili equivalenti Win32, anche se potrebbero esistere tali equivalenti. Poiché un numero elevato di queste proprietà e dei relativi significati è troppo specifico della tecnologia di rendering, il mapping non è una soluzione pratica. Di conseguenza, l'impostazione di proprietà quali [FlowDirection](#) su [HwndHost](#) non ha alcun effetto.
- [HwndHost](#) non possono essere ruotate, ridimensionate, inclinate o modificate in altro modo da una trasformazione.
- [HwndHost](#) non supporta la proprietà [Opacity](#) (fusione alfa). Se il contenuto all'interno del [HwndHost](#) esegue [System.Drawing](#) operazioni che includono informazioni Alpha, non si tratta di una violazione, ma l'[HwndHost](#) nel suo complesso supporta solo l'opacità = 1,0 (100%).
- [HwndHost](#) verranno visualizzati sopra gli altri elementi WPF nella stessa finestra di primo livello. Tuttavia, un menu [ToolTip](#) o [ContextMenu](#) generato è una finestra di primo livello distinta e quindi si comporterà correttamente con [HwndHost](#).
- [HwndHost](#) non rispetta l'area di visualizzazione del [UIElement](#) padre. Si tratta potenzialmente di un problema se si tenta di inserire una classe [HwndHost](#) all'interno di un'area di scorrimento o di un [Canvas](#).

Differenze rilevanti nel comportamento di input

- In generale, mentre i dispositivi di input sono limitati all'ambito della [HwndHost](#) area Win32 ospitata, gli eventi di input passano direttamente a Win32.
- Quando il mouse è posizionato sul [HwndHost](#), l'applicazione non riceve WPF eventi del mouse e il valore della proprietà WPF [IsMouseOver](#) verrà [false](#).
- Mentre il [HwndHost](#) ha lo stato attivo della tastiera, l'applicazione non riceverà WPF eventi della tastiera e il valore della proprietà WPF [IsKeyboardFocusWithin](#) verrà [false](#).

- Quando lo stato attivo è all'interno del [HwndHost](#) e le modifiche apportate a un altro controllo all'interno del [HwndHost](#), l'applicazione non riceverà gli eventi di WPF [GotFocus](#) o [LostFocus](#).
- Le proprietà e gli eventi dello stilo correlati sono analoghi e non segnalano informazioni quando lo stilo è sopra [HwndHost](#).

Tabulazioni, tasti di scelta e tasti di scelta rapida

Le interfacce [IKeyboardInputSink](#) e [IKeyboardInputSite](#) consentono di creare un'esperienza di tastiera senza problemi per le applicazioni WPF e Win32 miste:

- Tabulazione tra componenti Win32 e WPF
- Tasti di scelta e tasti di scelta rapida che funzionano sia quando lo stato attivo è all'interno di un componente Win32 che quando è all'interno di un componente WPF.

Le classi [HwndHost](#) e [HwndSource](#) forniscono entrambe le implementazioni di [IKeyboardInputSink](#), ma non possono gestire tutti i messaggi di input desiderati per scenari più avanzati. Eseguire l'override dei metodi appropriati per ottenere il comportamento di tastiera desiderato.

Le interfacce forniscono solo il supporto per le operazioni eseguite nella transizione tra le aree WPF e Win32. All'interno dell'area Win32, il comportamento di tabulazione è interamente controllato dalla logica implementata da Win32 per l'eventuale tabulazione.

Vedere anche

- [HwndHost](#)
- [HwndSource](#)
- [System.Windows.Interop](#)
- [Procedura dettagliata: Hosting di un controllo Win32 in WPF](#)
- [Procedura dettagliata: hosting di contenuto WPF in Win32](#)

Cenni preliminari sulle aree di tecnologia

10/01/2020 • 6 minutes to read • [Edit Online](#)

Se in un'applicazione si usano più tecnologie di presentazione, ad esempio WPF, Win32 o DirectX, queste devono condividere le aree di rendering all'interno di una finestra comune di primo livello. Questo argomento descrive i problemi che potrebbero influire sulla presentazione e l'input per l'applicazione di interoperatività WPF.

Arearie

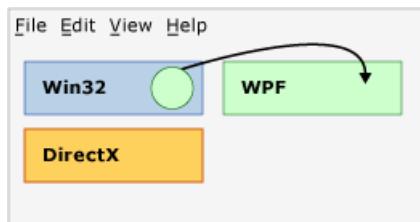
È possibile partire dal concetto che, all'interno di una finestra di primo livello, ogni HWND che include una delle tecnologie di un'applicazione di interoperatività ha una propria area, anche detta "spazio aereo". Ogni pixel all'interno della finestra appartiene esattamente a un HWND e ne costituisce l'area. Per essere precisi, se esistono più HWND WPF, esisteranno più aree di WPF, ma ai fini di questa spiegazione si può presupporre che ne esista uno solo. Il concetto di area implica che tutti i livelli o le altre finestre che provano a eseguire il rendering sopra tale pixel nel corso della durata dell'applicazione devono essere parte della stessa tecnologia a livello di rendering. Il tentativo di eseguire il rendering di WPF pixel su Win32 comporta risultati indesiderati e non è consentito quanto più possibile tramite le API di interoperatività.

Esempi di area

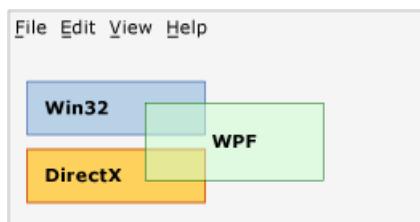
Nella figura seguente viene illustrata un'applicazione che combina Win32, DirectX e WPF. Ogni tecnologia usa un set di pixel specifico distinto, non sovrapposto, per cui non esistono problemi di area.



Si supponga che questa applicazione usi la posizione del puntatore del mouse per creare un'animazione che prova a eseguire il rendering su una qualsiasi di queste tre aree. Indipendentemente dalla tecnologia responsabile dell'animazione stessa, quella tecnologia violerebbe l'area delle altre due. La figura seguente illustra il tentativo di rendering di un cerchio WPF in un'area di Win32.



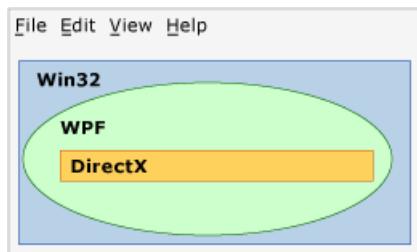
Un'altra violazione si verifica quando si prova a usare la trasparenza o la fusione alfa tra tecnologie diverse. Nella figura seguente la WPF box viola le aree Win32 e DirectX. Poiché i pixel in tale WPF box sono semitransparenti, devono essere di proprietà congiunta sia con DirectX che con WPF, operazione che non è possibile. Pertanto, si tratta di un'altra violazione che rende la compilazione impossibile.



Nei tre esempi precedenti vengono usate aree rettangolari, ma si possono usare altre forme. Ad esempio, un'area può presentare un foro. La figura seguente mostra un'area Win32 con un foro rettangolare, ovvero la dimensione delle aree WPF e DirectX combinate.



Le aree possono anche essere completamente non rettangolari o qualsiasi forma descrittiva da un HRGN Win32 (regione).



Trasparenza e finestre di primo livello

Gestione finestre di Windows elabora solo gli HWND Win32. Pertanto, ogni WPF [Window](#) è un HWND. Il [Window](#) HWND deve rispettare le regole generali per qualsiasi HWND. All'interno di tale HWND, WPF codice può eseguire qualsiasi tipo di supporto per le API WPF complessive. Tuttavia, per le interazioni con altri HWND sul desktop, WPF necessario rispettare le regole di rendering e di elaborazione di Win32. WPF supporta le finestre non rettangolari usando le API Win32, HRGNs per le finestre non rettangolari e le finestre sovrapposte per un Alpha per pixel.

La costante alfa e le chiavi di colore non sono supportate. Le funzionalità della finestra sovrapposta Win32 variano in base alla piattaforma.

Le finestre sovrapposte possono rendere semitrasparente l'intera finestra specificando un valore alfa da applicare a tutti i pixel della finestra. Win32 supporta in realtà l'alfa per pixel, ma è molto difficile da usare nei programmi pratici perché in questa modalità è necessario creare manualmente qualsiasi HWND figlio, inclusi i dialoghi e i menu a discesa.

WPF supporta HRGNs; non sono tuttavia disponibili API gestite per questa funzionalità. È possibile utilizzare [platform invoke](#) e [HwndSource](#) per chiamare le API Win32 pertinenti. Per altre informazioni, vedere [Chiamata di funzioni native da codice gestito](#).

Le finestre sovrapposte di WPF hanno funzionalità diverse in sistemi operativi diversi. Questo perché WPF usa DirectX per eseguire il rendering e le finestre sovrapposte sono state progettate principalmente per il rendering GDI, non per il rendering DirectX.

- WPF supporta le finestre sovrapposte con accelerazione hardware.
- WPF non supporta chiavi di colore trasparenza in quanto WPF non può garantire di eseguire il rendering del colore esatto richiesto, in particolare quando il rendering usa l'accelerazione hardware.

Vedere anche

- [Interoperatività di WPF e Win32](#)
- [Procedura dettagliata: Hosting di un oggetto Clock WPF in Win32](#)
- [Hosting di contenuto Win32 in WPF](#)

Condivisione dei cicli di messaggi tra Win32 e WPF

28/01/2020 • 8 minutes to read • [Edit Online](#)

In questo argomento viene descritto come implementare un ciclo di messaggi per l'interoperabilità con Windows Presentation Foundation (WPF), utilizzando l'esposizione del ciclo di messaggi esistente in [Dispatcher](#) o creando un ciclo di messaggi separato sul lato Win32 del codice di interoperabilità.

ComponentDispatcher e il ciclo di messaggi

Uno scenario normale per l'interoperabilità e il supporto degli eventi da tastiera consiste nell'implementare [IKeyboardInputSink](#) la sottoclasse da classi che implementano già [IKeyboardInputSink](#), ad esempio [HwndSource](#) o [HwndHost](#). Tuttavia, il supporto per i sink di tastiera non risolve tutte le possibili esigenze del ciclo di messaggi durante l'invio e la ricezione di messaggi tra i limiti di interoperatività. Per formalizzare un'architettura del ciclo di messaggi dell'applicazione, Windows Presentation Foundation (WPF) fornisce la classe [ComponentDispatcher](#), che definisce un protocollo semplice per il completamento di un ciclo di messaggi.

[ComponentDispatcher](#) è una classe statica che espone diversi membri. L'ambito di ogni metodo è associato in modo implicito al thread chiamante. Un ciclo di messaggi deve chiamare alcune di queste API in momenti critici, come definito nella sezione successiva.

[ComponentDispatcher](#) fornisce eventi che possono essere ascoltati da altri componenti (ad esempio il sink della tastiera). La classe [Dispatcher](#) chiama tutti i metodi di [ComponentDispatcher](#) appropriati in una sequenza appropriata. Se si implementa un ciclo di messaggi personalizzato, il codice è responsabile della chiamata di [ComponentDispatcher](#) metodi in modo analogo.

La chiamata di [ComponentDispatcher](#) metodi in un thread richiamerà solo i gestori eventi registrati su tale thread.

Scrittura di cicli di messaggi

Di seguito è riportato un elenco di controllo dei membri di [ComponentDispatcher](#) che verrà usato se si scrive il proprio ciclo di messaggi:

- [PushModal](#): il ciclo di messaggi deve chiamare questo oggetto per indicare che il thread è modale.
- [PopModal](#): il ciclo di messaggi deve chiamare questo oggetto per indicare che il thread è stato ripristinato come non modale.
- [Raiseldle](#): il ciclo di messaggi deve chiamare questo oggetto per indicare che [ComponentDispatcher](#) deve generare l'evento di [ThreadIdle](#). [ComponentDispatcher](#) non genererà [ThreadIdle](#) se [IsThreadModal](#) è `true`, ma i cicli di messaggi possono scegliere di chiamare [Raiseldle](#) anche se [ComponentDispatcher](#) non è in grado di rispondere allo stato modale.
- [RaiseThreadMessage](#): il ciclo di messaggi deve chiamare questo oggetto per indicare che è disponibile un nuovo messaggio. Il valore restituito indica se un listener a un evento [ComponentDispatcher](#) ha gestito il messaggio. Se [RaiseThreadMessage](#) restituisce `true` (gestito), il dispatcher non deve eseguire altre operazioni con il messaggio. Se il valore restituito è `false`, è previsto che il dispatcher chiama la funzione Win32 [TranslateMessage](#), quindi chiama [DispatchMessage](#).

Utilizzo di ComponentDispatcher e della gestione dei messaggi esistente

Di seguito è riportato un elenco di controllo dei membri [ComponentDispatcher](#) che si utilizzeranno se si utilizza il

ciclo di messaggi inerente WPF.

- [IsThreadModal](#): restituisce un valore che indica se l'applicazione non è più modale, ad esempio se è stato eseguito il push di un ciclo di messaggi modale. [ComponentDispatcher](#) possibile tenere traccia di questo stato poiché la classe mantiene un conteggio di [PushModal](#) e [PopModal](#) chiamate dal ciclo di messaggi.
- gli eventi [ThreadFilterMessage](#) e [ThreadPreprocessMessage](#) seguono le regole standard per le chiamate di delegati. I delegati vengono richiamati in un ordine non specificato e tutti i delegati vengono richiamati anche se il primo contrassegna il messaggio come gestito.
- [ThreadIdle](#): indica un tempo appropriato ed efficiente per l'elaborazione inattiva (non sono presenti altri messaggi in sospeso per il thread). [ThreadIdle](#) non verrà generato se il thread è modale.
- [ThreadFilterMessage](#): generato per tutti i messaggi elaborati dal message pump.
- [ThreadPreprocessMessage](#): generato per tutti i messaggi che non sono stati gestiti durante [ThreadFilterMessage](#).

Un messaggio viene considerato gestito se dopo l'evento [ThreadFilterMessage](#) o [ThreadPreprocessMessage](#), il parametro `handled` passato per riferimento nei dati dell'evento viene `true`. I gestori di eventi devono ignorare il messaggio se `handled` è `true`, perché questo significa che il messaggio è stato gestito prima dal gestore diverso. I gestori eventi di entrambi gli eventi possono modificare il messaggio. Il dispatcher deve inviare il messaggio modificato e non il messaggio originale non modificato. [ThreadPreprocessMessage](#) viene recapitato a tutti i listener, ma l'intenzione architettonica è che solo la finestra di primo livello contenente l'elemento HWND a cui i messaggi destinati devono richiamare il codice in risposta al messaggio.

modo in cui HwndSource considera gli eventi ComponentDispatcher

Se la [HwndSource](#) è una finestra di primo livello (nessun HWND padre), verrà registrata con [ComponentDispatcher](#). Se [ThreadPreprocessMessage](#) viene generato e se il messaggio è destinato alle finestre figlio o [HwndSource](#), [HwndSource](#) chiama la sequenza di sink della tastiera [IKeyboardInputSink.TranslateAccelerator](#), [TranslateCharOnMnemonic](#).

Se il [HwndSource](#) non è una finestra di primo livello (con un HWND padre), non vi sarà alcuna gestione. Per la gestione è prevista solo la finestra di primo livello ed è prevista una finestra di primo livello con supporto per il sink della tastiera come parte di uno scenario di interoperabilità.

Se [WndProc](#) su un [HwndSource](#) viene chiamato senza che venga chiamato per primo un metodo di sink della tastiera appropriato, l'applicazione riceverà gli eventi di tastiera di livello superiore, ad esempio [KeyDown](#). Tuttavia, non verrà chiamato alcun metodo di sink della tastiera, che elude le funzionalità desiderate del modello di input da tastiera, ad esempio il supporto della chiave di accesso. Questo problema potrebbe verificarsi perché il ciclo di messaggi non ha notificato correttamente al thread pertinente il [ComponentDispatcher](#)o perché l'elemento HWND padre non ha richiamato le risposte appropriate per il sink della tastiera.

Un messaggio che va al sink della tastiera potrebbe non essere inviato a HWND se sono stati aggiunti hook per quel messaggio usando il metodo [AddHook](#). Il messaggio potrebbe essere stato gestito a livello di message pump direttamente e non inviato alla funzione [DispatchMessage](#).

Vedere anche

- [ComponentDispatcher](#)
- [IKeyboardInputSink](#)
- [Interoperatività di WPF e Win32](#)
- [Modello di threading](#)
- [Cenni preliminari sull'input](#)

Hosting di contenuto Win32 in WPF

27/01/2020 • 13 minutes to read • [Edit Online](#)

Prerequisiti

Vedere l' [interoperatività di WPF e Win32](#).

Procedura dettagliata di Win32 all'interno di Windows Presentation Framework (HwndHost)

Per riutilizzare il contenuto Win32 all'interno WPF applicazioni, utilizzare [HwndHost](#), ovvero un controllo che rende gli [HWND](#) simili a WPF contenuto. Come [HwndSource](#), [HwndHost](#) è semplice da usare: derivare da [HwndHost](#) e implementare i metodi [BuildWindowCore](#) e [DestroyWindowCore](#), quindi creare un'istanza della classe derivata [HwndHost](#) e inserirla all'interno dell'applicazione WPF.

Se la logica Win32 è già inclusa in un pacchetto come controllo, l'implementazione del [BuildWindowCore](#) è poco più di una chiamata a [CreateWindow](#). Ad esempio, per creare un controllo LISTBOX Win32 in C++:

```
virtual HandleRef BuildWindowCore(HandleRef hwndParent) override {
    HWND handle = CreateWindowEx(0, L"LISTBOX",
        L"this is a Win32 listbox",
        WS_CHILD | WS_VISIBLE | LBS_NOTIFY
        | WS_VSCROLL | WS_BORDER,
        0, 0, // x, y
        30, 70, // height, width
        (HWND) hwndParent.Handle.ToPointer(), // parent hwnd
        0, // hmenu
        0, // hinstance
        0); // lparam

    return HandleRef(this, IntPtr(handle));
}

virtual void DestroyWindowCore(HandleRef hwnd) override {
    // HwndHost will dispose the hwnd for us
}
```

Ma si supponga che il codice Win32 non sia abbastanza indipendente? In tal caso, è possibile creare una finestra di dialogo Win32 e incorporarne il contenuto in un'applicazione WPF più grande. L'esempio mostra questo aspetto in Visual Studio C++ e, sebbene sia anche possibile eseguire questa operazione in un linguaggio diverso o nella riga di comando.

Iniziare con una semplice finestra di dialogo, compilata C++ in un progetto DLL.

Successivamente, introdurre la finestra di dialogo nell'applicazione WPF più grande:

- Compila la DLL come gestita ([/clr](#))
- Trasformare la finestra di dialogo in un controllo
- Definire la classe derivata di [HwndHost](#) con i metodi [BuildWindowCore](#) e [DestroyWindowCore](#)
- Eseguire l'override del metodo [TranslateAccelerator](#) per gestire le chiavi di dialogo
- Eseguire l'override del metodo [TabInto](#) per supportare la tabulazione

- Eseguire l'override `OnMnemonic` metodo per supportare i tasti di scelta
- Creare un'istanza della sottoclasse `HwndHost` e inserirla nell'elemento WPF corretto

Trasformare la finestra di dialogo in un controllo

È possibile trasformare una finestra di dialogo in un HWND figlio utilizzando gli stili WS_CHILD e DS_CONTROL. Passare al file di risorse (.RC) in cui è definita la finestra di dialogo e trovare l'inizio della definizione della finestra di dialogo:

```
IDD_DIALOG1 DIALOGEX 0, 0, 303, 121
STYLE DS_SETFONT | DS_MODALFRAME | DS_FIXEDSYS | WS_POPUP | WS_CAPTION | WS_SYSMENU
```

Modificare la seconda riga in:

```
STYLE DS_SETFONT | WS_CHILD | WS_BORDER | DS_CONTROL
```

Questa azione non esegue completamente il pacchetto in un controllo indipendente; è comunque necessario chiamare `IsDialogMessage()` in modo che Win32 possa elaborare determinati messaggi, ma la modifica del controllo fornisce un modo semplice per inserire i controlli all'interno di un altro HWND.

Classe HwndHost

Importare gli spazi dei nomi seguenti:

```
namespace ManagedCpp
{
    using namespace System;
    using namespace System::Windows;
    using namespace System::Windows::Interop;
    using namespace System::Windows::Input;
    using namespace System::Windows::Media;
    using namespace System::Runtime::InteropServices;
```

Creare quindi una classe derivata di `HwndHost` ed eseguire l'override dei metodi `BuildWindowCore` e `DestroyWindowCore`:

```
public ref class MyHwndHost : public HwndHost, IKeyboardInputSink {
private:
    HWND dialog;

protected:
    virtual HandleRef BuildWindowCore(HandleRef hwndParent) override {
        InitializeGlobals();
        dialog = CreateDialog(hInstance,
            MAKEINTRESOURCE(IDD_DIALOG1),
            (HWND) hwndParent.Handle.ToPointer(),
            (DLGPROC) About);
        return HandleRef(this, IntPtr(dialog));
    }

    virtual void DestroyWindowCore(HandleRef hwnd) override {
        // hwnd will be disposed for us
    }
}
```

Qui si usa il `CreateDialog` per creare la finestra di dialogo che è effettivamente un controllo. Poiché si tratta di uno dei primi metodi chiamati all'interno della DLL, è necessario eseguire anche un'inizializzazione Win32 standard chiamando una funzione che verrà definita in seguito, denominata `InitializeGlobals()`:

```
bool initialized = false;
void InitializeGlobals() {
    if (initialized) return;
    initialized = true;

    // TODO: Place code here.
    MSG msg;
    HACCEL hAccelTable;

    // Initialize global strings
    LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadString(hInstance, IDC_TYPICALWIN32DIALOG, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);
```

Eseguire l'override del metodo TranslateAccelerator per gestire le chiavi di dialogo

Se è stato eseguito questo esempio, si otterrebbe un controllo finestra di dialogo che viene visualizzato, ma si ignorerà tutta l'elaborazione della tastiera che rende una finestra di dialogo funzionante. A questo punto, è necessario eseguire l'override dell'implementazione di `TranslateAccelerator` (che deriva da `IKeyboardInputSink`, un'interfaccia che `HwndHost` implementa). Questo metodo viene chiamato quando l'applicazione riceve WM_KEYDOWN e WM_SYSKEYDOWN.

```

#define TranslateAccelerator
    virtual bool TranslateAccelerator(System::Windows::Interop::MSG% msg,
        ModifierKeys modifiers) override
    {
        ::MSG m = ConvertMessage(msg);

        // Win32's IsDialogMessage() will handle most of our tabbing, but doesn't know
        // what to do when it reaches the last tab stop
        if (m.message == WM_KEYDOWN && m.wParam == VK_TAB) {
            HWND firstTabStop = GetDlgItem(dialog, IDC_EDIT1);
            HWND lastTabStop = GetDlgItem(dialog, IDCANCEL);
            TraversalRequest^ request = nullptr;

            if (GetKeyState(VK_SHIFT) && GetFocus() == firstTabStop) {
                // this code should work, but there's a bug with interop shift-tab in current builds
                request = gcnew TraversalRequest(FocusNavigationDirection::Last);
            }
            else if (!GetKeyState(VK_SHIFT) && GetFocus() == lastTabStop) {
                request = gcnew TraversalRequest(FocusNavigationDirection::Next);
            }

            if (request != nullptr)
                return ((IKeyboardInputSink^) this)->KeyboardInputSite->OnNoMoreTabStops(request);
        }

        // Only call IsDialogMessage for keys it will do something with.
        if (msg.message == WM_SYSKEYDOWN || msg.message == WM_KEYDOWN) {
            switch (m.wParam) {
                case VK_TAB:
                case VK_LEFT:
                case VK_UP:
                case VK_RIGHT:
                case VK_DOWN:
                case VK_EXECUTE:
                case VK_RETURN:
                case VK_ESCAPE:
                case VK_CANCEL:
                    IsDialogMessage(dialog, &m);
                    // IsDialogMessage should be called ProcessDialogMessage --
                    // it processes messages without ever really telling you
                    // if it handled a specific message or not
                    return true;
            }
        }
    }

    return false; // not a key we handled
}

```

Si tratta di una grande quantità di codice in un pezzo, quindi è possibile usare alcune spiegazioni più dettagliate. Per prima cosa, il C++ codice C++ usa le macro e; è necessario tenere presente che esiste già una macro denominata `TranslateAccelerator`, definita in `winuser.h`:

```
#define TranslateAccelerator TranslateAcceleratorW
```

Assicurarsi quindi di definire un metodo di `TranslateAccelerator` e non un metodo di `TranslateAcceleratorW`.

Analogamente, sono presenti sia il messaggio `winuser.h` non gestito che lo struct `Microsoft::Win32::MSG` gestito. È possibile evitare ambiguità tra le due utilizzando l' C++ operatore `::`.

```

virtual bool TranslateAccelerator(System::Windows::Interop::MSG% msg,
    ModifierKeys modifiers) override
{
    ::MSG m = ConvertMessage(msg);
}

```

Both MSGs have the same data, but sometimes it is easier to work with the unmanaged definition, so in this sample you can define the obvious conversion routine:

```

```cpp
::MSG ConvertMessage(System::Windows::Interop::MSG% msg) {
 ::MSG m;
 m.hwnd = (HWND) msg(hwnd).ToPointer();
 m.lParam = (LPARAM) msg(lParam).ToPointer();
 m.message = msg.message;
 m.wParam = (WPARAM) msg(wParam).ToPointer();

 m.time = msg.time;

 POINT pt;
 pt.x = msg.pt_x;
 pt.y = msg.pt_y;
 m.pt = pt;

 return m;
}

```

Tornare a `TranslateAccelerator`. Il principio di base consiste nel chiamare la funzione Win32 `IsDialogMessage` per eseguire il maggior lavoro possibile, ma `IsDialogMessage` non può accedere ad alcun elemento all'esterno della finestra di dialogo. Come scheda utente intorno alla finestra di dialogo, quando la tabulazione viene eseguita oltre l'ultimo controllo nella finestra di dialogo, è necessario impostare lo stato attivo sulla parte WPF chiamando `IKeyboardInputSite::OnNoMoreStops`.

```

// Win32's IsDialogMessage() will handle most of the tabbing, but doesn't know
// what to do when it reaches the last tab stop
if (m.message == WM_KEYDOWN && m.wParam == VK_TAB) {
 HWND firstTabStop = GetDlgItem(dialog, IDC_EDIT1);
 HWND lastTabStop = GetDlgItem(dialog, IDCANCEL);
 TraversalRequest^ request = nullptr;

 if (GetKeyState(VK_SHIFT) && GetFocus() == firstTabStop) {
 request = gcnew TraversalRequest(FocusNavigationDirection::Last);
 }
 else if (!GetKeyState(VK_SHIFT) && GetFocus() == lastTabStop) {
 request = gcnew TraversalRequest(FocusNavigationDirection::Next);
 }

 if (request != nullptr)
 return ((IKeyboardInputSink^) this)->KeyboardInputSite->OnNoMoreTabStops(request);
}

```

Infine, viene chiamato `IsDialogMessage`. Tuttavia, una delle responsabilità di un metodo di `TranslateAccelerator` indica WPF se la sequenza di tasti è stata gestita o meno. Se non è stato gestito, l'evento di input può eseguire il tunneling e la bolla nel resto dell'applicazione. In questo caso si esporrà una peculiarità della gestione questo caso della tastiera e della natura dell'architettura di input in Win32. Sfortunatamente, `IsDialogMessage` non restituisce alcun modo se gestisce una particolare sequenza di tasti. Ancora peggio, chiamerà `DispatchMessage()` sulle sequenze di tasti che non devono essere gestite. Sarà quindi necessario decompilare `IsDialogMessage` e chiamarlo solo per le chiavi che si sa gestire:

```

// Only call IsDialogMessage for keys it will do something with.
if (msg.message == WM_SYSKEYDOWN || msg.message == WM_KEYDOWN) {
 switch (m.wParam) {
 case VK_TAB:
 case VK_LEFT:
 case VK_UP:
 case VK_RIGHT:
 case VK_DOWN:
 case VK_EXECUTE:
 case VK_RETURN:
 case VK_ESCAPE:
 case VK_CANCEL:
 IsDialogMessage(dialog, &m);
 // IsDialogMessage should be called ProcessDialogMessage --
 // it processes messages without ever really telling you
 // if it handled a specific message or not
 return true;
 }
}

```

### Eseguire l'override del metodo TabInto per supportare la tabulazione

Ora che è stata implementata `TranslateAccelerator`, un utente può spostarsi all'interno della finestra di dialogo e visualizzarne la scheda in un'applicazione WPF maggiore. Tuttavia, un utente non può riportare la scheda nella finestra di dialogo. Per risolvere il problema, è necessario eseguire l'override di `TabInto`:

```

public:
 virtual bool TabInto(TraversalRequest^ request) override {
 if (request->FocusNavigationDirection == FocusNavigationDirection::Last) {
 HWND lastTabStop = GetDlgItem(dialog, IDCANCEL);
 SetFocus(lastTabStop);
 }
 else {
 HWND firstTabStop = GetDlgItem(dialog, IDC_EDIT1);
 SetFocus(firstTabStop);
 }
 return true;
 }
}

```

Il parametro `TraversalRequest` indica se l'azione della scheda è una scheda o una scheda di spostamento.

### Eseguire l'override del metodo OnMnemonic per supportare i tasti di scelta

La gestione della tastiera è quasi completa, ma manca un elemento: i tasti di scelta non funzionano. Se un utente preme ALT + F, lo stato attivo non passa alla casella di modifica "First Name:". Quindi, si esegue l'override del metodo `OnMnemonic`:

```

virtual bool OnMnemonic(System::Windows::Interop::MSG% msg, ModifierKeys modifiers) override {
 ::MSG m = ConvertMessage(msg);

 // If it's one of our mnemonics, set focus to the appropriate hwnd
 if (msg.message == WM_SYSCHAR && GetKeyState(VK_MENU /*alt*/) < 0) {
 int dialogitem = 9999;
 switch (m.wParam) {
 case 's': dialogitem = IDOK; break;
 case 'c': dialogitem = IDCANCEL; break;
 case 'f': dialogitem = IDC_EDIT1; break;
 case 'l': dialogitem = IDC_EDIT2; break;
 case 'p': dialogitem = IDC_EDIT3; break;
 case 'a': dialogitem = IDC_EDIT4; break;
 case 'i': dialogitem = IDC_EDIT5; break;
 case 't': dialogitem = IDC_EDIT6; break;
 case 'z': dialogitem = IDC_EDIT7; break;
 }
 if (dialogitem != 9999) {
 HWND hwnd = GetDlgItem(dialog, dialogitem);
 SetFocus(hwnd);
 return true;
 }
 }
 return false; // key unhandled
};


```

Perché non chiamare `IsDialogMessage` qui? Si ha lo stesso problema di prima: è necessario essere in grado di informare WPF codice se il codice ha gestito o meno la sequenza di tasti e `IsDialogMessage` non è possibile eseguire questa operazione. Esiste anche un secondo problema, perché `IsDialogMessage` rifiuta di elaborare il tasto di scelta se l'HWND con stato attivo non si trova all'interno della finestra di dialogo.

### Creare un'istanza della classe derivata `HwndHost`

Infine, ora che tutto il supporto per chiavi e schede è disponibile, è possibile inserire il `HwndHost` nell'applicazione WPF più grande. Se l'applicazione principale viene scritta in XAML, il modo più semplice per inserirlo nella posizione corretta consiste nel lasciare un elemento `Border` vuoto in cui si desidera inserire il `HwndHost`. Qui viene creato un `Border` denominato `insertHwndHostHere`:

```

<Window x:Class="WPFApplication1.Window1"
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Windows Presentation Framework Application"
 Loaded="Window1_Loaded"
 >
 <StackPanel>
 <Button Content="WPF button"/>
 <Border Name="insertHwndHostHere" Height="200" Width="500"/>
 <Button Content="WPF button"/>
 </StackPanel>
</Window>

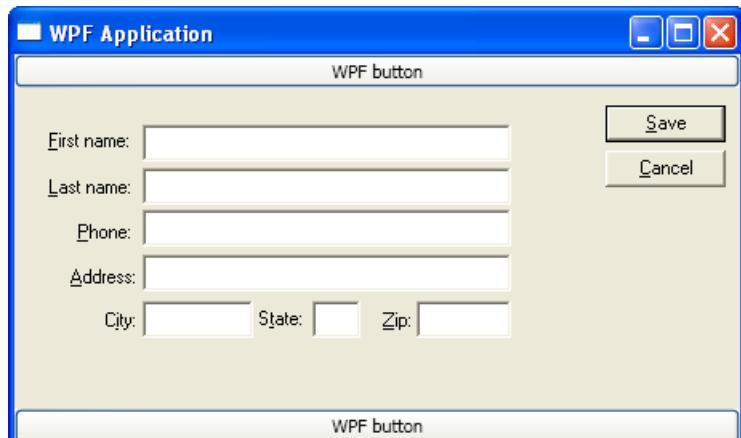
```

Quindi, tutto ciò che rimane è trovare un posto valido nella sequenza di codice per creare un'istanza del `HwndHost` e connetterlo al `Border`. In questo esempio, verrà inserito nel costruttore per la classe derivata `Window`:

```
public partial class Window1 : Window {
 public Window1() {
 }

 void Window1_Loaded(object sender, RoutedEventArgs e) {
 HwndHost host = new ManagedCpp.MyHwndHost();
 insertHwndHostHere.Child = host;
 }
}
```

Che offre:



## Vedere anche

- [Interoperatività di WPF e Win32](#)

# Procedura dettagliata: ospitare un controllo Win32 in WPF

03/02/2020 • 25 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) fornisce un ambiente completo per la creazione di applicazioni. Tuttavia, quando si ha un investimento sostanziale nel codice Win32, potrebbe essere più efficace riutilizzare almeno parte del codice nell'applicazione WPF anziché riscriverlo completamente. WPF fornisce un meccanismo semplice per l'hosting di una finestra Win32, in una pagina WPF.

In questo argomento viene illustrata un'applicazione che ospita un controllo [ListBox Win32 nell'esempio WPF](#) che ospita un controllo casella di riepilogo Win32. Questa procedura generale può essere estesa per ospitare qualsiasi finestra di Win32.

## Requisiti

In questo argomento si presuppone una conoscenza di base di WPF e della programmazione dell'API Windows. Per un'introduzione di base alla programmazione WPF, vedere [Introduzione](#). Per un'introduzione alla programmazione dell'API Windows, vedere uno dei numerosi libri sull'argomento, in particolare *programmare Windows* di Charles Petzold.

Poiché l'esempio che accompagna questo argomento viene implementato in C#, USA i servizi di chiamata della piattaforma (PInvoke) per accedere all'API Windows. Una certa familiarità con PInvoke è utile ma non essenziale.

### NOTE

Questo argomento include vari esempi di codice tratti dall'esempio associato. Tuttavia, per una questione di leggibilità, il codice di esempio completo non è compreso. È possibile ottenere o visualizzare il codice completo dall'[hosting di un controllo ListBox Win32 nell'esempio WPF](#).

## Procedura di base

Questa sezione descrive la procedura di base per l'hosting di una finestra Win32 in una pagina WPF. Le sezioni rimanenti illustrano in dettaglio i vari passaggi.

La procedura di hosting base è la seguente:

1. Implementare una pagina WPF per ospitare la finestra. Una tecnica consiste nel creare un elemento [Border](#) per riservare una sezione della pagina per la finestra ospitata.
2. Implementare una classe per ospitare il controllo che eredita da [HwndHost](#).
3. In tale classe, eseguire l'override del membro della classe [HwndHost BuildWindowCore](#).
4. Creare la finestra ospitata come figlio della finestra che contiene la pagina WPF. Sebbene la programmazione WPF convenzionale non debba utilizzarla in modo esplicito, la pagina di hosting è una finestra con un handle (HWND). Si riceve la pagina HWND tramite il parametro `hwndParent` del metodo [BuildWindowCore](#). La finestra ospitata deve essere creata come elemento figlio dell'oggetto HWND.
5. Dopo aver creato la finestra host, restituire l'oggetto HWND della finestra ospitata. Se si desidera ospitare uno o più controlli Win32, in genere si crea una finestra host come elemento figlio di HWND e si rendono i controlli figlio di tale finestra host. Il wrapping dei controlli in una finestra host offre un modo semplice per

la ricezione delle notifiche dai controlli da parte della pagina WPF, che riguarda alcuni problemi Win32 specifici con le notifiche attraverso il limite HWND.

6. Gestire i messaggi selezionati inviati alla finestra host, ad esempio notifiche dai controlli figlio. Questa operazione può essere eseguita in due modi.

- Se si preferisce gestire i messaggi nella classe host, eseguire l'override del metodo [WndProc](#) della classe [HwndHost](#).
- Se si preferisce che WPF gestisca i messaggi, gestire la classe [HwndHost MessageHook](#) evento nel code-behind. Questo evento si verifica per ogni messaggio ricevuto dalla finestra ospitata. Se si sceglie questa opzione, è comunque necessario eseguire l'override di [WndProc](#), ma è necessaria solo un'implementazione minima.

7. Eseguire l'override dei metodi [DestroyWindowCore](#) e [WndProc](#) di [HwndHost](#). È necessario eseguire l'override di questi metodi per soddisfare il contratto di [HwndHost](#), ma potrebbe essere necessario fornire solo un'implementazione minima.

8. Nel file code-behind creare un'istanza della classe di hosting del controllo e impostarla come figlio dell'elemento [Border](#) che deve ospitare la finestra.

9. Comunicare con la finestra ospitata inviando i messaggi di Microsoft Windows e gestendo i messaggi dalle relative finestre figlio, ad esempio le notifiche inviate dai controlli.

## Implementare il layout di pagina

Il layout per la pagina WPF che ospita il controllo ListBox è costituito da due aree. Il lato sinistro della pagina ospita diversi controlli WPF che forniscono un interfaccia utente che consente di modificare il controllo Win32. L'angolo superiore destro della pagina include un'area quadrata per il controllo ListBox ospitato.

Il codice per implementare questo layout è piuttosto semplice. L'elemento radice è un [DockPanel](#) che dispone di due elementi figlio. Il primo è un elemento [Border](#) che ospita il controllo ListBox. Occupa un quadrato 200x200 nell'angolo superiore destro della pagina. Il secondo è un elemento [StackPanel](#) che contiene un set di controlli WPF che visualizzano informazioni e consentono di modificare il controllo ListBox impostando proprietà di interoperatività esposte. Per ogni elemento figlio del [StackPanel](#), vedere il materiale di riferimento per i vari elementi utilizzati per informazioni dettagliate su questi elementi o sul loro funzionamento, elencati nel codice di esempio riportato di seguito, ma non verranno illustrati in questo articolo (il modello di interoperatività di base non richiede alcuno di essi, viene fornito per aggiungere alcune interattività all'esempio).

```

<Window
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 x:Class="WPF_Hosting_Win32_Control.HostWindow"
 Name="mainWindow"
 Loaded="On_UIReady">

 <DockPanel Background="LightGreen">
 <Border Name="ControlHostElement"
 Width="200"
 Height="200"
 HorizontalAlignment="Right"
 VerticalAlignment="Top"
 BorderBrush="LightGray"
 BorderThickness="3"
 DockPanel.Dock="Right"/>
 <StackPanel>
 <Label HorizontalAlignment="Center"
 Margin="0,10,0,0"
 FontSize="14"
 FontWeight="Bold">Control the Control</Label>
 <TextBlock Margin="10,10,10,10" >Selected Text: <TextBlock Name="selectedText"/></TextBlock>
 <TextBlock Margin="10,10,10,10" >Number of Items: <TextBlock Name="numItems"/></TextBlock>

 <Line X1="0" X2="200"
 Stroke="LightYellow"
 StrokeThickness="2"
 HorizontalAlignment="Center"
 Margin="0,20,0,0"/>

 <Label HorizontalAlignment="Center"
 Margin="10,10,10,10">Append an Item to the List</Label>
 <StackPanel Orientation="Horizontal">
 <Label HorizontalAlignment="Left"
 Margin="10,10,10,10">Item Text</Label>
 <TextBox HorizontalAlignment="Left"
 Name="txtAppend"
 Width="200"
 Margin="10,10,10,10"></TextBox>
 </StackPanel>

 <Button HorizontalAlignment="Left"
 Click="AppendText"
 Width="75"
 Margin="10,10,10,10">Append</Button>

 <Line X1="0" X2="200"
 Stroke="LightYellow"
 StrokeThickness="2"
 HorizontalAlignment="Center"
 Margin="0,20,0,0"/>

 <Label HorizontalAlignment="Center"
 Margin="10,10,10,10">Delete the Selected Item</Label>

 <Button Click="DeleteText"
 Width="125"
 Margin="10,10,10,10"
 HorizontalAlignment="Left">Delete</Button>
 </StackPanel>
 </DockPanel>
</Window>

```

Implementare una classe per l'hosting del controllo Microsoft Win32

La parte centrale di questo esempio è la classe che ospita effettivamente il controllo, ControlHost.cs. Eredita da [HwndHost](#). Il costruttore accetta due parametri, Height e Width, che corrispondono all'altezza e alla larghezza dell'elemento [Border](#) che ospita il controllo ListBox. Questi valori vengono usati in un secondo momento per assicurarsi che la dimensione del controllo corrisponda all'elemento [Border](#).

```
public class ControlHost : HwndHost
{
 IntPtr hwndControl;
 IntPtr hwndHost;
 int hostHeight, hostWidth;

 public ControlHost(double height, double width)
 {
 hostHeight = (int)height;
 hostWidth = (int)width;
 }
}
```

```
Public Class ControlHost
 Inherits HwndHost
 Private hwndControl As IntPtr
 Private hwndHost As IntPtr
 Private hostHeight, hostWidth As Integer

 Public Sub New(ByVal height As Double, ByVal width As Double)
 hostHeight = CInt(height)
 hostWidth = CInt(width)
 End Sub
```

È anche presente un insieme di costanti. Queste costanti vengono ricavate principalmente da winuser. h e consentono di usare nomi convenzionali quando si chiamano funzioni Win32.

```
internal const int
WS_CHILD = 0x40000000,
WS_VISIBLE = 0x10000000,
LBS_NOTIFY = 0x00000001,
HOST_ID = 0x00000002,
LISTBOX_ID = 0x00000001,
WS_VSCROLL = 0x00200000,
WS_BORDER = 0x00800000;
```

```
Friend Const WS_CHILD As Integer = &H40000000, WS_VISIBLE As Integer = &H10000000, LBS_NOTIFY As Integer =
&H00000001, HOST_ID As Integer = &H00000002, LISTBOX_ID As Integer = &H00000001, WS_VSCROLL As Integer =
&H00200000, WS_BORDER As Integer = &H00800000
```

## Eseguire l'override di BuildWindowCore per creare la finestra Microsoft Win32

È possibile eseguire l'override di questo metodo per creare la finestra Win32 che verrà ospitata dalla pagina e per effettuare la connessione tra la finestra e la pagina. Poiché questo esempio include l'hosting di un controllo ListBox vengono create due finestre. Il primo è la finestra che è effettivamente ospitata dalla pagina WPF. Il controllo ListBox viene creato come elemento figlio di questa finestra.

Lo scopo di questo approccio è semplificare il processo di ricezione di notifiche dal controllo. La classe [HwndHost](#) consente di elaborare i messaggi inviati alla finestra ospitata. Se si ospita un controllo Win32 direttamente, vengono ricevuti i messaggi inviati al ciclo di messaggi interno del controllo. È possibile visualizzare il controllo e inviare a esso messaggi ma non ricevere le notifiche che il controllo invia alla propria finestra padre. Ciò significa, tra l'altro, che non è possibile in alcun modo rilevare quando l'utente interagisce con il controllo. Creare invece una finestra host e impostare il controllo come elemento figlio di tale finestra. Ciò consente di elaborare i

messaggi per la finestra host, incluse le notifiche inviate a essa dal controllo. Per comodità, dal momento che la finestra host è poco più di un semplice wrapper per il controllo, il pacchetto verrà restituito come un controllo **ListBox**.

#### Creare la finestra host e il controllo **ListBox**

È possibile utilizzare **PInvoke** per creare una finestra host per il controllo creando e registrando una classe della finestra e così via. Tuttavia, un approccio molto più semplice consiste nel creare una finestra con la classe della finestra "statica" predefinita. Questo approccio rende disponibile la procedura della finestra necessaria per ricevere le notifiche dal controllo e richiede codice minimo.

L'oggetto **HWND** del controllo è esposto tramite una proprietà di sola lettura di modo che la pagina host possa usarlo per inviare messaggi al controllo.

```
public IntPtr hwndListBox
{
 get { return hwndControl; }
}
```

```
Public ReadOnly Property hwndListBox() As IntPtr
 Get
 Return hwndControl
 End Get
End Property
```

Il controllo **ListBox** viene creato come elemento figlio della finestra host. L'altezza e la larghezza di entrambe le finestre sono impostate sui valori passati dal costruttore, come illustrato in precedenza. Ciò garantisce che le dimensioni della finestra host e del controllo siano identiche all'area riservata nella pagina. Una volta create le finestre, l'esempio restituisce un [HandleRef](#) oggetto che contiene l'**HWND** della finestra host.

```
protected override HandleRef BuildWindowCore(HandleRef hwndParent)
{
 hwndControl = IntPtr.Zero;
 hwndHost = IntPtr.Zero;

 hwndHost = CreateWindowEx(0, "static", "",
 WS_CHILD | WS_VISIBLE,
 0, 0,
 hostWidth, hostHeight,
 hwndParent.Handle,
 (IntPtr)HOST_ID,
 IntPtr.Zero,
 0);

 hwndControl = CreateWindowEx(0, "listbox", "",
 WS_CHILD | WS_VISIBLE | LBS_NOTIFY
 | WS_VSCROLL | WS_BORDER,
 0, 0,
 hostWidth, hostHeight,
 hwndHost,
 (IntPtr) LISTBOX_ID,
 IntPtr.Zero,
 0);

 return new HandleRef(this, hwndHost);
}
```

```

Protected Overrides Function BuildWindowCore(ByVal hwndParent As HandleRef) As HandleRef
 hwndControl = IntPtr.Zero
 hwndHost = IntPtr.Zero

 hwndHost = CreateWindowEx(0, "static", "", WS_CHILD Or WS_VISIBLE, 0, 0, hostWidth, hostHeight,
 hwndParent.Handle, New IntPtr(HOST_ID), IntPtr.Zero, 0)

 hwndControl = CreateWindowEx(0, "listbox", "", WS_CHILD Or WS_VISIBLE Or LBS_NOTIFY Or WS_VSCROLL Or
 WS_BORDER, 0, 0, hostWidth, hostHeight, hwndHost, New IntPtr(LISTBOX_ID), IntPtr.Zero, 0)

 Return New HandleRef(Me, hwndHost)
End Function

```

```

//PInvoke declarations
[DllImport("user32.dll", EntryPoint = "CreateWindowEx", CharSet = CharSet.Unicode)]
internal static extern IntPtr CreateWindowEx(int dwExStyle,
 string lpszClassName,
 string lpszWindowName,
 int style,
 int x, int y,
 int width, int height,
 IntPtr hwndParent,
 IntPtr hMenu,
 IntPtr hInst,
 [MarshalAs(UnmanagedType.AsAny)] object pvParam);

```

```

'PInvoke declarations
<DllImport("user32.dll", EntryPoint := "CreateWindowEx", CharSet := CharSet.Unicode)>
Friend Shared Function CreateWindowEx(ByVal dwExStyle As Integer, ByVal lpszClassName As String, ByVal
lpszWindowName As String, ByVal style As Integer, ByVal x As Integer, ByVal y As Integer, ByVal width As
Integer, ByVal height As Integer, ByVal hwndParent As IntPtr, ByVal hMenu As IntPtr, ByVal hInst As IntPtr,
<MarshalAs(UnmanagedType.AsAny)> ByVal pvParam As Object) As IntPtr
End Function

```

## Implementare DestroyWindow e WndProc

Oltre a [BuildWindowCore](#), è necessario eseguire l'override anche dei metodi [WndProc](#) e [DestroyWindowCore](#) della [HwndHost](#). In questo esempio, i messaggi per il controllo vengono gestiti dal gestore di [MessageHook](#), quindi l'implementazione di [WndProc](#) e [DestroyWindowCore](#) è minima. Nel caso di [WndProc](#), impostare `handled` su `false` per indicare che il messaggio non è stato gestito e restituire 0. Per [DestroyWindowCore](#), è sufficiente eliminare la finestra.

```

protected override IntPtr WndProc(IntPtr hwnd, int msg, IntPtr wParam, IntPtr lParam, ref bool handled)
{
 handled = false;
 return IntPtr.Zero;
}

protected override void DestroyWindowCore(HandleRef hwnd)
{
 DestroyWindow(hwnd.Handle);
}

```

```

Protected Overrides Function WndProc(ByVal hwnd As IntPtr, ByVal msg As Integer, ByVal wParam As IntPtr,
ByVal lParam As IntPtr, ByRef handled As Boolean) As IntPtr
 handled = False
 Return IntPtr.Zero
End Function

Protected Overrides Sub DestroyWindowCore(ByVal hwnd As HandleRef)
 DestroyWindow(hwnd.Handle)
End Sub

```

```

[DllImport("user32.dll", EntryPoint = "DestroyWindow", CharSet = CharSet.Unicode)]
internal static extern bool DestroyWindow(IntPtr hwnd);

```

```

<DllImport("user32.dll", EntryPoint := "DestroyWindow", CharSet := CharSet.Unicode)>
Friend Shared Function DestroyWindow(ByVal hwnd As IntPtr) As Boolean
End Function

```

## Ospitare il controllo nella pagina

Per ospitare il controllo nella pagina, creare prima di tutto una nuova istanza della classe `ControlHost`. Passare l'altezza e la larghezza dell'elemento Border che contiene il controllo (`ControlHostElement`) al costruttore di `ControlHost`. Questo garantisce che le dimensioni di ListBox siano corrette. È quindi possibile ospitare il controllo nella pagina assegnando l'oggetto `controlHost` alla proprietà `Child` della `Borderhost`.

L'esempio connette un gestore all'evento `MessageHook` della `ControlHost` per ricevere messaggi dal controllo. Questo evento viene generato per ogni messaggio inviato alla finestra ospitata. In questo caso, si tratta dei messaggi inviati alla finestra che esegue il wrapping del controllo ListBox effettivo, incluse le notifiche dal controllo. L'esempio chiama `SendMessage` per ottenere informazioni dal controllo e ne modifica il contenuto. I dettagli di come la pagina comunica con il controllo sono illustrati nella sezione successiva.

### NOTE

Si noti che sono presenti due dichiarazioni `PInvoke` per `SendMessage`. Questa operazione è necessaria perché si usa il parametro `wParam` per passare una stringa e l'altra la usa per passare un valore integer. È necessaria una dichiarazione separata per ogni firma per assicurare che venga eseguito correttamente il marshalling dei dati.

```

public partial class HostWindow : Window
{
 int selectedItem;
 IntPtr hwndListBox;
 ControlHost listControl;
 Application app;
 Window myWindow;
 int itemCount;

 private void On_UIReady(object sender, EventArgs e)
 {
 app = System.Windows.Application.Current;
 myWindow = app.MainWindow;
 myWindow.SizeToContent = SizeToContent.WidthAndHeight;
 listControl = new ControlHost(ControlHostElement.ActualHeight, ControlHostElement.ActualWidth);
 ControlHostElement.Child = listControl;
 listControl.MessageHook += new HwndSourceHook(ControlMsgFilter);
 hwndListBox = listControl(hwndListBox);
 for (int i = 0; i < 15; i++) //populate listbox
 {
 string itemText = "Item" + i.ToString();
 SendMessage(hwndListBox, LB_ADDSTRING, IntPtr.Zero, itemText);
 }
 itemCount = SendMessage(hwndListBox, LB_GETCOUNT, IntPtr.Zero, IntPtr.Zero);
 numItems.Text = "" + itemCount.ToString();
 }
}

```

```

Partial Public Class HostWindow
 Inherits Window
 Private selectedItem As Integer
 Private hwndListBox As IntPtr
 Private listControl As ControlHost
 Private app As Application
 Private myWindow As Window
 Private itemCount As Integer

 Private Sub On_UIReady(ByVal sender As Object, ByVal e As EventArgs)
 app = System.Windows.Application.Current
 myWindow = app.MainWindow
 myWindow.SizeToContent = SizeToContent.WidthAndHeight
 listControl = New ControlHost(ControlHostElement.ActualHeight, ControlHostElement.ActualWidth)
 ControlHostElement.Child = listControl
 AddHandler listControl.MessageHook, AddressOf ControlMsgFilter
 hwndListBox = listControl(hwndListBox)
 For i As Integer = 0 To 14 'populate listbox
 Dim itemText As String = "Item" & i.ToString()
 SendMessage(hwndListBox, LB_ADDSTRING, IntPtr.Zero, itemText)
 Next i
 itemCount = SendMessage(hwndListBox, LB_GETCOUNT, IntPtr.Zero, IntPtr.Zero)
 numItems.Text = "" & itemCount.ToString()
 End Sub

```



```

Private Function ControlMsgFilter(ByVal hwnd As IntPtr, ByVal msg As Integer, ByVal wParam As IntPtr, ByVal lParam As IntPtr, ByRef handled As Boolean) As IntPtr
 Dim textLength As Integer

 handled = False
 If msg = WM_COMMAND Then
 Select Case CUInt(wParam.ToInt32()) >> 16 And &HFFFF 'extract the HIWORD
 Case LBN_SELCHANGE 'Get the item text and display it
 selectedItem = SendMessage(listControl.hwndListBox, LB_GETCURSEL, IntPtr.Zero, IntPtr.Zero)
 textLength = SendMessage(listControl.hwndListBox, LB_GETTEXTLEN, IntPtr.Zero, IntPtr.Zero)
 Dim itemText As New StringBuilder()
 SendMessage(hwndListBox, LB_GETTEXT, selectedItem, itemText)
 selectedText.Text = itemText.ToString()
 handled = True
 End Select
 End If
 Return IntPtr.Zero
End Function

Friend Const LBN_SELCHANGE As Integer = &H1, WM_COMMAND As Integer = &H111, LB_GETCURSEL As Integer = &H188,
LB_GETTEXTLEN As Integer = &H18A, LB_ADDSTRING As Integer = &H180, LB_GETTEXT As Integer = &H189,
LB_DELETESTRING As Integer = &H182, LB_GETCOUNT As Integer = &H18B

<DllImport("user32.dll", EntryPoint:="SendMessage", CharSet:=CharSet.Unicode)>
Friend Shared Function SendMessage(ByVal hwnd As IntPtr, ByVal msg As Integer, ByVal wParam As IntPtr, ByVal lParam As IntPtr) As Integer
End Function

<DllImport("user32.dll", EntryPoint:="SendMessage", CharSet:=CharSet.Unicode)>
Friend Shared Function SendMessage(ByVal hwnd As IntPtr, ByVal msg As Integer, ByVal wParam As Integer,
<MarshalAs(UnmanagedType.LPWStr)> ByVal lParam As StringBuilder) As Integer
End Function

<DllImport("user32.dll", EntryPoint:="SendMessage", CharSet:=CharSet.Unicode)>
Friend Shared Function SendMessage(ByVal hwnd As IntPtr, ByVal msg As Integer, ByVal wParam As IntPtr, ByVal lParam As String) As IntPtr
End Function

```

## Implementare la comunicazione tra il controllo e la pagina

Il controllo viene modificato inviando messaggi di Windows. Il controllo comunica quando l'utente interagisce con esso inviando notifiche alla propria finestra host. L'esempio che [ospita un controllo ListBox Win32 in WPF](#) include un'interfaccia utente che fornisce diversi esempi di funzionamento:

- Accodare un elemento all'elenco.
- Eliminare l'elemento selezionato dall'elenco.
- Visualizzare il testo dell'elemento attualmente selezionato.
- Visualizzare il numero di elementi nell'elenco.

L'utente può anche selezionare un elemento nella casella di riepilogo facendo clic su di esso, così come per un'applicazione Win32 convenzionale. I dati visualizzati vengono aggiornati ogni volta che l'utente modifica lo stato della casella di riepilogo, selezionando, aggiungendo o accodando un elemento.

Per accodare elementi, inviare alla casella di riepilogo un [messaggio LB\\_ADDSTRING](#). Per eliminare gli elementi, inviare [LB\\_GETCURSEL](#) per ottenere l'indice della selezione corrente e quindi [LB\\_DELETESTRING](#) per eliminare l'elemento. L'esempio invia anche [LB\\_GETCOUNT](#) e usa il valore restituito per aggiornare la visualizzazione che mostra il numero di elementi. Entrambe le istanze di [SendMessage](#) utilizzano una delle dichiarazioni PInvoke illustrate nella sezione precedente.

```

private void AppendText(object sender, EventArgs args)
{
 if (!string.IsNullOrEmpty(txtAppend.Text))
 {
 SendMessage(hwndListBox, LB_ADDSTRING, IntPtr.Zero, txtAppend.Text);
 }
 itemCount = SendMessage(hwndListBox, LB_GETCOUNT, IntPtr.Zero, IntPtr.Zero);
 numItems.Text = "" + itemCount.ToString();
}
private void DeleteText(object sender, EventArgs args)
{
 selectedItem = SendMessage(listControl(hwndListBox, LB_GETCURSEL, IntPtr.Zero, IntPtr.Zero);
 if (selectedItem != -1) //check for selected item
 {
 SendMessage(hwndListBox, LB_DELETESTRING, (IntPtr)selectedItem, IntPtr.Zero);
 }
 itemCount = SendMessage(hwndListBox, LB_GETCOUNT, IntPtr.Zero, IntPtr.Zero);
 numItems.Text = "" + itemCount.ToString();
}

```

```

Private Sub AppendText(ByVal sender As Object, ByVal args As EventArgs)
 If txtAppend.Text <> String.Empty Then
 SendMessage(hwndListBox, LB_ADDSTRING, IntPtr.Zero, txtAppend.Text)
 End If
 itemCount = SendMessage(hwndListBox, LB_GETCOUNT, IntPtr.Zero, IntPtr.Zero)
 numItems.Text = "" & itemCount.ToString()
End Sub
Private Sub DeleteText(ByVal sender As Object, ByVal args As EventArgs)
 selectedItem = SendMessage(listControl(hwndListBox, LB_GETCURSEL, IntPtr.Zero, IntPtr.Zero)
 If selectedItem <> -1 Then 'check for selected item
 SendMessage(hwndListBox, LB_DELETESTRING, New IntPtr(selectedItem), IntPtr.Zero)
 End If
 itemCount = SendMessage(hwndListBox, LB_GETCOUNT, IntPtr.Zero, IntPtr.Zero)
 numItems.Text = "" & itemCount.ToString()
End Sub

```

Quando l'utente seleziona un elemento o ne modifica la selezione, il controllo notifica alla finestra host inviando un messaggio `WM_COMMAND`, che genera l'evento `MessageHook` per la pagina. Il gestore riceve le stesse informazioni della routine della finestra principale della finestra host. Passa inoltre un riferimento a un valore booleano `handled`. Impostare `handled` su `true` per indicare che il messaggio è stato gestito e che non sono necessarie ulteriori elaborazioni.

`WM_COMMAND` viene inviato per diversi motivi, pertanto è necessario esaminare l'ID notifica per determinare se si tratta di un evento che si desidera gestire. L'ID è contenuto nella parola alta del parametro `wParam`. Nell'esempio vengono utilizzati operatori bit per bit per estrarre l'ID. Se l'utente ha apportato o modificato la selezione, l'ID verrà `LBN_SELCHANGE`.

Quando viene ricevuto `LBN_SELCHANGE`, l'esempio ottiene l'indice dell'elemento selezionato inviando al controllo un messaggio di `LB_GETCURSEL`. Per ottenere il testo, creare prima di tutto un `StringBuilder`. Il controllo viene quindi inviato a un messaggio di `LB_GETTEXT`. Passare l'oggetto `StringBuilder` vuoto come parametro di `wParam`. Quando `SendMessage` restituisce, il `StringBuilder` conterrà il testo dell'elemento selezionato. Questo utilizzo di `SendMessage` richiede ancora un'altra dichiarazione PInvoke.

Infine, impostare `handled` su `true` per indicare che il messaggio è stato gestito.

## Vedere anche

- [HwndHost](#)
- [Interoperatività di WPF e Win32](#)

- Procedura dettagliata: Prima applicazione desktop WPF

# Procedura dettagliata: hosting di contenuto WPF in Win32

12/02/2020 • 29 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) fornisce un ambiente completo per la creazione di applicazioni. Tuttavia, quando si ha un investimento sostanziale nel codice Win32, potrebbe essere più efficace aggiungere WPF funzionalità all'applicazione anziché riscrivere il codice originale. WPF fornisce un meccanismo semplice per l'hosting di contenuto di WPF in una finestra di Win32.

In questa esercitazione viene descritto come scrivere un'applicazione di esempio, [ospitando contenuto WPF in un esempio di finestra Win32](#), che ospita WPF contenuto in una finestra Win32. È possibile estendere questo esempio per ospitare qualsiasi finestra di Win32. Poiché comporta la combinazione di codice gestito e non gestito, l'applicazione viene scritta in C++/cli.

## Requisiti

Questa esercitazione presuppone una conoscenza di base della programmazione WPF e Win32. Per un'introduzione di base alla programmazione WPF, vedere [Introduzione](#). Per un'introduzione alla programmazione Win32, è necessario fare riferimento a uno dei numerosi libri sull'argomento, in particolare *programmare Windows* di Charles Petzold.

Poiché l'esempio che accompagna questa esercitazione è implementato in/CLI C++, in questa esercitazione si presuppone una certa familiarità con C++ l'uso di per programmare l'API Windows, oltre a comprendere la programmazione del codice gestito. La familiarità C++ con/CLI è utile ma non essenziale.

### NOTE

Questa esercitazione include numerosi esempi di codice relativi all'esempio associato. Tuttavia, per una questione di leggibilità, il codice di esempio completo non è compreso. Per il codice di esempio completo, vedere l'esempio relativo all'[hosting di contenuto WPF in una finestra Win32](#).

## Procedura di base

In questa sezione viene illustrata la procedura di base utilizzata per ospitare WPF contenuto in una finestra Win32. Le sezioni rimanenti illustrano in dettaglio i vari passaggi.

La chiave per ospitare WPF contenuto in una finestra Win32 è la classe [HwndSource](#). Questa classe esegue il wrapping del contenuto WPF in una finestra Win32, consentendo di incorporarlo nell'interfaccia utente come finestra figlio. L'approccio seguente combina Win32 e WPF in un'unica applicazione.

1. Implementare il contenuto del WPF come classe gestita.
2. Implementare un'applicazione Windows con C++/cli. Se si inizia con un'applicazione esistente e il C++ codice non gestito, in genere è possibile abilitarlo per chiamare il codice gestito modificando le impostazioni del progetto in modo da includere il flag del compilatore `/clr`.
3. Impostare il modello di threading su apartment a thread singolo (STA).
4. Gestire la notifica [WM\\_CREATE](#) nella procedura della finestra ed eseguire le operazioni seguenti:
  - a. Creare un nuovo oggetto [HwndSource](#) con la finestra padre come parametro `parent`.

- b. Creare un'istanza della classe contenuto WPF.
  - c. Assegnare un riferimento all'oggetto WPF contenuto alla proprietà `RootVisual` dell'`HwndSource`.
  - d. Ottenere HWND per il contenuto. La proprietà `Handle` dell'oggetto `HwndSource` contiene l'handle di finestra (HWND). Per ottenere un HWND utilizzabile nella parte non gestita dell'applicazione, eseguire il cast di `Handle.ToPointer()` a un HWND.
5. Implementare una classe gestita contenente un campo statico con un riferimento al contenuto WPF.  
Questa classe consente di ottenere un riferimento al contenuto del WPF dal codice Win32.
  6. Assegnare il contenuto WPF al campo statico.
  7. Ricevere notifiche dal contenuto del WPF connettendo un gestore a uno o più eventi di WPF.
  8. Comunicare con il contenuto WPF usando il riferimento archiviato nel campo statico per impostare le proprietà ed eseguire altre operazioni.

#### NOTE

È anche possibile usare Extensible Application Markup Language (XAML) per implementare il contenuto del WPF. Tuttavia, sarà necessario compilarlo separatamente come libreria a collegamento dinamico (DLL) e fare riferimento a tale DLL dall'applicazione Win32. La parte restante della procedura è simile a quella appena descritta.

## Implementazione dell'applicazione host

In questa sezione viene descritto come ospitare WPF contenuto in un'applicazione Win32 di base. Il contenuto stesso viene implementato in C++/CLI come classe gestita. In gran parte si tratta di semplice programmazione WPF. Gli aspetti principali dell'implementazione del contenuto vengono illustrati in [implementazione del contenuto WPF](#).

- [Applicazione di base](#)
- [Hosting del contenuto WPF](#)
- [Riferimento al contenuto WPF](#)
- [Comunicazione con il contenuto WPF](#)

### Applicazione di base

Il punto di partenza per l'applicazione host consiste nel creare un modello di Visual Studio 2005.

1. Aprire Visual Studio 2005 e scegliere **nuovo progetto** dal menu **file**.
2. Selezionare **Win32** dall'elenco dei tipi di C++ progetto visivi. Se il linguaggio predefinito non C++ è, questi tipi di progetto sono disponibili in **altri linguaggi**.
3. Selezionare un modello di **progetto Win32**, assegnare un nome al progetto e fare clic su **OK** per avviare la **creazione guidata applicazione Win32**.
4. Accettare le impostazioni predefinite della procedura guidata e fare clic su **fine** per avviare il progetto.

Il modello crea un'applicazione Win32 di base, tra cui:

- Un punto di ingresso per l'applicazione.
- Una finestra, con una procedura di finestra associata (WndProc).
- Menu con intestazioni di **file** e della **Guida**. Il menu **file** contiene un elemento **Exit** che chiude l'applicazione. Il menu? contiene un elemento about che **consente** di avviare una semplice finestra **di**

dialogo.

Prima di iniziare a scrivere il codice per ospitare il contenuto del WPF, è necessario apportare due modifiche al modello di base.

La prima consiste nel compilare il progetto come codice gestito. Per impostazione predefinita, il progetto viene compilato come codice non gestito. Tuttavia, dal momento che WPF è implementato in codice gestito, il progetto deve essere compilato di conseguenza.

1. Fare clic con il pulsante destro del mouse sul nome del progetto in **Esplora soluzioni** e scegliere **Proprietà** dal menu di scelta rapida per aprire la finestra di dialogo **pagine delle proprietà**.
2. Selezionare **proprietà di configurazione** nella visualizzazione albero nel riquadro sinistro.
3. Selezionare supporto **Common Language Runtime** dall'elenco **impostazioni definite progetto** nel riquadro di destra.
4. Selezionare **supporto Common Language Runtime (/CLR)** dall'elenco a discesa.

#### NOTE

Questo flag del compilatore consente di usare codice gestito nell'applicazione, ma il codice non gestito verrà comunque compilato come in precedenza.

WPF usa il modello di threading STA (apartment a thread singolo). Per funzionare correttamente con il codice del contenuto WPF, è necessario impostare il modello di threading dell'applicazione su STA applicando un attributo al punto di ingresso.

```
[System::STAThreadAttribute] //Needs to be an STA thread to play nicely with WPF
int APIENTRY _tWinMain(HINSTANCE hInstance,
 HINSTANCE hPrevInstance,
 LPTSTR lpCmdLine,
 int nCmdShow)
{
```

#### Hosting del contenuto WPF

Il contenuto del WPF è una semplice applicazione di immissione dell'indirizzo. È costituito da diversi controlli **TextBox** nei quali immettere nome utente, indirizzo e così via. Sono inoltre disponibili due **Button** controlli, **OK** e **Annulla**. Quando l'utente fa clic su **OK**, il gestore dell'evento **Click** del pulsante raccoglie i dati dai controlli **TextBox**, li assegna alle proprietà corrispondenti e genera un evento personalizzato, **OnButtonClicked**. Quando l'utente fa clic su **Annulla**, il gestore genera semplicemente **OnButtonClicked**. L'oggetto argomento dell'evento per **OnButtonClicked** contiene un campo booleano che indica il pulsante scelto.

Il codice per ospitare il contenuto del WPF viene implementato in un gestore per la notifica **WM\_CREATE** nella finestra host.

```
case WM_CREATE :
 GetClientRect(hWnd, &rect);
 wpfHwnd = GetHwnd(hWnd, rect.right-375, 0, 375, 250);
 CreateDataDisplay(hWnd, 275, rect.right-375, 375);
 CreateRadioButtons(hWnd);
 break;
```

Il metodo **GetHwnd** acquisisce informazioni sulle dimensioni e sulla posizione più l'handle della finestra padre e restituisce l'handle della finestra del contenuto della WPF ospitata.

#### NOTE

Non è possibile usare una direttiva `#using` per lo spazio dei nomi `System::Windows::Interop`, in quanto si creerebbe un conflitto di nomi tra la struttura `MSG` nello spazio dei nomi e la struttura `MSG` dichiarata in `winuser.h`. Al contrario, occorre usare nomi completi per accedere al contenuto dello spazio dei nomi.

```
HWND GetHwnd(HWND parent, int x, int y, int width, int height)
{
 System::Windows::Interop::HwndSourceParameters^ sourceParams = gcnew
 System::Windows::Interop::HwndSourceParameters(
 "hi" // NAME
);
 sourceParams->PositionX = x;
 sourceParams->PositionY = y;
 sourceParams->Height = height;
 sourceParams->Width = width;
 sourceParams->ParentWindow = IntPtr(parent);
 sourceParams->WindowStyle = WS_VISIBLE | WS_CHILD; // style
 System::Windows::Interop::HwndSource^ source = gcnew System::Windows::Interop::HwndSource(*sourceParams);
 WPFPAGE ^myPage = gcnew WPFPAGE(width, height);
 //Assign a reference to the WPF page and a set of UI properties to a set of static properties in a class
 //that is designed for that purpose.
 WPFPAGEHOST::hostedPage = myPage;
 WPFPAGEHOST::initBackBrush = myPage->Background;
 WPFPAGEHOST::initFontFamily = myPage->DefaultFontFamily;
 WPFPAGEHOST::initFontSize = myPage->DefaultFontSize;
 WPFPAGEHOST::initFontStyle = myPage->DefaultFontStyle;
 WPFPAGEHOST::initFontWeight = myPage->DefaultFontWeight;
 WPFPAGEHOST::initForeBrush = myPage->DefaultForeBrush;
 myPage->OnButtonClicked += gcnew WPFPAGE::ButtonClickedHandler(WPFPAGEHOST::OnButtonClicked);
 source->RootVisual = myPage;
 return (HWND) source->Handle.ToPointer();
}
```

Non è possibile ospitare il contenuto del WPF direttamente nella finestra dell'applicazione. Al contrario, occorre prima creare un oggetto `HwndSource` per eseguire il wrapping del contenuto WPF. Questo oggetto è fondamentalmente una finestra progettata per ospitare un contenuto WPF. Per ospitare l'oggetto `HwndSource` nella finestra padre, è necessario crearlo come elemento figlio di una finestra Win32 che fa parte dell'applicazione. I parametri del costruttore `HwndSource` contengono molto le stesse informazioni che verrebbero passate a `CreateWindow` quando si crea una finestra figlio Win32.

Si crea quindi un'istanza del WPF oggetto contenuto. In questo caso, il contenuto del WPF viene implementato come classe separata, `WPFPAGE`, usando C++/cli. È anche possibile implementare il contenuto WPF mediante XAML. A tale scopo, tuttavia, è necessario configurare un progetto separato e compilare il contenuto del WPF come una DLL. È possibile aggiungere al progetto un riferimento a tale DLL e utilizzare tale riferimento per creare un'istanza del contenuto WPF.

Per visualizzare il contenuto del WPF nella finestra figlio, assegnare un riferimento al contenuto della WPF alla proprietà `RootVisual` della `HwndSource`.

La riga di codice successiva associa un gestore eventi, `WPFPAGEHOST::OnButtonClicked`, all'evento WPF del contenuto `OnButtonClicked`. Questo gestore viene chiamato quando l'utente fa clic sul pulsante **OK** o **Annulla**. Per ulteriori informazioni su questo gestore eventi, vedere [communicating\\_with\\_the\\_WPF\\_content](#).

La riga finale del codice illustrata restituisce l'handle della finestra (HWND) associato all'oggetto `HwndSource`. È possibile usare questo handle dal codice Win32 per inviare messaggi alla finestra ospitata, sebbene l'esempio non lo faccia. L'oggetto `HwndSource` genera un evento ogni volta che riceve un messaggio. Per elaborare i messaggi, chiamare il metodo `AddHook` per associare un gestore di messaggi, quindi elaborare i messaggi in tale gestore.

## Riferimento al contenuto WPF

Per molte applicazioni si vorrà comunicare con il contenuto WPF in un secondo momento. Ad esempio, sarà possibile modificare le proprietà del contenuto WPF o fare in modo che l'oggetto `HwndSource` ospiti un contenuto WPF diverso. A tale scopo, è necessario un riferimento all'oggetto `HwndSource` o al contenuto WPF. L'oggetto `HwndSource` e il contenuto WPF associato restano in memoria fino all'eliminazione dell'handle della finestra. Tuttavia, la variabile assegnata all'oggetto `HwndSource` uscirà dall'ambito non appena si esce dalla procedura di finestra. Il modo personalizzato per gestire questo problema con le applicazioni Win32 consiste nell'usare una variabile statica o globale. Sfortunatamente, non è possibile assegnare un oggetto gestito ai suddetti tipi di variabili. È possibile assegnare l'handle della finestra associato all'oggetto `HwndSource` a una variabile globale o statica, ma ciò non consentirà di accedere all'oggetto in questione.

La soluzione più semplice a questo problema consiste nell'implementare una classe gestita contenente un insieme di campi statici che a loro volta contengono i riferimenti agli oggetti gestiti ai quali si vuole accedere. Nell'esempio viene usata la classe `WPFPageHost`, che contiene un riferimento al contenuto WPF oltre ai valori iniziali di diverse proprietà, modificabili in un secondo momento dall'utente. La classe viene definita nell'intestazione.

```
public ref class WPFPageHost
{
public:
 WPFPageHost();
 static WPFPage^ hostedPage;
 //initial property settings
 static System::Windows::Media::Brush^ initBackBrush;
 static System::Windows::Media::Brush^ initForeBrush;
 static System::Windows::Media::FontFamily^ initFontFamily;
 static System::Windows::FontStyle initFontSize;
 static System::Windows::FontWeight initFontWeight;
 static double initFontSize;
};
```

La seconda parte della funzione `GetHwnd` assegna valori ai suddetti campi che verranno usati in un secondo momento, con `myPage` ancora nell'ambito.

## Comunicazione con il contenuto WPF

Esistono due tipi di comunicazione con il contenuto WPF. L'applicazione riceve informazioni dal contenuto del WPF quando l'utente fa clic sui pulsanti **OK** o **Annulla**. L'applicazione dispone anche di un'Interfaccia utente che consente di modificare diverse proprietà del contenuto WPF, ad esempio il colore di sfondo o le dimensioni predefinite del carattere.

Come indicato in precedenza, quando l'utente fa clic su uno dei pulsanti il contenuto WPF genera un evento `OnButtonClicked`. L'applicazione associa un gestore a questo evento per ricevere le notifiche. Se è stato fatto clic sul pulsante **OK**, il gestore ottiene le informazioni sull'utente dal contenuto del WPF e le Visualizza in un set di controlli statici.

```

void WPFPButtonClicked(Object ^sender, MyPageEventArgs ^args)
{
 if(args->IsOK) //display data if OK button was clicked
 {
 WPFPPage ^myPage = WPFPPageHost::hostedPage;
 LPCWSTR userName = (LPCWSTR) InteropServices::Marshal::StringToHGlobalAuto("Name: " + myPage-
>EnteredName).ToPointer();
 SetWindowText(nameLabel, userName);
 LPCWSTR userAddress = (LPCWSTR) InteropServices::Marshal::StringToHGlobalAuto("Address: " + myPage-
>EnteredAddress).ToPointer();
 SetWindowText(addressLabel, userAddress);
 LPCWSTR userCity = (LPCWSTR) InteropServices::Marshal::StringToHGlobalAuto("City: " + myPage-
>EnteredCity).ToPointer();
 SetWindowText(cityLabel, userCity);
 LPCWSTR userState = (LPCWSTR) InteropServices::Marshal::StringToHGlobalAuto("State: " + myPage-
>EnteredState).ToPointer();
 SetWindowText(stateLabel, userState);
 LPCWSTR userZip = (LPCWSTR) InteropServices::Marshal::StringToHGlobalAuto("Zip: " + myPage-
>EnteredZip).ToPointer();
 SetWindowText(zipLabel, userZip);
 }
 else
 {
 SetWindowText(nameLabel, L"Name: ");
 SetWindowText(addressLabel, L"Address: ");
 SetWindowText(cityLabel, L"City: ");
 SetWindowText(stateLabel, L"State: ");
 SetWindowText(zipLabel, L"Zip: ");
 }
}

```

Il gestore riceve un oggetto argomento dell'evento personalizzato dal contenuto WPF, `MyPageEventArgs`. La proprietà `IsOK` dell'oggetto è impostata su `true` se è stato fatto clic sul pulsante **OK** e `false` se è stato fatto clic sul pulsante **Annulla**.

Se è stato fatto clic sul pulsante **OK**, il gestore ottiene un riferimento al contenuto del WPF dalla classe contenitore. Raccoglie quindi le informazioni utente contenute nelle proprietà del contenuto WPF associate e sa i controlli statici per visualizzare le informazioni nella finestra padre. Poiché i dati del contenuto WPF sono sotto forma di stringa gestita, è necessario eseguirne il marshalling per l'uso da parte di un controllo Win32. Se è stato fatto clic sul pulsante **Annulla**, il gestore cancella i dati dai controlli statici.

L'Interfaccia utente dell'applicazione fornisce un set di pulsanti di opzione che consentono di modificare il colore di sfondo del contenuto WPF e diverse proprietà correlate al tipo di carattere. Nell'esempio seguente viene illustrato un estratto della procedura di finestra (WndProc) dell'applicazione, nonché la gestione dei messaggi tramite la quale è possibile impostare varie proprietà in messaggi diversi, incluso il colore di sfondo. La parte restante è simile, pertanto non viene illustrata. Vedere l'esempio completo per i dettagli e il contesto.

```

case WM_COMMAND:
 wMid = LOWORD(wParam);
 wParamEvent = HIWORD(wParam);

 switch (wMid)
 {
 //Menu selections
 case IDM_ABOUT:
 DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
 break;
 case IDM_EXIT:
 DestroyWindow(hWnd);
 break;
 //RadioButtons
 case IDC_ORIGINALBACKGROUND :
 WPFPAGEHOST::hostedPage->Background = WPFPAGEHOST::initBackBrush;
 break;
 case IDC_LIGHTGREENBACKGROUND :
 WPFPAGEHOST::hostedPage->Background = gcnew SolidColorBrush(Colors::LightGreen);
 break;
 case IDC_LIGHTSALMONBACKGROUND :
 WPFPAGEHOST::hostedPage->Background = gcnew SolidColorBrush(Colors::LightSalmon);
 break;
 }
}

```

Per impostare il colore di sfondo, ottenere un riferimento al contenuto WPF (`hostedPage`) da `WPFPAGEHOST` e impostare la proprietà del colore di sfondo sul colore appropriato. Nell'esempio vengono usate tre opzioni di colore: il colore originale, verde chiaro o salmone chiaro. Il colore di sfondo originale viene archiviato come campo statico nella classe `WPFPAGEHOST`. Per impostare gli altri due colori, creare un nuovo oggetto `SolidColorBrush` e passare al costruttore un valore di colore statico dall'oggetto `Colors`.

## Implementazione della pagina WPF

È possibile ospitare e usare il contenuto del WPF senza alcuna conoscenza dell'implementazione effettiva. Se il contenuto del WPF è stato incluso in una DLL separata, potrebbe essere stato compilato in qualsiasi linguaggio Common Language Runtime (CLR). Di seguito viene riportata una breve procedura C++ dettagliata dell'implementazione di/CLI utilizzata nell'esempio. Questa sezione contiene le sottosezioni seguenti.

- [Layout](#)
- [Restituzione dei dati alla finestra host](#)
- [Impostazione delle proprietà WPF](#)

### Layout

Gli elementi Interfaccia utente nel contenuto WPF sono costituiti da cinque controlli `TextBox`, con i controlli di `Label` associati: Name, Address, City, state e zip. Sono disponibili anche due controlli `Button`, **OK** e **Annulla**.

Il contenuto WPF è implementato nella classe `WPFPAGE`. Il layout viene gestito mediante un apposito elemento `Grid`. La classe eredita da `Grid`, il che la rende effettivamente l'elemento radice del contenuto WPF.

Il costruttore del contenuto WPF accetta la larghezza e l'altezza richieste e ridimensiona il `Grid` di conseguenza. Viene quindi definito il layout di base creando un set di `ColumnDefinition` e `RowDefinition` oggetti e aggiungendoli rispettivamente al `Grid.ColumnDefinitions` di base degli oggetti e alle raccolte di `RowDefinitions`. Viene così definita una griglia di cinque righe e sette colonne, le cui dimensioni sono determinate dal contenuto delle celle.

```

WPFFPage::WPFFPage(int allottedWidth, int allottedHeight)
{
 array<ColumnDefinition ^> ^ columnDef = gcnew array<ColumnDefinition ^> (4);
 array<RowDefinition ^> ^ rowDef = gcnew array<RowDefinition ^> (6);

 this->Height = allottedHeight;
 this->Width = allottedWidth;
 this->Background = gcnew SolidColorBrush(Colors::LightGray);

 //Set up the Grid's row and column definitions
 for(int i=0; i<4; i++)
 {
 columnDef[i] = gcnew ColumnDefinition();
 columnDef[i]->Width = GridLength(1, GridUnitType::Auto);
 this->ColumnDefinitions->Add(columnDef[i]);
 }
 for(int i=0; i<6; i++)
 {
 rowDef[i] = gcnew RowDefinition();
 rowDef[i]->Height = GridLength(1, GridUnitType::Auto);
 this->RowDefinitions->Add(rowDef[i]);
 }
}

```

Successivamente il costruttore aggiunge gli elementi dell'Interfaccia utente a [Grid](#). Il primo elemento è il testo del titolo, ovvero un controllo [Label](#) centrato nella prima riga della griglia.

```

//Add the title
titleText = gcnew Label();
titleText->Content = "Simple WPF Control";
titleText->HorizontalAlignment = System::Windows::HorizontalAlignment::Center;
titleText->Margin = Thickness(10, 5, 10, 0);
titleText->FontWeight = FontWeights::Bold;
titleText->FontSize = 14;
Grid::SetColumn(titleText, 0);
Grid::SetRow(titleText, 0);
Grid::SetColumnSpan(titleText, 4);
this->Children->Add(titleText);

```

La riga successiva contiene il controllo [Label](#) Name e il controllo [TextBox](#) associato. Dal momento che per ogni coppia etichetta/casella di testo viene usato lo stesso codice, questo viene collocato in una coppia di metodi privati e usato per tutte e cinque le coppie etichetta/casella di testo. I metodi creano il controllo appropriato e chiamano i metodi [Grid](#) e [SetColumn](#) statici della classe [SetRow](#) per posizionare i controlli nella cella appropriata. Una volta creato il controllo, nell'esempio viene chiamato il metodo [Add](#) sulla proprietà [Children](#) di [Grid](#) per aggiungere il controllo alla griglia. Il codice usato per aggiungere le coppie etichetta/casella di testo restanti è simile. Vedere il codice di esempio per i dettagli.

```

//Add the Name Label and TextBox
nameLabel = CreateLabel(0, 1, "Name");
this->Children->Add(nameLabel);
nameTextBox = CreateTextBox(1, 1, 3);
this->Children->Add(nameTextBox);

```

D seguito viene riportata l'implementazione dei due metodi:

```

Label ^WPFPPage::CreateLabel(int column, int row, String ^ text)
{
 Label ^ newLabel = gcnew Label();
 newLabel->Content = text;
 newLabel->Margin = Thickness(10, 5, 10, 0);
 newLabel->FontWeight = FontWeights::Normal;
 newLabel->FontSize = 12;
 Grid::SetColumn(newLabel, column);
 Grid::SetRow(newLabel, row);
 return newLabel;
}
TextBox ^WPFPPage::CreateTextBox(int column, int row, int span)
{
 TextBox ^newTextBox = gcnew TextBox();
 newTextBox->Margin = Thickness(10, 5, 10, 0);
 Grid::SetColumn(newTextBox, column);
 Grid::SetRow(newTextBox, row);
 Grid::SetColumnSpan(newTextBox, span);
 return newTextBox;
}

```

Infine, l'esempio aggiunge i pulsanti **OK** e **Annulla** e connette un gestore eventi ai relativi eventi [Click](#).

```

//Add the Buttons and attach event handlers
okButton = CreateButton(0, 5, "OK");
cancelButton = CreateButton(1, 5, "Cancel");
this->Children->Add(okButton);
this->Children->Add(cancelButton);
okButton->Click += gcnew RoutedEventHandler(this, &WPFPPage::ButtonClicked);
cancelButton->Click += gcnew RoutedEventHandler(this, &WPFPPage::ButtonClicked);

```

### Restituzione dei dati alla finestra host

Facendo clic su un pulsante, viene generato il rispettivo evento [Click](#). La finestra host potrebbe semplicemente associare i gestori a questi eventi e ottenere i dati direttamente dai controlli [TextBox](#). Nell'esempio viene usato un approccio meno diretto. Gestisce la [Click](#) all'interno del contenuto WPF e quindi genera un [OnButtonClicked](#) di eventi personalizzato per notificare il contenuto WPF. Ciò consente al contenuto del WPF di eseguire una convalida dei parametri prima di notificare l'host. Il gestore ottiene il testo dai controlli [TextBox](#) e lo assegna a proprietà pubbliche, dalle quali l'host può recuperare le informazioni.

Dichiarazione di evento in [WPFPPage.h](#):

```

public:
 delegate void ButtonClickHandler(Object ^, MyPageEventArgs ^);
 WPFPPage();
 WPFPPage(int height, int width);
 event ButtonClickHandler ^OnButtonClicked;

```

Gestore eventi [Click](#) in [WPFPPage.cpp](#):

```

void WPFPAGE::ButtonClicked(Object ^sender, RoutedEventArgs ^args)
{
 //TODO: validate input data
 bool okClicked = true;
 if(sender == cancelButton)
 okClicked = false;
 EnteredName = nameTextBox->Text;
 EnteredAddress = addressTextBox->Text;
 EnteredCity = cityTextBox->Text;
 EnteredState = stateTextBox->Text;
 EnteredZip = zipTextBox->Text;
 OnButtonClicked(this, gcnew MyPageEventArgs(okClicked));
}

```

## Impostazione delle proprietà WPF

L'host Win32 consente all'utente di modificare diverse proprietà del contenuto WPF. Dal lato Win32, è sufficiente modificare le proprietà. L'implementazione nella classe del contenuto WPF è leggermente più complessa, poiché non esiste un'unica proprietà globale che controlla i tipi di carattere per tutti i controlli. La proprietà adatta per ogni controllo viene modificata nelle funzioni di accesso dell'insieme di proprietà. Nell'esempio seguente viene illustrato il codice per la proprietà `DefaultFontFamily`. Impostando la proprietà, viene chiamato un metodo privato che a sua volta imposta le proprietà `FontFamily` per i vari controlli.

Da WPFPAGE.h:

```

property FontFamily^ DefaultFontFamily
{
 FontFamily^ get() {return _defaultFontFamily;}
 void set(FontFamily^ value) {SetFontFamily(value);}
};

```

Da WPFPAGE.cpp:

```

void WPFPAGE::SetFontFamily(FontFamily^ newFontFamily)
{
 _defaultFontFamily = newFontFamily;
 titleText->FontFamily = newFontFamily;
 nameLabel->FontFamily = newFontFamily;
 addressLabel->FontFamily = newFontFamily;
 cityLabel->FontFamily = newFontFamily;
 stateLabel->FontFamily = newFontFamily;
 zipLabel->FontFamily = newFontFamily;
}

```

## Vedere anche

- [HwndSource](#)
- [Interoperatività di WPF e Win32](#)

# Procedura dettagliata: ospitare un Clock WPF in Win32

12/02/2020 • 14 minutes to read • [Edit Online](#)

Per inserire WPF all'interno delle applicazioni Win32, utilizzare [HwndSource](#), che fornisce l'HWND che contiene il contenuto di WPF. Innanzitutto si crea il [HwndSource](#), fornendo parametri simili a CreateWindow. Si dirà quindi al [HwndSource](#) sul contenuto del WPF che si vuole al suo interno. Infine, si ottiene il HWND dal [HwndSource](#). In questa procedura dettagliata viene illustrato come creare una WPF mista all'interno di un'applicazione Win32 che implementa nuovamente la finestra di dialogo **Proprietà data e ora** del sistema operativo.

## Prerequisites

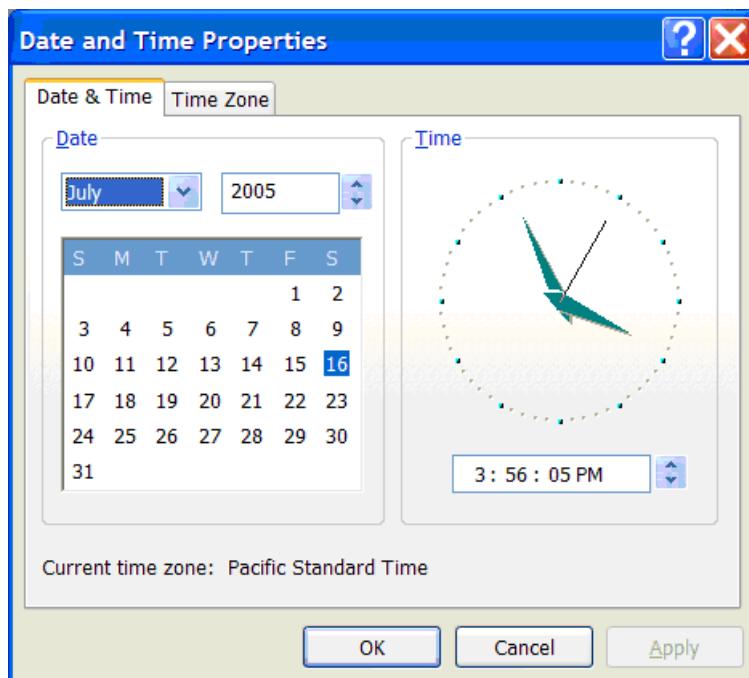
Vedere l'[interoperatività di WPF e Win32](#).

## Come usare questa esercitazione

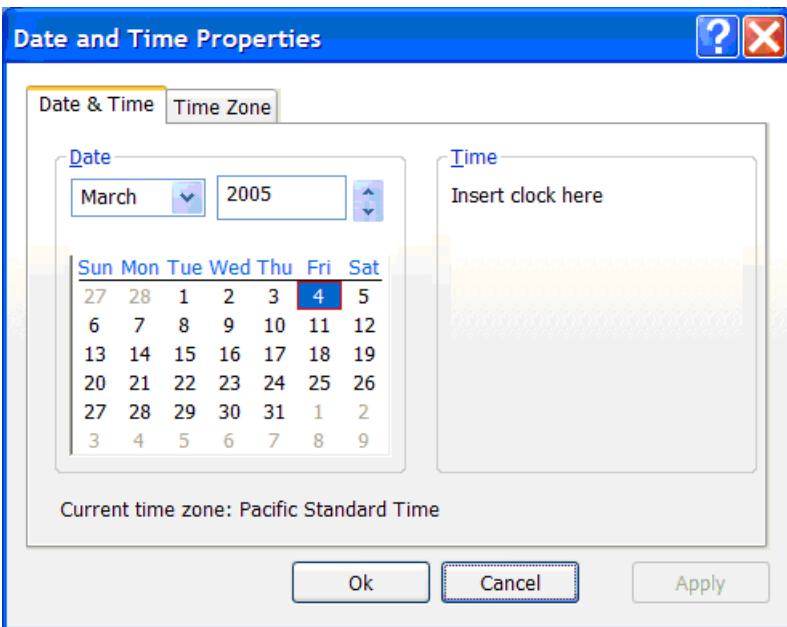
Questa esercitazione si concentra sui passaggi importanti per la produzione di un'applicazione di interoperatività. L'esercitazione è supportata da un esempio di [interoperatività con clock Win32](#), ma tale esempio riflette il prodotto finale. Questa esercitazione illustra i passaggi come se si iniziasse con un progetto Win32 esistente, ad esempio un progetto preesistente e si stesse aggiungendo una WPF ospitata all'applicazione. È possibile confrontare il prodotto finale con l'[esempio di interoperatività con clock Win32](#).

## Procedura dettagliata di Windows Presentation Framework in Win32 (HwndSource)

Il grafico seguente illustra il prodotto finale previsto di questa esercitazione:



Per ricreare questa finestra di dialogo è possibile C++ creare un progetto Win32 in Visual Studio e utilizzare l'editor finestre per creare gli elementi seguenti:



Non è necessario usare Visual Studio per usare [HwndSource](#) non è necessario usare C++ per scrivere programmi Win32, ma si tratta di un modo piuttosto comune per eseguire questa operazione e si presta bene a una spiegazione graduale dell'esercitazione.

Per inserire un WPF Clock nella finestra di dialogo, è necessario eseguire cinque passaggi specifici:

1. Abilitare il progetto Win32 per chiamare il codice gestito (**/CLR**) modificando le impostazioni del progetto in Visual Studio.
2. Creare una [WPF Page](#) in una DLL separata.
3. Inserire il [WPF Page](#) all'interno di un [HwndSource](#).
4. Ottenere un HWND per tale [Page](#) utilizzando la proprietà [Handle](#).
5. Utilizzare Win32 per decidere dove posizionare il HWND all'interno dell'applicazione Win32 di dimensioni maggiori

## /clr

Il primo passaggio consiste nel trasformare questo progetto Win32 non gestito in uno in grado di chiamare codice gestito. Si usa l'opzione del compilatore/CLR, che consente di collegare le DLL necessarie da usare e di modificare il metodo Main per l'uso con WPF.

Per abilitare l'uso del codice gestito all'interno C++ del progetto: fare clic con il pulsante destro del mouse sul progetto win32clock e scegliere **Proprietà**. Nella pagina delle proprietà **generale** (impostazione predefinita) modificare il supporto Common Language Runtime in **/clr**.

Aggiungere quindi i riferimenti alle DLL necessarie per WPF: PresentationCore.dll, PresentationFramework.dll, System.dll, WindowsBase.dll, UIAutomationProvider.dll e UIAutomationTypes.dll. (le istruzioni seguenti presuppongono che il sistema operativo sia installato nell'unità C:).

1. Fare clic con il pulsante destro del mouse su progetto win32clock e selezionare **riferimenti...** e all'interno di tale finestra di dialogo:
2. Fare clic con il pulsante destro del mouse su progetto win32clock e selezionare **riferimenti...**.
3. Fare clic su **Aggiungi nuovo riferimento**, fare clic sulla scheda Sfoglia, immettere C:\program files\reference assemblies\microsoft\framework\v3.0\PresentationCore.dll e fare clic su OK.
4. Ripetere l'operazione per PresentationFramework.dll: C:\Programmi\Reference

Assemblies\Microsoft\Framework\v3.0\PresentationFramework.dll.

5. Ripetere l'operazione per WindowsBase.dll: C:\Programmi\Reference Assemblies\Microsoft\Framework\v3.0\WindowsBase.dll.
6. Ripetere l'operazione per UIAutomationTypes.dll: C:\Programmi\Reference Assemblies\Microsoft\Framework\v3.0\UIAutomationTypes.dll.
7. Ripetere l'operazione per UIAutomationProvider.dll: C:\Programmi\Reference Assemblies\Microsoft\Framework\v3.0\UIAutomationProvider.dll.
8. Fare clic su **Aggiungi nuovo riferimento**, selezionare System. dll e fare clic su **OK**.
9. Fare clic su **OK** per chiudere le pagine delle proprietà di win32clock per l'aggiunta di riferimenti.

Infine, aggiungere il `STAThreadAttribute` al metodo `_tWinMain` per l'utilizzo con WPF:

```
[System::STAThreadAttribute]
int APIENTRY _tWinMain(HINSTANCE hInstance,
 HINSTANCE hPrevInstance,
 LPTSTR lpCmdLine,
 int nCmdShow)
```

Questo attributo indica al Common Language Runtime (CLR) che, quando inizializza Component Object Model (COM), deve usare un modello di Apartment a thread singolo (STA), che è necessario per WPF (e Windows Forms).

## Creare una pagina di Windows Presentation Framework

Successivamente, si crea una DLL che definisce una [Pagedi WPF](#). Spesso è più semplice creare il [WPFPage](#) come applicazione autonoma e scrivere ed eseguire il debug della parte WPF in questo modo. Al termine, il progetto può essere trasformato in una DLL facendo clic con il pulsante destro del mouse sul progetto, facendo clic su **Proprietà**, passando all'applicazione e modificando il tipo di output in libreria di classi di Windows.

Il progetto di WPF dll può quindi essere combinato con il progetto Win32 (una soluzione che contiene due progetti): fare clic con il pulsante destro del mouse sulla soluzione e selezionare **progetto Add\Existing**.

Per utilizzare tale DLL WPF dal progetto Win32, è necessario aggiungere un riferimento:

1. Fare clic con il pulsante destro del mouse su progetto win32clock e selezionare **riferimenti...** .
2. Fare clic su **Aggiungi nuovo riferimento**.
3. Fare clic sulla scheda **progetti** . Selezionare WPFClock, quindi fare clic su OK.
4. Fare clic su **OK** per chiudere le pagine delle proprietà di win32clock per l'aggiunta di riferimenti.

## HwndSource

Usare quindi [HwndSource](#) per rendere il [WPFPage](#) simile a un HWND. Aggiungere questo blocco di codice a un file C++:

```

namespace ManagedCode
{
 using namespace System;
 using namespace System::Windows;
 using namespace System::Windows::Interop;
 using namespace System::Windows::Media;

 HWND GetHwnd(HWND parent, int x, int y, int width, int height) {
 HwndSource^ source = gcnew HwndSource(
 0, // class style
 WS_VISIBLE | WS_CHILD, // style
 0, // exstyle
 x, y, width, height,
 "hi", // NAME
 IntPtr(parent) // parent window
);

 UIElement^ page = gcnew WPFClock::Clock();
 source->RootVisual = page;
 return (HWND) source->Handle.ToPointer();
 }
}
}

```

Poiché si tratta di una parte estesa di codice possono essere necessarie alcune spiegazioni. La prima parte contiene varie clausole che evitano di specificare il nome completo di tutte le chiamate:

```

namespace ManagedCode
{
 using namespace System;
 using namespace System::Windows;
 using namespace System::Windows::Interop;
 using namespace System::Windows::Media;

```

Si definisce quindi una funzione che crea il contenuto del WPF, ne inserisce una [HwndSource](#) e restituisce HWND:

```

HWND GetHwnd(HWND parent, int x, int y, int width, int height) {

```

Prima di tutto si crea un [HwndSource](#), i cui parametri sono simili a CreateWindow:

```

HwndSource^ source = gcnew HwndSource(
 0, // class style
 WS_VISIBLE | WS_CHILD, // style
 0, // exstyle
 x, y, width, height,
 "hi", // NAME
 IntPtr(parent) // parent window
);

```

Si crea quindi la classe di contenuto WPF chiamando il relativo costruttore:

```

UIElement^ page = gcnew WPFClock::Clock();

```

Si connette quindi la pagina alla [HwndSource](#):

```

source->RootVisual = page;

```

Nella riga finale, restituire HWND per la `HwndSource`:

```
return (HWND) source->Handle.ToPointer();
```

## Posizionamento dell'oggetto HWND

Ora che è presente un HWND che contiene il WPF clock, è necessario inserire tale HWND nella finestra di dialogo Win32. Se si conoscesse solo la posizione in cui inserire HWND, è sufficiente passare le dimensioni e la posizione alla funzione `GetHwnd` definita in precedenza. Tuttavia, si è usato un file di risorse per definire la finestra di dialogo perciò non si conosce esattamente la posizione degli oggetti HWND. È possibile usare l'editor finestre di Visual Studio per inserire un controllo statico Win32 in cui si vuole che l'orologio venga usato ("Inserisci orologio") e lo si usi per posizionare il WPF Clock.

Quando si gestiscono WM\_INITDIALOG, si utilizza `GetDlgItem` per recuperare HWND per il segnaposto STATIC:

```
HWND placeholder = GetDlgItem(hDlg, IDC_CLOCK);
```

Quindi si calcolano le dimensioni e la posizione del segnaposto statico, in modo che sia possibile inserire il WPF clock in tale posizione:

Rettangolo RECT;

```
GetWindowRect	placeholder, &rectangle;
int width = rectangle.right - rectangle.left;
int height = rectangle.bottom - rectangle.top;
POINT point;
point.x = rectangle.left;
point.y = rectangle.top;
result = MapWindowPoints(NULL, hDlg, &point, 1);
```

Dopodiché si nasconde il segnaposto STATIC:

```
ShowWindow	placeholder, SW_HIDE);
```

E creare il WPF HWND clock in tale percorso:

```
HWND clock = ManagedCode::GetHwnd(hDlg, point.x, point.y, width, height);
```

Per rendere interessante l'esercitazione e per produrre un vero e proprio WPF clock, è necessario creare un controllo del clock WPF a questo punto. È possibile eseguire questa operazione principalmente nel markup, con solo alcuni gestori eventi in code-behind. Poiché questa esercitazione riguarda l'interoperatività e non la progettazione dei controlli, il codice completo per il WPF clock viene fornito qui come blocco di codice, senza istruzioni discrete per la compilazione o il significato di ogni parte. Modificare questo codice per variare l'aspetto o le funzionalità del controllo.

Questo è il markup:

```
<Page x:Class="WPFClock.Clock"
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 >
 <Grid>
 <Grid.Background>
 <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
```

```

 <GradientStop Color="#fcfcfe" Offset="0" />
 <GradientStop Color="#f6f4f0" Offset="1.0" />
 </LinearGradientBrush>
</Grid.Background>

<Grid Name="PodClock" HorizontalAlignment="Center" VerticalAlignment="Center">
 <Grid.Resources>
 <Storyboard x:Key="sb">
 <DoubleAnimation From="0" To="360" Duration="12:00:00" RepeatBehavior="Forever"
 Storyboard.TargetName="HourHand"
 Storyboard.TargetProperty="(Rectangle.RenderTransform).(RotateTransform.Angle)"
 />
 <DoubleAnimation From="0" To="360" Duration="01:00:00" RepeatBehavior="Forever"
 Storyboard.TargetName="MinuteHand"
 Storyboard.TargetProperty="(Rectangle.RenderTransform).(RotateTransform.Angle)"
 />
 <DoubleAnimation From="0" To="360" Duration="0:1:00" RepeatBehavior="Forever"
 Storyboard.TargetName="SecondHand"
 Storyboard.TargetProperty="(Rectangle.RenderTransform).(RotateTransform.Angle)"
 />
 </Storyboard>
 </Grid.Resources>

 <Ellipse Width="108" Height="108" StrokeThickness="3">
 <Ellipse.Stroke>
 <LinearGradientBrush>
 <GradientStop Color="LightBlue" Offset="0" />
 <GradientStop Color="DarkBlue" Offset="1" />
 </LinearGradientBrush>
 </Ellipse.Stroke>
 </Ellipse>
 <Ellipse VerticalAlignment="Center" HorizontalAlignment="Center" Width="104" Height="104"
 Fill="LightBlue" StrokeThickness="3">
 <Ellipse.Stroke>
 <LinearGradientBrush>
 <GradientStop Color="DarkBlue" Offset="0" />
 <GradientStop Color="LightBlue" Offset="1" />
 </LinearGradientBrush>
 </Ellipse.Stroke>
 </Ellipse>
 <Border BorderThickness="1" BorderBrush="Black" Background="White" Margin="20"
 HorizontalAlignment="Right" VerticalAlignment="Center">
 <TextBlock Name="MonthDay" Text="{Binding}"/>
 </Border>
 <Canvas Width="102" Height="102">
 <Ellipse Width="8" Height="8" Fill="Black" Canvas.Top="46" Canvas.Left="46" />
 <Rectangle Canvas.Top="5" Canvas.Left="48" Fill="Black" Width="4" Height="8">
 <Rectangle.RenderTransform>
 <RotateTransform CenterX="2" CenterY="46" Angle="0" />
 </Rectangle.RenderTransform>
 </Rectangle>
 <Rectangle Canvas.Top="5" Canvas.Left="49" Fill="Black" Width="2" Height="6">
 <Rectangle.RenderTransform>
 <RotateTransform CenterX="2" CenterY="46" Angle="30" />
 </Rectangle.RenderTransform>
 </Rectangle>
 <Rectangle Canvas.Top="5" Canvas.Left="49" Fill="Black" Width="2" Height="6">
 <Rectangle.RenderTransform>
 <RotateTransform CenterX="2" CenterY="46" Angle="60" />
 </Rectangle.RenderTransform>
 </Rectangle>
 <Rectangle Canvas.Top="5" Canvas.Left="48" Fill="Black" Width="4" Height="8">
 <Rectangle.RenderTransform>
 <RotateTransform CenterX="2" CenterY="46" Angle="90" />
 </Rectangle.RenderTransform>
 </Rectangle>
 <Rectangle Canvas.Top="5" Canvas.Left="49" Fill="Black" Width="2" Height="6">
 <Rectangle.RenderTransform>
 <RotateTransform CenterX="2" CenterY="46" Angle="120" />
 </Rectangle.RenderTransform>
 </Rectangle>
 </Canvas>

```

```

 </Rectangle.RenderTransform>
 </Rectangle>
 <Rectangle Canvas.Top="5" Canvas.Left="49" Fill="Black" Width="2" Height="6">
 <Rectangle.RenderTransform>
 <RotateTransform CenterX="2" CenterY="46" Angle="150" />
 </Rectangle.RenderTransform>
 </Rectangle>
 <Rectangle Canvas.Top="5" Canvas.Left="48" Fill="Black" Width="4" Height="8">
 <Rectangle.RenderTransform>
 <RotateTransform CenterX="2" CenterY="46" Angle="180" />
 </Rectangle.RenderTransform>
 </Rectangle>
 <Rectangle Canvas.Top="5" Canvas.Left="49" Fill="Black" Width="2" Height="6">
 <Rectangle.RenderTransform>
 <RotateTransform CenterX="2" CenterY="46" Angle="210" />
 </Rectangle.RenderTransform>
 </Rectangle>
 <Rectangle Canvas.Top="5" Canvas.Left="49" Fill="Black" Width="2" Height="6">
 <Rectangle.RenderTransform>
 <RotateTransform CenterX="2" CenterY="46" Angle="240" />
 </Rectangle.RenderTransform>
 </Rectangle>
 <Rectangle Canvas.Top="5" Canvas.Left="48" Fill="Black" Width="4" Height="8">
 <Rectangle.RenderTransform>
 <RotateTransform CenterX="2" CenterY="46" Angle="270" />
 </Rectangle.RenderTransform>
 </Rectangle>
 <Rectangle Canvas.Top="5" Canvas.Left="49" Fill="Black" Width="2" Height="6">
 <Rectangle.RenderTransform>
 <RotateTransform CenterX="2" CenterY="46" Angle="300" />
 </Rectangle.RenderTransform>
 </Rectangle>
 <Rectangle Canvas.Top="5" Canvas.Left="49" Fill="Black" Width="2" Height="6">
 <Rectangle.RenderTransform>
 <RotateTransform CenterX="2" CenterY="46" Angle="330" />
 </Rectangle.RenderTransform>
 </Rectangle>

 <Rectangle x:Name="HourHand" Canvas.Top="21" Canvas.Left="48"
 Fill="Black" Width="4" Height="30">
 <Rectangle.RenderTransform>
 <RotateTransform x:Name="HourHand2" CenterX="2" CenterY="30" />
 </Rectangle.RenderTransform>
 </Rectangle>
 <Rectangle x:Name="MinuteHand" Canvas.Top="6" Canvas.Left="49"
 Fill="Black" Width="2" Height="45">
 <Rectangle.RenderTransform>
 <RotateTransform CenterX="1" CenterY="45" />
 </Rectangle.RenderTransform>
 </Rectangle>
 <Rectangle x:Name="SecondHand" Canvas.Top="4" Canvas.Left="49"
 Fill="Red" Width="1" Height="47">
 <Rectangle.RenderTransform>
 <RotateTransform CenterX="0.5" CenterY="47" />
 </Rectangle.RenderTransform>
 </Rectangle>
</Canvas>
</Grid>
</Grid>
</Page>

```

Questo è il code-behind associato:

```

using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using System.Windows.Threading;

namespace WPFClock
{
 /// <summary>
 /// Interaction logic for Clock.xaml
 /// </summary>
 public partial class Clock : Page
 {
 private DispatcherTimer _dayTimer;

 public Clock()
 {
 InitializeComponent();
 this.Loaded += new RoutedEventHandler(Clock_Loaded);
 }

 void Clock_Loaded(object sender, RoutedEventArgs e) {
 // set the datacontext to be today's date
 DateTime now = DateTime.Now;
 DataContext = now.Day.ToString();

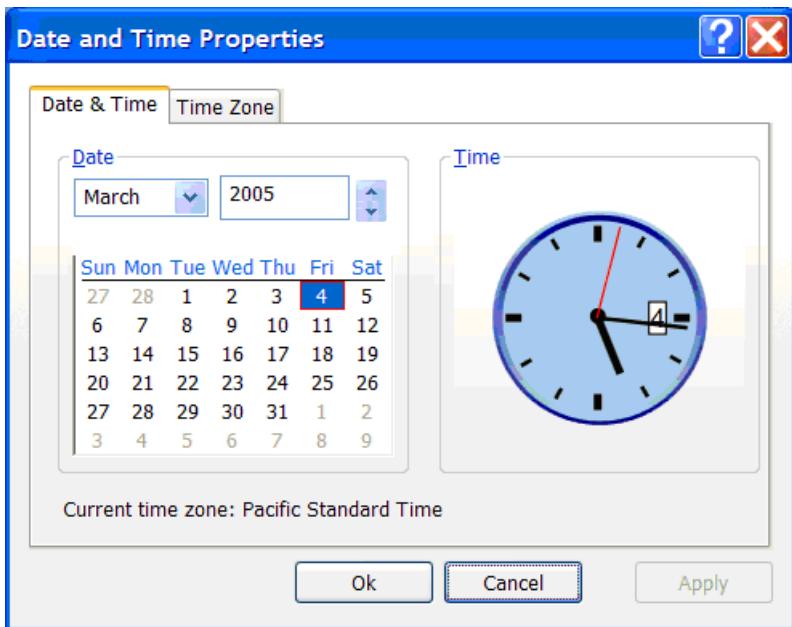
 // then set up a timer to fire at the start of tomorrow, so that we can update
 // the datacontext
 _dayTimer = new DispatcherTimer();
 _dayTimer.Interval = new TimeSpan(1, 0, 0, 0) - now.TimeOfDay;
 _dayTimer.Tick += new EventHandler(OnDayChange);
 _dayTimer.Start();

 // finally, seek the timeline, which assumes a beginning at midnight, to the appropriate
 // offset
 Storyboard sb = (Storyboard)PodClock.FindResource("sb");
 sb.Begin(PodClock, HandoffBehavior.SnapshotAndReplace, true);
 sb.Seek(PodClock, now.TimeOfDay, TimeSeekOrigin.BeginTime);
 }

 private void OnDayChange(object sender, EventArgs e)
 {
 // date has changed, update the datacontext to reflect today's date
 DateTime now = DateTime.Now;
 DataContext = now.Day.ToString();
 _dayTimer.Interval = new TimeSpan(1, 0, 0, 0);
 }
 }
}

```

Il risultato finale è il seguente:



Per confrontare il risultato finale con il codice che ha prodotto questa schermata, vedere [esempio di interoperatività di Win32 Clock](#).

## Vedere anche

- [HwndSource](#)
- [Interoperatività di WPF e Win32](#)
- [Esempio di interoperatività con l'orologio Win32](#)

# Interoperatività di WPF e Direct3D9

28/01/2020 • 17 minutes to read • [Edit Online](#)

È possibile includere contenuto Direct3D9 in un'applicazione Windows Presentation Foundation (WPF). Questo argomento descrive come creare contenuto Direct3D9 in modo che interagisca in modo efficiente con WPF.

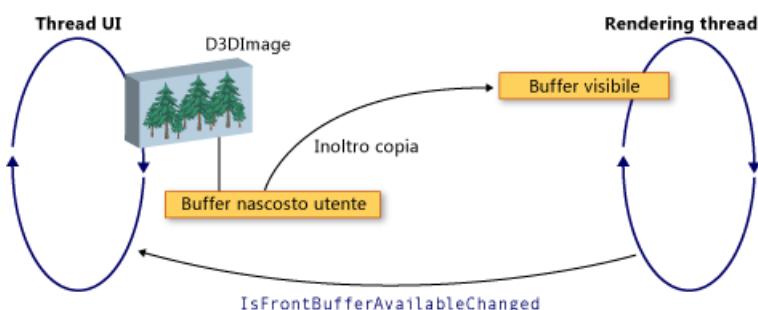
## NOTE

Quando si usa il contenuto Direct3D9 in WPF, è necessario considerare anche le prestazioni. Per ulteriori informazioni su come ottimizzare le prestazioni, vedere [considerazioni sulle prestazioni per l'interoperabilità Direct3D9 e WPF](#).

## Buffer di visualizzazione

La classe [D3DImage](#) gestisce due buffer di visualizzazione, chiamati *buffer nascosto* e il *buffer anteriore*. Il buffer nascosto è la superficie di Direct3D9. Le modifiche apportate al buffer nascosto vengono copiate in avanti nel buffer anteriore quando si chiama il metodo [Unlock](#).

Nella figura seguente viene illustrata la relazione tra il buffer nascosto e il buffer anteriore.



## Creazione del dispositivo Direct3D9

Per eseguire il rendering del contenuto di Direct3D9, è necessario creare un dispositivo Direct3D9. Esistono due oggetti Direct3D9 che è possibile usare per creare un dispositivo, [IDirect3D9](#) e [IDirect3D9Ex](#). Usare questi oggetti per creare rispettivamente i dispositivi [IDirect3DDevice9](#) e [IDirect3DDevice9Ex](#).

Creare un dispositivo chiamando uno dei metodi seguenti.

- `IDirect3D9 * Direct3DCreate9(UINT SDKVersion);`
- `HRESULT Direct3DCreate9Ex(UINT SDKVersion, IDirect3D9Ex **ppD3D);`

In Windows Vista o sistemi operativi successivi utilizzare il metodo [Direct3DCreate9Ex](#) con una visualizzazione configurata per l'utilizzo di Windows Display Driver Model (WDDM). Usare il metodo [Direct3DCreate9](#) su qualsiasi altra piattaforma.

### Disponibilità del metodo Direct3DCreate9Ex

D3d9.dll ha il metodo [Direct3DCreate9Ex](#) solo in Windows Vista o in un sistema operativo successivo. Se si collega direttamente la funzione in Windows XP, il caricamento dell'applicazione non riesce. Per determinare se il metodo [Direct3DCreate9Ex](#) è supportato, caricare la DLL e cercare l'indirizzo proc. Nel codice seguente viene illustrato come eseguire il test per il metodo [Direct3DCreate9Ex](#). Per un esempio di codice completo, vedere [procedura dettagliata: creazione di contenuto Direct3D9 per l'hosting in WPF](#).

```

HRESULT
CRendererManager::EnsureD3DOBJECTS()
{
 HRESULT hr = S_OK;

 HMODULE hD3D = NULL;
 if (!m_pD3D)
 {
 hD3D = LoadLibrary(TEXT("d3d9.dll"));
 DIRECT3DCREATE9EXFUNCTION pfnCreate9Ex = (DIRECT3DCREATE9EXFUNCTION)GetProcAddress(hD3D,
"Direct3DCreate9Ex");
 if (pfnCreate9Ex)
 {
 IFC((*pfnCreate9Ex)(D3D_SDK_VERSION, &m_pD3DEx));
 IFC(m_pD3DEx->QueryInterface(__uuidof(IDirect3D9), reinterpret_cast<void **>(&m_pD3D)));
 }
 else
 {
 m_pD3D = Direct3DCreate9(D3D_SDK_VERSION);
 if (!m_pD3D)
 {
 IFC(E_FAIL);
 }
 }
 }

 m_cAdapters = m_pD3D->GetAdapterCount();
}

Cleanup:
 if (hD3D)
 {
 FreeLibrary(hD3D);
 }

 return hr;
}

```

## Creazione di HWND

Per la creazione di un dispositivo è necessario un HWND. In generale, si crea un HWND fittizio per Direct3D da usare. Nell'esempio di codice seguente viene illustrato come creare un HWND fittizio.

```

HRESULT
CRendererManager::EnsureHWND()
{
 HRESULT hr = S_OK;

 if (!m_hwnd)
 {
 WNDCLASS wndclass;

 wndclass.style = CS_HREDRAW | CS_VREDRAW;
 wndclass.lpfnWndProc = DefWindowProc;
 wndclass.cbClsExtra = 0;
 wndclass.cbWndExtra = 0;
 wndclass.hInstance = NULL;
 wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
 wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);
 wndclass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
 wndclass.lpszMenuName = NULL;
 wndclass.lpszClassName = szAppName;

 if (!RegisterClass(&wndclass))
 {
 IFC(E_FAIL);
 }

 m_hwnd = CreateWindow(szAppName,
 TEXT("D3DImageSample"),
 WS_OVERLAPPEDWINDOW,
 0, // Initial X
 0, // Initial Y
 0, // Width
 0, // Height
 NULL,
 NULL,
 NULL,
 NULL);
 }

 Cleanup:
 return hr;
}

```

## Parametri presenti

La creazione di un dispositivo richiede anche un `D3DPRESENT_PARAMETERS` struct, ma sono importanti solo alcuni parametri. Questi parametri vengono scelti per ridurre al minimo il footprint di memoria.

Impostare i campi `BackBufferHeight` e `BackBufferWidth` su 1. Impostando il valore su 0, tali elementi vengono impostati sulle dimensioni di HWND.

Impostare sempre i flag di `D3DCREATE_MULTITHREADED` e di `D3DCREATE_FPU_PRESERVE` per impedire che la memoria utilizzata da Direct3D9 venga danneggiata e impedire che Direct3D9 modifichi le impostazioni FPU.

Il codice seguente illustra come inizializzare il `D3DPRESENT_PARAMETERS` struct.

```

HRESULT
CRenderer::Init(IDirect3D9 *pD3D, IDirect3D9Ex *pD3DEX, HWND hwnd, UINT uAdapter)
{
 HRESULT hr = S_OK;

 D3DPRESENT_PARAMETERS d3dpp;
 ZeroMemory(&d3dpp, sizeof(d3dpp));
 d3dpp.Windowed = TRUE;
 d3dpp.BackBufferFormat = D3DFMT_UNKNOWN;
 d3dpp.BackBufferHeight = 1;
 d3dpp.BackBufferWidth = 1;
 d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;

 D3DCAPS9 caps;
 DWORD dwVertexProcessing;
 IFC(pD3D->GetDeviceCaps(uAdapter, D3DDEVTYPE_HAL, &caps));
 if ((caps.DevCaps & D3DDEVCAPS_HWTRANSFORMANDLIGHT) == D3DDEVCAPS_HWTRANSFORMANDLIGHT)
 {
 dwVertexProcessing = D3DCREATE_HARDWARE_VERTEXPROCESSING;
 }
 else
 {
 dwVertexProcessing = D3DCREATE_SOFTWARE_VERTEXPROCESSING;
 }

 if (pD3DEX)
 {
 IDirect3DDevice9Ex *pd3dDevice = NULL;
 IFC(pD3DEX->CreateDeviceEx(
 uAdapter,
 D3DDEVTYPE_HAL,
 hwnd,
 dwVertexProcessing | D3DCREATE_MULTITHREADED | D3DCREATE_FPU_PRESERVE,
 &d3dpp,
 NULL,
 &m_pd3dDeviceEx
));

 IFC(m_pd3dDeviceEx->QueryInterface(__uuidof(IDirect3DDevice9), reinterpret_cast<void**>(
 &m_pd3dDevice)));
 }
 else
 {
 assert(pD3D);

 IFC(pD3D->CreateDevice(
 uAdapter,
 D3DDEVTYPE_HAL,
 hwnd,
 dwVertexProcessing | D3DCREATE_MULTITHREADED | D3DCREATE_FPU_PRESERVE,
 &d3dpp,
 &m_pd3dDevice
));
 }

 Cleanup:
 return hr;
}

```

## Creazione della destinazione di rendering del buffer nascosto

Per visualizzare il contenuto di Direct3D9 in una [D3DImage](#), è necessario creare una superficie Direct3D e assegnarla chiamando il metodo [SetBackBuffer](#).

### Verifica del supporto dell'adapter

Prima di creare una superficie, verificare che tutti gli adapter supportino le proprietà di superficie richieste. Anche se si esegue il rendering in una sola scheda, è possibile che la finestra WPF venga visualizzata in qualsiasi scheda del sistema. È necessario scrivere sempre il codice Direct3D9 che gestisce le configurazioni a più schede ed è necessario controllare tutte le schede per il supporto, perché WPF potrebbe spostare la superficie tra le schede disponibili.

Nell'esempio di codice seguente viene illustrato come controllare tutte le schede del sistema per il supporto di Direct3D9.

```

HRESULT
CRendererManager::TestSurfaceSettings()
{
 HRESULT hr = S_OK;

 D3DFORMAT fmt = m_fUseAlpha ? D3DFMT_A8R8G8B8 : D3DFMT_X8R8G8B8;

 //
 // We test all adapters because we potentially use all adapters.
 // But even if this sample only rendered to the default adapter, you
 // should check all adapters because WPF may move your surface to
 // another adapter for you!
 //

 for (UINT i = 0; i < m_cAdapters; ++i)
 {
 // Can we get HW rendering?
 IFC(m_pD3D->CheckDeviceType(
 i,
 D3DDEVTYPE_HAL,
 D3DFMT_X8R8G8B8,
 fmt,
 TRUE
));

 // Is the format okay?
 IFC(m_pD3D->CheckDeviceFormat(
 i,
 D3DDEVTYPE_HAL,
 D3DFMT_X8R8G8B8,
 D3DUSAGE_RENDERTARGET | D3DUSAGE_DYNAMIC, // We'll use dynamic when on XP
 D3DRTYPE_SURFACE,
 fmt
));

 // D3DImage only allows multisampling on 9Ex devices. If we can't
 // multisample, overwrite the desired number of samples with 0.
 if (m_pD3DEx && m_uNumSamples > 1)
 {
 assert(m_uNumSamples <= 16);

 if (FAILED(m_pD3D->CheckDeviceMultiSampleType(
 i,
 D3DDEVTYPE_HAL,
 fmt,
 TRUE,
 static_cast<D3DMULTISAMPLE_TYPE>(m_uNumSamples),
 NULL
)))
 {
 m_uNumSamples = 0;
 }
 }
 else
 {
 m_uNumSamples = 0;
 }
 }

 Cleanup:
 return hr;
}

```

## Creazione della superficie

Prima di creare una superficie, verificare che le funzionalità del dispositivo supportino prestazioni ottimali sul

sistema operativo di destinazione. Per ulteriori informazioni, vedere [considerazioni sulle prestazioni per l'interoperabilità Direct3D9 e WPF](#).

Una volta verificate le funzionalità del dispositivo, è possibile creare la superficie. Nell'esempio di codice seguente viene illustrato come creare la destinazione di rendering.

```
HRESULT
CRenderer::CreateSurface(UINT uWidth, UINT uHeight, bool fUseAlpha, UINT m_uNumSamples)
{
 HRESULT hr = S_OK;

 SAFE_RELEASE(m_pd3dRTS);

 IFC(m_pd3dDevice->CreateRenderTarget(
 uWidth,
 uHeight,
 fUseAlpha ? D3DFMT_A8R8G8B8 : D3DFMT_X8R8G8B8,
 static_cast<D3DMULTISAMPLE_TYPE>(m_uNumSamples),
 0,
 m_pd3dDeviceEx ? FALSE : TRUE, // Lockable RT required for good XP perf
 &m_pd3dRTS,
 NULL
));

 IFC(m_pd3dDevice->SetRenderTarget(0, m_pd3dRTS));

 Cleanup:
 return hr;
}
```

## WDDM

In Windows Vista e nei sistemi operativi successivi, che sono configurati per l'uso di WDDM, è possibile creare una trama di destinazione di rendering e passare la superficie di livello 0 al metodo [SetBackBuffer](#). Questo approccio non è consigliato in Windows XP, perché non è possibile creare una trama di destinazione di rendering bloccabile e le prestazioni risulteranno ridotte.

## Gestione dello stato del dispositivo

La classe [D3DImage](#) gestisce due buffer di visualizzazione, chiamati *buffer nascosto* e il *buffer anteriore*. Il buffer nascosto è la superficie Direct3D. Le modifiche apportate al buffer nascosto vengono copiate in avanti nel buffer anteriore quando si chiama il metodo [Unlock](#), dove viene visualizzato sull'hardware. In alcuni casi, il buffer anteriore diventa non disponibile. Questa mancanza di disponibilità può essere causata da blocchi schermo, applicazioni Direct3D esclusive a schermo intero, cambio utente o altre attività del sistema. In tal caso, l'applicazione WPF riceve una notifica gestendo l'evento [IsFrontBufferAvailableChanged](#). La modalità di risposta dell'applicazione al buffer anteriore diventa non disponibile a seconda che WPF sia abilitato per il fallback al rendering del software. Il metodo [SetBackBuffer](#) dispone di un overload che accetta un parametro che specifica se WPF esegue il fallback al rendering software.

Quando si chiama l'overload [SetBackBuffer\(D3DResourceType, IntPtr\)](#) o si chiama l'overload [SetBackBuffer\(D3DResourceType, IntPtr, Boolean\)](#) con il parametro `enableSoftwareFallback` impostato su `false`, il sistema di rendering rilascia il riferimento al buffer nascosto quando il buffer anteriore diventa non disponibile e non viene visualizzato nulla. Quando il buffer anteriore è nuovamente disponibile, il sistema di rendering genera l'evento [IsFrontBufferAvailableChanged](#) per inviare una notifica all'applicazione WPF. È possibile creare un gestore eventi per l'evento [IsFrontBufferAvailableChanged](#) per riavviare nuovamente il rendering con una superficie Direct3D valida. Per riavviare il rendering, è necessario chiamare [SetBackBuffer](#).

Quando si chiama l'overload di [SetBackBuffer\(D3DResourceType, IntPtr, Boolean\)](#) con il parametro `enableSoftwareFallback` impostato su `true`, il sistema di rendering conserva il riferimento al buffer nascosto

quando il buffer anteriore non è più disponibile, pertanto non è necessario chiamare `SetBackBuffer` quando il buffer anteriore è nuovamente disponibile.

Quando il rendering del software è abilitato, in alcune situazioni il dispositivo dell'utente non è più disponibile, ma il sistema di rendering mantiene un riferimento alla superficie Direct3D. Per verificare se un dispositivo Direct3D9 non è disponibile, chiamare il metodo `TestCooperativeLevel`. Per controllare che i dispositivi Direct3D9Ex chiamino il metodo `CheckDeviceState`, perché il `TestCooperativeLevel` metodo è deprecato e restituisce sempre success. Se il dispositivo utente non è più disponibile, chiamare `SetBackBuffer` per rilasciare il riferimento di WPF al buffer nascosto. Se è necessario reimpostare il dispositivo, chiamare `SetBackBuffer` con il parametro `backBuffer` impostato su `null`, quindi chiamare di nuovo `SetBackBuffer` con `backBuffer` impostato su una superficie Direct3D valida.

Chiamare il metodo `Reset` per eseguire il ripristino da un dispositivo non valido solo se si implementa il supporto per più adapter. In caso contrario, rilasciare tutte le interfacce Direct3D9 e ricrearle completamente. Se il layout dell'adapter è stato modificato, gli oggetti Direct3D9 creati prima della modifica non vengono aggiornati.

## Gestione del ridimensionamento

Se un `D3DImage` viene visualizzato con una risoluzione diversa dalle dimensioni native, viene ridimensionato in base al `BitmapScalingMode` corrente, ad eccezione del fatto che `Bilinear` viene sostituito per `Fant`.

Se è necessaria una maggiore fedeltà, è necessario creare una nuova superficie quando il contenitore del `D3DImage` cambia dimensione.

Sono disponibili tre approcci possibili per gestire il ridimensionamento.

- Partecipare al sistema di layout e creare una nuova superficie quando cambiano le dimensioni. Non creare troppe superfici perché è possibile che si verifichi un esaurimento o un frammento di memoria video.
- Attendere fino a quando non si è verificato un evento di ridimensionamento per un periodo di tempo fisso per la creazione della nuova superficie.
- Creare una `DispatcherTimer` che controlla le dimensioni del contenitore più volte al secondo.

## Ottimizzazione di più monitor

Una riduzione significativa delle prestazioni può verificarsi quando il sistema di rendering sposta un `D3DImage` in un altro monitor.

In WDDM, purché i monitoraggi si trovino nella stessa scheda video e si usi `Direct3DCreate9Ex`, non si verifica alcuna riduzione delle prestazioni. Se i monitoraggi si trovano su schede video separate, le prestazioni sono ridotte. In Windows XP, le prestazioni sono sempre ridotte.

Quando il `D3DImage` si sposta su un altro monitor, è possibile creare una nuova superficie sull'adattatore corrispondente per ripristinare le prestazioni ottimali.

Per evitare la riduzione delle prestazioni, scrivere codice specifico per il caso di più monitor. Nell'elenco seguente viene illustrato un modo per scrivere codice di più monitor.

1. Trovare un punto del `D3DImage` nello spazio dello schermo con il metodo `Visual1.ProjectToScreen`.
2. Utilizzare il `MonitorFromPoint` metodo GDI per individuare il monitoraggio che Visualizza il punto.
3. Utilizzare il metodo `IDirect3D9::GetAdapterMonitor` per individuare l'adattatore Direct3D9 su cui si trova il monitoraggio.
4. Se l'adapter non è uguale all'adapter con il buffer nascosto, creare un nuovo buffer nascosto sul nuovo monitoraggio e assegnarlo alla `D3DImage` buffer nascosto.

#### NOTE

Se il [D3DImage](#) si trova a cavalcioni dei monitoraggi, le prestazioni saranno lente, tranne nel caso di WDDM e [IDirect3D9Ex](#) sulla stessa scheda. In questa situazione non è possibile migliorare le prestazioni.

Nell'esempio di codice riportato di seguito viene illustrato come individuare il monitoraggio corrente.

```
void
CRendererManager::SetAdapter(POINT screenSpacePoint)
{
 CleanupInvalidDevices();

 //
 // After CleanupInvalidDevices, we may not have any D3D objects. Rather than
 // recreate them here, ignore the adapter update and wait for render to recreate.
 //

 if (m_pD3D && m_rgRenderers)
 {
 HMONITOR hMon = MonitorFromPoint(screenSpacePoint, MONITOR_DEFAULTTONULL);

 for (UINT i = 0; i < m_cAdapters; ++i)
 {
 if (hMon == m_pD3D->GetAdapterMonitor(i))
 {
 m_pCurrentRenderer = m_rgRenderers[i];
 break;
 }
 }
 }
}
```

Aggiornare il monitoraggio quando le dimensioni o la posizione del contenitore di [D3DImage](#) cambiano oppure aggiornare il monitoraggio usando un [DispatcherTimer](#) che viene aggiornato alcune volte al secondo.

## Rendering del software WPF

WPF esegue il rendering in modo sincrono sul thread dell'interfaccia utente nel software nelle situazioni seguenti.

- Stampa
- [BitmapEffect](#)
- [RenderTargetBitmap](#)

Quando si verifica una di queste situazioni, il sistema di rendering chiama il metodo [CopyBackBuffer](#) per copiare il buffer hardware nel software. L'implementazione predefinita chiama il metodo [GetRenderTargetData](#) con la superficie. Poiché questa chiamata si verifica al di fuori del modello di blocco/sblocco, potrebbe non riuscire. In questo caso, il metodo [CopyBackBuffer](#) restituisce `null` e non viene visualizzata alcuna immagine.

È possibile eseguire l'override del metodo [CopyBackBuffer](#), chiamare l'implementazione di base e, se viene restituito `null`, è possibile restituire un [BitmapSource](#) segnaposto.

È anche possibile implementare il proprio rendering software anziché chiamare l'implementazione di base.

#### NOTE

Se WPF esegue il rendering completo del software, [D3DImage](#) non viene visualizzato perché WPF non dispone di un buffer anteriore.

## Vedere anche

- [D3DImage](#)
- [Considerazioni sulle prestazioni per l'interoperabilità fra Direct3D9 e WPF](#)
- [Procedura dettagliata: Creazione di contenuto Direct3D9 per l'hosting in WPF](#)
- [Procedura dettagliata: Hosting di contenuto Direct3D9 in WPF](#)

# Considerazioni sulle prestazioni per l'interoperabilità fra Direct3D9 e WPF

31/01/2020 • 8 minutes to read • [Edit Online](#)

È possibile ospitare il contenuto di Direct3D9 usando la classe [D3DImage](#). L'hosting del contenuto di Direct3D9 può influire sulle prestazioni dell'applicazione. In questo argomento vengono descritte le procedure consigliate per ottimizzare le prestazioni quando si ospita il contenuto Direct3D9 in un'applicazione Windows Presentation Foundation (WPF). Queste procedure consigliate includono l'utilizzo di [D3DImage](#) e le procedure consigliate quando si utilizzano le visualizzazioni di Windows Vista, Windows XP e multimonitor.

## NOTE

Per esempi di codice che illustrano queste procedure consigliate, vedere l' [interoperatività di WPF e Direct3D9](#).

## USA D3DImage sporadicamente

Il rendering del contenuto Direct3D9 ospitato in un'istanza di [D3DImage](#) non viene eseguito in modo rapido come in un'applicazione Direct3D pura. La copia della superficie e lo svuotamento del buffer dei comandi possono essere operazioni costose. Man mano che il numero di istanze di [D3DImage](#) aumenta, si verificano più Scaricamenti e si verifica un peggioramento delle prestazioni. Pertanto, è consigliabile utilizzare [D3DImage](#) sporadicamente.

## Procedure consigliate in Windows Vista

Per prestazioni ottimali in Windows Vista con uno schermo configurato per l'uso di Windows Display Driver Model (WDDM), creare la superficie Direct3D9 in un dispositivo [IDirect3DDevice9Ex](#). In questo modo viene abilitata la condivisione della superficie. La scheda video deve supportare le funzionalità di [D3DDEVCAPS2\\_CAN\\_STRETCHRECT\\_FROM\\_TEXTURES](#) e [D3DCAPS2\\_CANSHARERESOURCE](#) driver in Windows Vista. Tutte le altre impostazioni comportano la copia della superficie tramite software, riducendo in modo significativo le prestazioni.

## NOTE

Se Windows Vista dispone di una visualizzazione configurata per l'utilizzo di Windows XP Display Driver Model (XDDM), la superficie viene sempre copiata tramite software, indipendentemente dalle impostazioni. Con le impostazioni e la scheda video appropriate, è possibile ottenere prestazioni migliori in Windows Vista quando si usa WDDM, perché le copie di superficie vengono eseguite nell'hardware.

## Procedure consigliate su Windows XP

Per prestazioni ottimali in Windows XP, che utilizza il modello di driver di visualizzazione di Windows XP (XDDM), creare una superficie bloccabile che funziona correttamente quando viene chiamato il metodo [IDirect3DSurface9::GetDC](#). Internamente, il metodo [BitBlt](#) trasferisce la superficie tra i dispositivi nell'hardware. Il metodo [GetDC](#) funziona sempre sulle superfici sulle XRGB mentre. Tuttavia, se nel computer client è in esecuzione Windows XP con SP3 o SP2 e se il client dispone anche dell'hotfix per la funzionalità finestra a più livelli, questo metodo funziona solo su superfici ARGB. La scheda video deve supportare la funzionalità del driver [D3DDEVCAPS2\\_CAN\\_STRETCHRECT\\_FROM\\_TEXTURES](#).

Una profondità di visualizzazione desktop a 16 bit può ridurre significativamente le prestazioni. È consigliabile usare un desktop a 32 bit.

Se si sta sviluppando per Windows Vista e Windows XP, testare le prestazioni in Windows XP. L'esaurimento della memoria del video in Windows XP costituisce un problema. Inoltre, [D3DImage](#) in Windows XP utilizza una maggiore quantità di memoria e larghezza di banda video rispetto a Windows Vista WDDM, a causa di una copia di memoria video aggiuntiva necessaria. Pertanto, è possibile che le prestazioni siano peggiori in Windows XP rispetto a Windows Vista per lo stesso hardware video.

#### NOTE

XDDM è disponibile sia in Windows XP che in Windows Vista. Tuttavia, WDDM è disponibile solo in Windows Vista.

## Procedure consigliate generali

Quando si crea il dispositivo, usare il flag di creazione `D3DCREATE_MULTITHREADED`. Questo consente di ridurre le prestazioni, ma il sistema di rendering WPF chiama i metodi su questo dispositivo da un altro thread. Assicurarsi di seguire correttamente il protocollo di blocco, in modo che nessuno dei due thread acceda al dispositivo nello stesso momento.

Se il rendering viene eseguito su un thread gestito WPF, è consigliabile creare il dispositivo con il flag di creazione del `D3DCREATE_FPU_PRESERVE`. Senza questa impostazione, il rendering D3D può ridurre l'accuratezza delle operazioni di precisione doppia WPF e introdurre problemi di rendering.

L'affiancamento di un [D3DImage](#) è veloce, a meno che non si riquadri una superficie non pow2 senza supporto hardware oppure se si affianca un [DrawingBrush](#) o [VisualBrush](#) che contiene una [D3DImage](#).

## Procedure consigliate per la visualizzazione di più monitor

Se si utilizza un computer che dispone di più monitoraggi, è necessario attenersi alle procedure consigliate descritte in precedenza. Esistono inoltre alcune considerazioni aggiuntive sulle prestazioni per una configurazione a più monitor.

Quando si crea il buffer nascosto, questo viene creato su un dispositivo e un adattatore specifici, ma WPF può visualizzare il buffer anteriore su qualsiasi scheda. La copia tra gli adapter per aggiornare il buffer anteriore può essere molto costosa. In Windows Vista, configurato per l'uso di WDDM con più schede video e con un dispositivo `IDirect3DDevice9Ex`, se il buffer anteriore si trova in una scheda diversa ma ancora la stessa scheda video, non si verifica alcun calo delle prestazioni. Tuttavia, in Windows XP e XDDM con più schede video, si verifica un calo significativo delle prestazioni quando il buffer anteriore viene visualizzato su un adattatore diverso rispetto al buffer nascosto. Per ulteriori informazioni, vedere l'[interoperatività di WPF e Direct3D9](#).

## Riepilogo prestazioni

La tabella seguente illustra le prestazioni dell'aggiornamento del buffer anteriore come funzione del sistema operativo, del formato pixel e della blocco della superficie. Si presuppone che il buffer anteriore e il buffer indietro si trovino nella stessa scheda. A seconda della configurazione dell'adapter, gli aggiornamenti hardware sono in genere molto più veloci degli aggiornamenti software.

FORMATO PIXEL SUPERFICIE	WINDOWS VISTA, WDDM E 9EX	ALTRÉ CONFIGURAZIONI DI WINDOWS VISTA	WINDOWS XP SP3 O SP2 W/HOTFIX	WINDOWS XP SP2
D3DFMT_X8R8G8B8 (non bloccabile)	<b>Aggiornamento hardware</b>	Aggiornamento software	Aggiornamento software	Aggiornamento software

FORMATO PIXEL SUPERFICIE	WINDOWS VISTA, WDDM E 9EX	ALTRÉ CONFIGURAZIONI DI WINDOWS VISTA	WINDOWS XP SP3 O SP2 W/HOTFIX	WINDOWS XP SP2
D3DFMT_X8R8G8B8 (bloccabile)	<b>Aggiornamento hardware</b>	Aggiornamento software	<b>Aggiornamento hardware</b>	<b>Aggiornamento hardware</b>
D3DFMT_A8R8G8B8 (non bloccabile)	<b>Aggiornamento hardware</b>	Aggiornamento software	Aggiornamento software	Aggiornamento software
D3DFMT_A8R8G8B8 (bloccabile)	<b>Aggiornamento hardware</b>	Aggiornamento software	<b>Aggiornamento hardware</b>	Aggiornamento software

## Vedere anche

- [D3DImage](#)
- [Interoperatività di WPF e Direct3D9](#)
- [Procedura dettagliata: Creazione di contenuto Direct3D9 per l'hosting in WPF](#)
- [Procedura dettagliata: Hosting di contenuto Direct3D9 in WPF](#)

# Procedura dettagliata: creazione di contenuto Direct3D9 per l'hosting in WPF

03/02/2020 • 16 minutes to read • [Edit Online](#)

In questa procedura dettagliata viene illustrato come creare contenuto Direct3D9 adatto per l'hosting in un'applicazione Windows Presentation Foundation (WPF). Per ulteriori informazioni sull'hosting di contenuto Direct3D9 nelle applicazioni WPF, vedere [WPF and Direct3D9 Interoperation](#).

Questa procedura dettagliata prevede l'esecuzione delle attività seguenti:

- Creare un progetto Direct3D9.
- Configurare il progetto Direct3D9 per l'hosting in un'applicazione WPF.

Al termine, si disporrà di una DLL che contiene contenuto Direct3D9 per l'uso in un'applicazione WPF.

## Prerequisiti

Per completare la procedura dettagliata, è necessario disporre dei componenti seguenti:

- Visual Studio 2010.
- DirectX SDK 9 o versione successiva.

## Creazione del progetto Direct3D9

Il primo passaggio consiste nel creare e configurare il progetto Direct3D9.

### Per creare il progetto Direct3D9

1. Creare un nuovo progetto Win32 in C++ denominato `D3DContent`.

Si apre la creazione guidata applicazione Win32 e viene visualizzata la schermata iniziale.

2. Fare clic su **Avanti**.

Viene visualizzata la schermata Impostazioni applicazione.

3. Nella sezione **tipo di applicazione**: selezionare l'opzione **dll**.

4. Fare clic su **Fine**.

Viene generato il progetto D3DContent.

5. In Esplora soluzioni fare clic con il pulsante destro del mouse sul progetto D3DContent e scegliere **Proprietà**.

Verrà visualizzata la finestra di dialogo **pagine delle proprietà di D3DContent**.

6. Selezionare il nodo **CC++ /**.

7. Nel campo **directory di inclusione aggiuntive** specificare il percorso della cartella di inclusione DirectX. Il percorso predefinito per questa cartella è %Programmi%\Microsoft DirectX SDK (Version) \Include.

8. Fare doppio clic sul nodo **linker** per espanderlo.

9. Nel campo **Directory librerie aggiuntive** specificare il percorso della cartella librerie DirectX. Il percorso

predefinito per questa cartella è %Programmi%\Microsoft DirectX SDK (Version) \Lib\x86.

10. Selezionare il nodo di **input**.
11. Nel campo **dipendenze aggiuntive** aggiungere i file di `d3d9.lib` e `d3dx9.lib`.
12. In Esplora soluzioni aggiungere un nuovo file di definizione del modulo (con estensione def) denominato `D3DContent.def` al progetto.

## Creazione del contenuto di Direct3D9

Per ottenere prestazioni ottimali, il contenuto di Direct3D9 deve usare impostazioni particolari. Nel codice seguente viene illustrato come creare una superficie Direct3D9 con le caratteristiche di prestazioni ottimali. Per ulteriori informazioni, vedere [considerazioni sulle prestazioni per l'interoperabilità Direct3D9 e WPF](#).

### Per creare il contenuto di Direct3D9

1. Utilizzando Esplora soluzioni, aggiungere tre C++ classi al progetto denominate quanto segue.

`CRenderer` (con distruttore virtuale)

`CRendererManager`

`CTriangleRenderer`

2. Aprire `Renderer.h` nell'editor di codice e sostituire il codice generato automaticamente con il codice seguente.

```
#pragma once

class CRenderer
{
public:
 virtual ~CRenderer();

 HRESULT CheckDeviceState();
 HRESULT CreateSurface(UINT uWidth, UINT uHeight, bool fUseAlpha, UINT m_uNumSamples);

 virtual HRESULT Render() = 0;

 IDirect3DSurface9 *GetSurfaceNoRef() { return m_pd3dRTS; }

protected:
 CRenderer();

 virtual HRESULT Init(IDirect3D9 *pD3D, IDirect3D9Ex *pD3DEx, HWND hwnd, UINT uAdapter);

 IDirect3DDevice9 *m_pd3dDevice;
 IDirect3DDevice9Ex *m_pd3dDeviceEx;

 IDirect3DSurface9 *m_pd3dRTS;

};
```

3. Aprire `Renderer.cpp` nell'editor di codice e sostituire il codice generato automaticamente con il codice seguente.

```
//-----
//
// CRenderer
//
// An abstract base class that creates a device and a target render
// surface. Derive from this class and override TInit() and Render()
```

```

// Surface. Derive from this class and override Init() and Render()
// to do your own rendering. See CTriangleRenderer for an example.

//-----
//include "StdAfx.h"

//+-----
// Member:
// CRenderer ctor
//
//-----
CRenderer::CRenderer() : m_pd3dDevice(NULL), m_pd3dDeviceEx(NULL), m_pd3dRTS(NULL)
{
}

//+-----
// Member:
// CRenderer dtor
//
//-----
CRenderer::~CRenderer()
{
 SAFE_RELEASE(m_pd3dDevice);
 SAFE_RELEASE(m_pd3dDeviceEx);
 SAFE_RELEASE(m_pd3dRTS);
}

//+-----
// Member:
// CRenderer::CheckDeviceState
//
// Synopsis:
// Returns the status of the device. 9Ex devices are a special case because
// TestCooperativeLevel() has been deprecated in 9Ex.
//
//-----
HRESULT
CRenderer::CheckDeviceState()
{
 if (m_pd3dDeviceEx)
 {
 return m_pd3dDeviceEx->CheckDeviceState(NULL);
 }
 else if (m_pd3dDevice)
 {
 return m_pd3dDevice->TestCooperativeLevel();
 }
 else
 {
 return D3DERR_DEVICELOST;
 }
}

//+-----
// Member:
// CRenderer::CreateSurface
//
// Synopsis:
// Creates and sets the render target
//
//-----
HRESULT
CRenderer::CreateSurface(UINT uWidth, UINT uHeight, bool fUseAlpha, UINT m_uNumSamples)
{
}

```

```

 {
 HRESULT hr = S_OK;

 SAFE_RELEASE(m_pd3dRTS);

 IFC(m_pd3DDevice->CreateRenderTarget(
 uWidth,
 uHeight,
 fUseAlpha ? D3DFMT_A8R8G8B8 : D3DFMT_X8R8G8B8,
 static_cast<D3DMULTISAMPLE_TYPE>(m_uNumSamples),
 0,
 m_pd3dDeviceEx ? FALSE : TRUE, // Lockable RT required for good XP perf
 &m_pd3dRTS,
 NULL
));

 IFC(m_pd3DDevice->SetRenderTarget(0, m_pd3dRTS));

 Cleanup:
 return hr;
 }

//+-----
// Member:
// CRenderer::Init
//
// Synopsis:
// Creates the device
//-
HRESULT CRenderer::Init(IDirect3D9 *pD3D, IDirect3D9Ex *pD3DEx, HWND hwnd, UINT uAdapter)
{
 HRESULT hr = S_OK;

 D3DPRESENT_PARAMETERS d3dpp;
 ZeroMemory(&d3dpp, sizeof(d3dpp));
 d3dpp.Windowed = TRUE;
 d3dpp.BackBufferFormat = D3DFMT_UNKNOWN;
 d3dpp.BackBufferHeight = 1;
 d3dpp.BackBufferWidth = 1;
 d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;

 D3DCAPS9 caps;
 DWORD dwVertexProcessing;
 IFC(pD3D->GetDeviceCaps(uAdapter, D3DDEVTYPE_HAL, &caps));
 if ((caps.DevCaps & D3DDEVCAPS_HWTRANSFORMANDLIGHT) == D3DDEVCAPS_HWTRANSFORMANDLIGHT)
 {
 dwVertexProcessing = D3DCREATE_HARDWARE_VERTEXPROCESSING;
 }
 else
 {
 dwVertexProcessing = D3DCREATE_SOFTWARE_VERTEXPROCESSING;
 }

 if (pD3DEx)
 {
 IDirect3DDevice9Ex *pd3dDevice = NULL;
 IFC(pD3DEx->CreateDeviceEx(
 uAdapter,
 D3DDEVTYPE_HAL,
 hwnd,
 dwVertexProcessing | D3DCREATE_MULTITHREADED | D3DCREATE_FPU_PRESERVE,
 &d3dpp,
 NULL,
 &m_pd3dDeviceEx
));
 }
}

```

```
 IFC(m_pd3dDeviceEx->QueryInterface(__uuidof(IDirect3DDevice9), reinterpret_cast<void**>
(&m_pd3dDevice)));
}
else
{
 assert(pD3D);

 IFC(pDXD->CreateDevice(
 uAdapter,
 D3DDEVTYPE_HAL,
 hwnd,
 dwVertexProcessing | D3DCREATE_MULTITHREADED | D3DCREATE_FPU_PRESERVE,
 &d3dpp,
 &m_pd3dDevice
));
}

Cleanup:
return hr;
}
```

4. Aprire RendererManager.h nell'editor di codice e sostituire il codice generato automaticamente con il codice seguente.

```

#pragma once

class CRenderer;

class CRendererManager
{
public:
 static HRESULT Create(CRendererManager **ppManager);
 ~CRendererManager();

 HRESULT EnsureDevices();

 void SetSize(UINT uWidth, UINT uHeight);
 void SetAlpha(bool fUseAlpha);
 void SetNumDesiredSamples(UINT uNumSamples);
 void SetAdapter(POINT screenSpacePoint);

 HRESULT GetBackBufferNoRef(IDirect3DSurface9 **ppSurface);

 HRESULT Render();

private:
 CRendererManager();

 void CleanupInvalidDevices();
 HRESULT EnsureRenderers();
 HRESULT EnsureHWND();
 HRESULT EnsureD3DOBJECTS();
 HRESULT TestSurfaceSettings();
 void DestroyResources();

 IDirect3D9 *m_pD3D;
 IDirect3D9Ex *m_pD3DEx;

 UINT m_cAdapters;
 CRenderer **m_rgRenderers;
 CRenderer *m_pCurrentRenderer;

 HWND m_hwnd;

 UINT m_uWidth;
 UINT m_uHeight;
 UINT m_uNumSamples;
 bool m_fUseAlpha;
 bool m_fSurfaceSettingsChanged;
};


```

- Aprire RendererManager.cpp nell'editor di codice e sostituire il codice generato automaticamente con il codice seguente.

```

//-----
// CRendererManager
//
// Manages the list of CRenderers. Managed code pinvoke into this class
// and this class forwards to the appropriate CRenderer.
//
//-----

#include "StdAfx.h"

const static TCHAR szAppName[] = TEXT("D3DImageSample");
typedef HRESULT (WINAPI *DIRECT3DCREATE9EXFUNCTION)(UINT SDKVersion, IDirect3D9Ex**);

```

```

//+-----
// Member:
// CRendererManager ctor
//-
//-
CRendererManager::CRendererManager()
:
 m_pD3D(NULL),
 m_pD3DEx(NULL),
 m_cAdapters(0),
 m_hwnd(NULL),
 m_pCurrentRenderer(NULL),
 m_rgRenderers(NULL),
 m_uWidth(1024),
 m_uHeight(1024),
 m_uNumSamples(0),
 m_fUseAlpha(false),
 m_fSurfaceSettingsChanged(true)
{
}

//+-----
// Member:
// CRendererManager dtor
//-
//-
CRendererManager::~CRendererManager()
{
 DestroyResources();

 if (m_hwnd)
 {
 DestroyWindow(m_hwnd);
 UnregisterClass(szAppName, NULL);
 }
}

//+-----
// Member:
// CRendererManager::Create
//-
// Synopsis:
// Creates the manager
//-
//-
HRESULT
CRendererManager::Create(CRendererManager **ppManager)
{
 HRESULT hr = S_OK;

 *ppManager = new CRendererManager();
 IFCOOM(*ppManager);

 Cleanup:
 return hr;
}

//+-----
// Member:
// CRendererManager::EnsureRenderers
//-
// Synopsis:
// Makes sure the CRenderer objects exist
//-

```

```

/*
//-----
HRESULT
CRendererManager::EnsureRenderers()
{
 HRESULT hr = S_OK;

 if (!m_rgRenderers)
 {
 IFC(EnsureHWNDF());

 assert(m_cAdapters);
 m_rgRenderers = new CRenderer*[m_cAdapters];
 IFCOM(m_rgRenderers);
 ZeroMemory(m_rgRenderers, m_cAdapters * sizeof(m_rgRenderers[0]));

 for (UINT i = 0; i < m_cAdapters; ++i)
 {
 IFC(CTriangleRenderer::Create(m_pD3D, m_pD3DEx, m_hwnd, i, &m_rgRenderers[i]));
 }

 // Default to the default adapter
 m_pCurrentRenderer = m_rgRenderers[0];
 }

 Cleanup:
 return hr;
}

//+-----
// Member:
// CRendererManager::EnsureHWNDF
//
// Synopsis:
// Makes sure an HWND exists if we need it
//
//-----

HRESULT
CRendererManager::EnsureHWNDF()
{
 HRESULT hr = S_OK;

 if (!m_hwnd)
 {
 WNDCLASS wndclass;

 wndclass.style = CS_HREDRAW | CS_VREDRAW;
 wndclass.lpfWndProc = DefWindowProc;
 wndclass.cbClsExtra = 0;
 wndclass.cbWndExtra = 0;
 wndclass.hInstance = NULL;
 wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
 wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);
 wndclass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
 wndclass.lpszMenuName = NULL;
 wndclass.lpszClassName = szAppName;

 if (!RegisterClass(&wndclass))
 {
 IFC(E_FAIL);
 }

 m_hwnd = CreateWindow(szAppName,
 TEXT("D3DImageSample"),
 WS_OVERLAPPEDWINDOW,
 0, // Initial X
 0, // Initial Y
 0, // Width
 ^ // Height
);
 }
}

```

```

 0, // Height
 NULL,
 NULL,
 NULL,
 NULL);
}

Cleanup:
 return hr;
}

//+-----+
//
// Member:
// CRendererManager::EnsureD3DObjects
//
// Synopsis:
// Makes sure the D3D objects exist
//
//-----

HRESULT
CRendererManager::EnsureD3DObjects()
{
 HRESULT hr = S_OK;

 HMODULE hD3D = NULL;
 if (!m_pD3D)
 {
 hD3D = LoadLibrary(TEXT("d3d9.dll"));
 DIRECT3DCREATE9EXFUNCTION pfnCreate9Ex = (DIRECT3DCREATE9EXFUNCTION)GetProcAddress(hD3D,
"Direct3DCreate9Ex");
 if (pfnCreate9Ex)
 {
 IFC((*pfnCreate9Ex)(D3D_SDK_VERSION, &m_pD3DEx));
 IFC(m_pD3DEx->QueryInterface(__uuidof(IDirect3D9), reinterpret_cast<void **>(&m_pD3D)));
 }
 else
 {
 m_pD3D = Direct3DCreate9(D3D_SDK_VERSION);
 if (!m_pD3D)
 {
 IFC(E_FAIL);
 }
 }
 }

 m_cAdapters = m_pD3D->GetAdapterCount();
}

Cleanup:
 if (hD3D)
 {
 FreeLibrary(hD3D);
 }

 return hr;
}

//+-----+
//
// Member:
// CRendererManager::CleanupInvalidDevices
//
// Synopsis:
// Checks to see if any devices are bad and if so, deletes all resources
//
// We could delete resources and wait for D3DERR_DEVICENOTRESET and reset
// the devices, but if the device is lost because of an adapter order
// change then our existing D3D objects would have stale adapter
// information. We'll delete everything to be safe rather than sorry.

```

```

// -----
// void
CRendererManager::CleanupInvalidDevices()
{
 for (UINT i = 0; i < m_cAdapters; ++i)
 {
 if (FAILED(m_rgRenderers[i]->CheckDeviceState()))
 {
 DestroyResources();
 break;
 }
 }
}

//+-----
// Member:
// CRendererManager::GetBackBufferNoRef
//
// Synopsis:
// Returns the surface of the current renderer without adding a reference
//
// This can return NULL if we're in a bad device state.
//
// -----
HRESULT
CRendererManager::GetBackBufferNoRef(IDirect3DSurface9 **ppSurface)
{
 HRESULT hr = S_OK;

 // Make sure we at least return NULL
 *ppSurface = NULL;

 CleanupInvalidDevices();

 IFC(EnsureD3DOBJECTS());

 //
 // Even if we never render to another adapter, this sample creates devices
 // and resources on each one. This is a potential waste of video memory,
 // but it guarantees that we won't have any problems (e.g. out of video
 // memory) when switching to render on another adapter. In your own code
 // you may choose to delay creation but you'll need to handle the issues
 // that come with it.
 //

 IFC(EnsureRenderers());

 if (m_fSurfaceSettingsChanged)
 {
 if (FAILED(TestSurfaceSettings()))
 {
 IFC(E_FAIL);
 }

 for (UINT i = 0; i < m_cAdapters; ++i)
 {
 IFC(m_rgRenderers[i]->CreateSurface(m_uWidth, m_uHeight, m_fUseAlpha, m_uNumSamples));
 }

 m_fSurfaceSettingsChanged = false;
 }

 if (m_pCurrentRenderer)
 {
 *ppSurface = m_pCurrentRenderer->GetSurfaceNoRef();
 }
}

```

```

Cleanup:
 // If we failed because of a bad device, ignore the failure for now and
 // we'll clean up and try again next time.
 if (hr == D3DERR_DEVICELOST)
 {
 hr = S_OK;
 }

 return hr;
}

//-----
// Member:
// CRendererManager::TestSurfaceSettings
//
// Synopsis:
// Checks to see if our current surface settings are allowed on all
// adapters.
//
//-----

HRESULT
CRendererManager::TestSurfaceSettings()
{
 HRESULT hr = S_OK;

 D3DFORMAT fmt = m_fUseAlpha ? D3DFMT_A8R8G8B8 : D3DFMT_X8R8G8B8;

 //
 // We test all adapters because because we potentially use all adapters.
 // But even if this sample only rendered to the default adapter, you
 // should check all adapters because WPF may move your surface to
 // another adapter for you!
 //

 for (UINT i = 0; i < m_cAdapters; ++i)
 {
 // Can we get HW rendering?
 IFC(m_pD3D->CheckDeviceType(
 i,
 D3DDEVTYPE_HAL,
 D3DFMT_X8R8G8B8,
 fmt,
 TRUE
));

 // Is the format okay?
 IFC(m_pD3D->CheckDeviceFormat(
 i,
 D3DDEVTYPE_HAL,
 D3DFMT_X8R8G8B8,
 D3DUSAGE_RENDERTARGET | D3DUSAGE_DYNAMIC, // We'll use dynamic when on XP
 D3DRTYPE_SURFACE,
 fmt
));

 // D3DImage only allows multisampling on 9Ex devices. If we can't
 // multisample, overwrite the desired number of samples with 0.
 if (m_pD3DEx && m_uNumSamples > 1)
 {
 assert(m_uNumSamples <= 16);

 if (FAILED(m_pD3D->CheckDeviceMultiSampleType(
 i,
 D3DDEVTYPE_HAL,
 fmt,
 TRUE,
 static_cast<D3DMULTISAMPLE_TYPE>(m_uNumSamples),
 NULL
)))
 }
 }
}

```

```

)))

 {
 m_uNumSamples = 0;
 }
}

else
{
 m_uNumSamples = 0;
}
}

Cleanup:
 return hr;
}

//+-----
// Member:
// CRendererManager::DestroyResources
//
// Synopsis:
// Delete all D3D resources
//
//-----
void
CRendererManager::DestroyResources()
{
 SAFE_RELEASE(m_pD3D);
 SAFE_RELEASE(m_pD3DEX);

 for (UINT i = 0; i < m_cAdapters; ++i)
 {
 delete m_rgRenderers[i];
 }
 delete [] m_rgRenderers;
 m_rgRenderers = NULL;

 m_pCurrentRenderer = NULL;
 m_cAdapters = 0;

 m_fSurfaceSettingsChanged = true;
}

//+-----
// Member:
// CRendererManager::SetSize
//
// Synopsis:
// Update the size of the surface. Next render will create a new surface.
//
//-----
void
CRendererManager::SetSize(UINT uWidth, UINT uHeight)
{
 if (uWidth != m_uWidth || uHeight != m_uHeight)
 {
 m_uWidth = uWidth;
 m_uHeight = uHeight;
 m_fSurfaceSettingsChanged = true;
 }
}

//+-----
// Member:
// CRendererManager::SetAlpha
//
// Synopsis:

```

```

// Update the format of the surface. Next render will create a new surface.
//

//-----

void

CRendererManager::SetAlpha(bool fUseAlpha)

{

 if (fUseAlpha != m_fUseAlpha)

 {

 m_fUseAlpha = fUseAlpha;

 m_fSurfaceSettingsChanged = true;

 }

}

//+-----

//

// Member:

// CRendererManager::SetNumDesiredSamples

//

// Synopsis:

// Update the MSAA settings of the surface. Next render will create a

// new surface.

//

//-----

void

CRendererManager::SetNumDesiredSamples(UINT uNumSamples)

{

 if (m_uNumSamples != uNumSamples)

 {

 m_uNumSamples = uNumSamples;

 m_fSurfaceSettingsChanged = true;

 }

}

//+-----

//

// Member:

// CRendererManager::SetAdapter

//

// Synopsis:

// Update the current renderer. Next render will use the new renderer.

//

//-----

void

CRendererManager::SetAdapter(POINT screenSpacePoint)

{

 CleanupInvalidDevices();

 //

 // After CleanupInvalidDevices, we may not have any D3D objects. Rather than

 // recreate them here, ignore the adapter update and wait for render to recreate.

 //

 if (m_pD3D && m_rgRenderers)

 {

 HMONITOR hMon = MonitorFromPoint(screenSpacePoint, MONITOR_DEFAULTTONULL);

 for (UINT i = 0; i < m_cAdapters; ++i)

 {

 if (hMon == m_pD3D->GetAdapterMonitor(i))

 {

 m_pCurrentRenderer = m_rgRenderers[i];

 break;

 }

 }
 }
}

//+-----

//
```

```


// Member:
// CRendererManager::Render
//
// Synopsis:
// Forward to the current renderer
//
//-----
HRESULT
CRendererManager::Render()
{
 return m_pCurrentRenderer ? m_pCurrentRenderer->Render() : S_OK;
}


```

6. Aprire TriangleRenderer.h nell'editor di codice e sostituire il codice generato automaticamente con il codice seguente.

```


#pragma once

class CTriangleRenderer : public CRenderer
{
public:
 static HRESULT Create(IDirect3D9 *pD3D, IDirect3D9Ex *pD3DEX, HWND hwnd, UINT uAdapter, CRenderer
 **ppRenderer);
 ~CTriangleRenderer();

 HRESULT Render();

protected:
 HRESULT Init(IDirect3D9 *pD3D, IDirect3D9Ex *pD3DEX, HWND hwnd, UINT uAdapter);

private:
 CTriangleRenderer();
 IDirect3DVertexBuffer9 *m_pd3dVB;
};


```

7. Aprire TriangleRenderer.cpp nell'editor di codice e sostituire il codice generato automaticamente con il codice seguente.

```


//+
// CTriangleRenderer
// Subclass of CRenderer that renders a single, spinning triangle
//-
#include "StdAfx.h"

struct CUSTOMVERTEX
{
 FLOAT x, y, z;
 DWORD color;
};

#define D3DFVF_CUSTOMVERTEX (D3DFVF_XYZ | D3DFVF_DIFFUSE)

//+
// Member:
// CTriangleRenderer ctor
//-


```

```

CTriangleRenderer::CTriangleRenderer() : CRenderer(), m_pd3dVB(NULL)
{
}

//+-----
// Member:
// CTriangleRenderer dtor
//
//-----
CTriangleRenderer::~CTriangleRenderer()
{
 SAFE_RELEASE(m_pd3dVB);
}

//+-----
// Member:
// CTriangleRenderer::Create
//
// Synopsis:
// Creates the renderer
//
//-----
HRESULT
CTriangleRenderer::Create(IDirect3D9 *pD3D, IDirect3D9Ex *pD3DEx, HWND hwnd, UINT uAdapter, CRenderer
**ppRenderer)
{
 HRESULT hr = S_OK;

 CTriangleRenderer *pRenderer = new CTriangleRenderer();
 IFCOOM(pRenderer);

 IFC(pRenderer->Init(pD3D, pD3DEx, hwnd, uAdapter));

 *ppRenderer = pRenderer;
 pRenderer = NULL;

 Cleanup:
 delete pRenderer;

 return hr;
}

//+-----
// Member:
// CTriangleRenderer::Init
//
// Synopsis:
// Override of CRenderer::Init that calls base to create the device and
// then creates the CTriangleRenderer-specific resources
//
//-----
HRESULT
CTriangleRenderer::Init(IDirect3D9 *pD3D, IDirect3D9Ex *pD3DEx, HWND hwnd, UINT uAdapter)
{
 HRESULT hr = S_OK;
 D3DXMATRIXA16 matView, matProj;
 D3DXVECTOR3 vEyePt(0.0f, 0.0f, -5.0f);
 D3DXVECTOR3 vLookatPt(0.0f, 0.0f, 0.0f);
 D3DXVECTOR3 vUpVec(0.0f, 1.0f, 0.0f);

 // Call base to create the device and render target
 IFC(CRenderer::Init(pD3D, pD3DEx, hwnd, uAdapter));

 // Set up the VB
 CUSTOMVERTEX vertices[] =
 {

```

```

 { -1.0f, -1.0f, 0.0f, 0xfffff0000, }, // x, y, z, color
 { 1.0f, -1.0f, 0.0f, 0xff00ff00, },
 { 0.0f, 1.0f, 0.0f, 0xff00ffff, },
 };

 IFC(m_pd3dDevice->CreateVertexBuffer(sizeof(vertices), 0, D3DFVF_CUSTOMVERTEX, D3DPOOL_DEFAULT,
&m_pd3dVB, NULL));

 void *pVertices;
 IFC(m_pd3dVB->Lock(0, sizeof(vertices), &pVertices, 0));
 memcpy(pVertices, vertices, sizeof(vertices));
 m_pd3dVB->Unlock();

 // Set up the camera
 D3DXMatrixLookAtLH(&matView, &vEyePt, &vLookatPt, &vUpVec);
 IFC(m_pd3dDevice->SetTransform(D3DTS_VIEW, &matView));
 D3DXMatrixPerspectiveFovLH(&matProj, D3DX_PI / 4, 1.0f, 1.0f, 100.0f);
 IFC(m_pd3dDevice->SetTransform(D3DTS_PROJECTION, &matProj));

 // Set up the global state
 IFC(m_pd3dDevice->SetRenderState(D3DRS_CULLMODE, D3DCULL_NONE));
 IFC(m_pd3dDevice->SetRenderState(D3DRS_LIGHTING, FALSE));
 IFC(m_pd3dDevice->SetStreamSource(0, m_pd3dVB, 0, sizeof(CUSTOMVERTEX)));
 IFC(m_pd3dDevice->SetFVF(D3DFVF_CUSTOMVERTEX));

 Cleanup:
 return hr;
}

//+-----
// Member:
// CTriangleRenderer::Render
//
// Synopsis:
// Renders the rotating triangle
//-
HRESULT
CTriangleRenderer::Render()
{
 HRESULT hr = S_OK;
 D3DXMATRIXA16 matWorld;

 IFC(m_pd3dDevice->BeginScene());
 IFC(m_pd3dDevice->Clear(
 0,
 NULL,
 D3DCLEAR_TARGET,
 D3DCOLOR_ARGB(128, 0, 0, 128), // NOTE: Premultiplied alpha!
 1.0f,
 0
));

 // Set up the rotation
 UINT iTime = GetTickCount() % 1000;
 FLOAT fAngle = iTime * (2.0f * D3DX_PI) / 1000.0f;
 D3DXMatrixRotationY(&matWorld, fAngle);
 IFC(m_pd3dDevice->SetTransform(D3DTS_WORLD, &matWorld));

 IFC(m_pd3dDevice->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 1));

 IFC(m_pd3dDevice->EndScene());

 Cleanup:
 return hr;
}

```

8. Aprire stdafx.h nell'editor di codice e sostituire il codice generato automaticamente con il codice seguente.

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#pragma once

#define WIN32_LEAN_AND_MEAN // Exclude rarely-used stuff from Windows headers
// Windows Header Files:
#include <windows.h>

#include <d3d9.h>
#include <d3dx9.h>

#include <assert.h>

#include "RendererManager.h"
#include "Renderer.h"
#include "TriangleRenderer.h"

#define IFC(x) { hr = (x); if (FAILED(hr)) goto Cleanup; }
#define IFCOOM(x) { if ((x) == NULL) { hr = E_OUTOFMEMORY; IFC(hr); } }
#define SAFE_RELEASE(x) { if (x) { x->Release(); x = NULL; } }

Cleanup:
```

9. Aprire DllMain.cpp nell'editor di codice e sostituire il codice generato automaticamente con il codice seguente.

```
// dllmain.cpp : Defines the entry point for the DLL application.
#include "stdafx.h"

BOOL APIENTRY DllMain(HMODULE hModule,
 DWORD ul_reason_for_call,
 LPVOID lpReserved
)
{
 switch (ul_reason_for_call)
 {
 case DLL_PROCESS_ATTACH:
 case DLL_THREAD_ATTACH:
 case DLL_THREAD_DETACH:
 case DLL_PROCESS_DETACH:
 break;
 }
 return TRUE;
}

static CRendererManager *pManager = NULL;

static HRESULT EnsureRendererManager()
{
 return pManager ? S_OK : CRendererManager::Create(&pManager);
}

extern "C" HRESULT WINAPI SetSize(UINT uWidth, UINT uHeight)
{
 HRESULT hr = S_OK;

 IFC(EnsureRendererManager());

 pManager->SetSize(uWidth, uHeight);

Cleanup:
 return hr;
```

```

 }

extern "C" HRESULT WINAPI SetAlpha(BOOL fUseAlpha)
{
 HRESULT hr = S_OK;

 IFC(EnsureRendererManager());

 pManager->SetAlpha(!fUseAlpha);

Cleanup:
 return hr;
}

extern "C" HRESULT WINAPI SetNumDesiredSamples(UINT uNumSamples)
{
 HRESULT hr = S_OK;

 IFC(EnsureRendererManager());

 pManager->SetNumDesiredSamples(uNumSamples);

Cleanup:
 return hr;
}

extern "C" HRESULT WINAPI SetAdapter(POINT screenSpacePoint)
{
 HRESULT hr = S_OK;

 IFC(EnsureRendererManager());

 pManager->SetAdapter(screenSpacePoint);

Cleanup:
 return hr;
}

extern "C" HRESULT WINAPI GetBackBufferNoRef(IDirect3DSurface9 **ppSurface)
{
 HRESULT hr = S_OK;

 IFC(EnsureRendererManager());

 IFC(pManager->GetBackBufferNoRef(ppSurface));

Cleanup:
 return hr;
}

extern "C" HRESULT WINAPI Render()
{
 assert(pManager);

 return pManager->Render();
}

extern "C" void WINAPI Destroy()
{
 delete pManager;
 pManager = NULL;
}

```

10. Aprire D3DContent.def nell'editor di codice.
11. Sostituire il codice generato automaticamente con il codice seguente.

```
LIBRARY "D3DContent"
```

```
EXPORTS
```

```
SetSize
SetAlpha
SetNumDesiredSamples
SetAdapter
```

```
GetBackBufferNoRef
Render
Destroy
```

12. Compilazione del progetto.

## Passaggi successivi

- Ospitare il contenuto di Direct3D9 in un'applicazione WPF. Per ulteriori informazioni, vedere [procedura dettagliata: hosting di contenuto Direct3D9 in WPF](#).

## Vedere anche

- [D3DImage](#)
- [Considerazioni sulle prestazioni per l'interoperabilità fra Direct3D9 e WPF](#)
- [Procedura dettagliata: Hosting di contenuto Direct3D9 in WPF](#)

# Procedura dettagliata: hosting di contenuto Direct3D9 in WPF

28/01/2020 • 7 minutes to read • [Edit Online](#)

In questa procedura dettagliata viene illustrato come ospitare il contenuto di Direct3D9 in un'applicazione Windows Presentation Foundation (WPF).

Questa procedura dettagliata prevede l'esecuzione delle attività seguenti:

- Creare un progetto WPF per ospitare il contenuto di Direct3D9.
- Importa il contenuto di Direct3D9.
- Visualizzare il contenuto Direct3D9 usando la classe [D3DImage](#).

Al termine, si saprà come ospitare il contenuto di Direct3D9 in un'applicazione WPF.

## Prerequisiti

Per completare la procedura dettagliata, è necessario disporre dei componenti seguenti:

- Visual Studio.
- DirectX SDK 9 o versione successiva.
- Una DLL che contiene contenuto Direct3D9 in un formato compatibile con WPF. Per altre informazioni, vedere [interoperatività di WPF e Direct3D9](#) e [procedura dettagliata: creazione di contenuto Direct3D9 per l'hosting in WPF](#).

## Creazione del progetto WPF

Il primo passaggio consiste nel creare il progetto per l'applicazione WPF.

### Per creare il progetto WPF

Creare un nuovo progetto di applicazione WPF in C# un oggetto visivo denominato [D3DHost](#). Per altre informazioni, vedere [Procedura dettagliata: La prima applicazione desktop WPF](#).

MainWindow. XAML viene aperto in WPF Designer.

## Importazione del contenuto di Direct3D9

Il contenuto Direct3D9 viene importato da una DLL non gestita usando l'attributo [DllImport](#).

### Per importare il contenuto di Direct3D9

1. Aprire MainWindow.xaml.cs nell'editor di codice.
2. Sostituire il codice generato automaticamente con il codice seguente.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
```

```
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Interop;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Windows.Threading;
using System.Runtime.InteropServices;
using System.Security.Permissions;

namespace D3DHost
{
 public partial class MainWindow : Window
 {
 public MainWindow()
 {
 InitializeComponent();

 // Set up the initial state for the D3DImage.
 HRESULT.Check(SetSize(512, 512));
 HRESULT.Check(SetAlpha(false));
 HRESULT.Check(SetNumDesiredSamples(4));

 //
 // Optional: Subscribing to the IsFrontBufferAvailableChanged event.
 //
 // If you don't render every frame (e.g. you only render in
 // reaction to a button click), you should subscribe to the
 // IsFrontBufferAvailableChanged event to be notified when rendered content
 // is no longer being displayed. This event also notifies you when
 // the D3DImage is capable of being displayed again.

 // For example, in the button click case, if you don't render again when
 // the IsFrontBufferAvailable property is set to true, your
 // D3DImage won't display anything until the next button click.
 //
 // Because this application renders every frame, there is no need to
 // handle the IsFrontBufferAvailableChanged event.
 //
 CompositionTarget.Rendering += new EventHandler(CompositionTarget_Rendering);

 //
 // Optional: Multi-adapter optimization
 //
 // The surface is created initially on a particular adapter.
 // If the WPF window is dragged to another adapter, WPF
 // ensures that the D3DImage still shows up on the new
 // adapter.
 //
 // This process is slow on Windows XP.
 //
 // Performance is better on Vista with a 9Ex device. It's only
 // slow when the D3DImage crosses a video-card boundary.
 //
 // To work around this issue, you can move your surface when
 // the D3DImage is displayed on another adapter. To
 // determine when that is the case, transform a point on the
 // D3DImage into screen space and find out which adapter
 // contains that screen space point.
 //
 // When your D3DImage straddles two adapters, nothing
 // can be done, because one will be updating slowly.
 //
 _adapterTimer = new DispatcherTimer();
 _adapterTimer.Tick += new EventHandler(AdapterTimer_Tick);
 _adapterTimer.Interval = new TimeSpan(0, 0, 0, 0, 500);
 _adapterTimer.Start();
 }
 }
}
```

```

//
// Optional: Surface resizing
//
// The D3DImage is scaled when WPF renders it at a size
// different from the natural size of the surface. If the
// D3DImage is scaled up significantly, image quality
// degrades.
//
// To avoid this, you can either create a very large
// texture initially, or you can create new surfaces as
// the size changes. Below is a very simple example of
// how to do the latter.
//
// By creating a timer at Render priority, you are guaranteed
// that new surfaces are created while the element
// is still being arranged. A 200 ms interval gives
// a good balance between image quality and performance.
// You must be careful not to create new surfaces too
// frequently. Frequently allocating a new surface may
// fragment or exhaust video memory. This issue is more
// significant on XDDM than it is on WDDM, because WDDM
// can page out video memory.
//
// Another approach is deriving from the Image class,
// participating in layout by overriding the ArrangeOverride method, and
// updating size in the overridden method. Performance will degrade
// if you resize too frequently.
//
// Blurry D3DImages can still occur due to subpixel
// alignments.
//
_sizeTimer = new DispatcherTimer(DispatcherPriority.Render);
_sizeTimer.Tick += new EventHandler(SizeTimer_Tick);
_sizeTimer.Interval = new TimeSpan(0, 0, 0, 0, 200);
_sizeTimer.Start();
}

~MainWindow()
{
 Destroy();
}

void AdapterTimer_Tick(object sender, EventArgs e)
{
 POINT p = new POINT(imgelt.PointToScreen(new Point(0, 0)));

 HRESULT.Check(SetAdapter(p));
}

void SizeTimer_Tick(object sender, EventArgs e)
{
 // The following code does not account for RenderTransforms.
 // To handle that case, you must transform up to the root and
 // check the size there.

 // Given that the D3DImage is at 96.0 DPI, its Width and Height
 // properties will always be integers. ActualWidth/Height
 // may not be integers, so they are cast to integers.
 uint actualWidth = (uint)imgelt.ActualWidth;
 uint actualHeight = (uint)imgelt.ActualHeight;
 if ((actualWidth > 0 && actualHeight > 0) &&
 (actualWidth != (uint)d3dimg.Width || actualHeight != (uint)d3dimg.Height))
 {
 HRESULT.Check(SetSize(actualWidth, actualHeight));
 }
}

void CompositionTarget_Rendering(object sender, EventArgs e)

```

```

 {
 RenderingEventArgs args = (RenderingEventArgs)e;

 // It's possible for Rendering to call back twice in the same frame
 // so only render when we haven't already rendered in this frame.
 if (d3dimg.IsFrontBufferAvailable && _lastRender != args.RenderingTime)
 {
 IntPtr pSurface = IntPtr.Zero;
 HRESULT.Check(GetBackBufferNoRef(out pSurface));
 if (pSurface != IntPtr.Zero)
 {
 d3dimg.Lock();
 // Repeatedly calling SetBackBuffer with the same IntPtr is
 // a no-op. There is no performance penalty.
 d3dimg.SetBackBuffer(D3DResourceType.IDirect3DSurface9, pSurface);
 HRESULT.Check(Render());
 d3dimg.AddDirtyRect(new Int32Rect(0, 0, d3dimg.PixelWidth, d3dimg.PixelHeight));
 d3dimg.Unlock();

 _lastRender = args.RenderingTime;
 }
 }
 }

 DispatcherTimer _sizeTimer;
 DispatcherTimer _adapterTimer;
 TimeSpan _lastRender;

 // Import the methods exported by the unmanaged Direct3D content.

 [DllImport("D3DCode.dll")]
 static extern int GetBackBufferNoRef(out IntPtr pSurface);

 [DllImport("D3DCode.dll")]
 static extern int SetSize(uint width, uint height);

 [DllImport("D3DCode.dll")]
 static extern int SetAlpha(bool useAlpha);

 [DllImport("D3DCode.dll")]
 static extern int SetNumDesiredSamples(uint numSamples);

 [StructLayout(LayoutKind.Sequential)]
 struct POINT
 {
 public POINT(Point p)
 {
 x = (int)p.X;
 y = (int)p.Y;
 }

 public int x;
 public int y;
 }

 [DllImport("D3DCode.dll")]
 static extern int SetAdapter(POINT screenSpacePoint);

 [DllImport("D3DCode.dll")]
 static extern int Render();

 [DllImport("D3DCode.dll")]
 static extern void Destroy();
}

public static class HRESULT
{
 [SecurityPermissionAttribute(SecurityAction.Demand, Flags =
 SecurityPermissionFlag.UnmanagedCode)]

```

```
 public static void Check(int hr)
 {
 Marshal.ThrowExceptionForHR(hr);
 }
}
```

## Hosting del contenuto di Direct3D9

Infine, usare la classe [D3DImage](#) per ospitare il contenuto di Direct3D9.

### Per ospitare il contenuto di Direct3D9

1. In MainWindow.xaml sostituire il codice XAML generato automaticamente con il codice XAML seguente.

```
<Window x:Class="D3DHost.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:i="clr-namespace:System.Windows.Interop;assembly=PresentationCore"
Title="MainWindow" Height="300" Width="300" Background="PaleGoldenrod">
<Grid>
 <Image x:Name="imgelt">
 <Image.Source>
 <i:D3DImage x:Name="d3dimg" />
 </Image.Source>
 </Image>
</Grid>
</Window>
```

2. Compilazione del progetto.
3. Copiare la DLL che contiene il contenuto di Direct3D9 nella cartella bin/debug.
4. Premere F5 per eseguire il progetto.

Il contenuto di Direct3D9 viene visualizzato all'interno dell'applicazione WPF.

## Vedere anche

- [D3DImage](#)
- [Considerazioni sulle prestazioni per l'interoperabilità fra Direct3D9 e WPF](#)

# Prestazioni

23/10/2019 • 2 minutes to read • [Edit Online](#)

Per ottenere prestazioni ottimali delle applicazioni richiede un'organizzazione preventiva nella progettazione dell'applicazione e la comprensione delle procedure consigliate per Windows Presentation Foundation (WPF) lo sviluppo di applicazioni. Negli argomenti di questa sezione vengono fornite informazioni aggiuntive sulla compilazione ad alte prestazioni WPF applicazioni.

## In questa sezione

[Livelli di rendering della grafica](#)

[Ottimizzazione delle prestazioni di applicazioni WPF](#)

[Procedura dettagliata: La memorizzazione nella cache i dati dell'applicazione in un'applicazione WPF](#)

## Riferimenti

[RenderCapability](#)

[Tier](#)

[PresentationTraceSources](#)

## Vedere anche

- [Layout](#)
- [Suggerimenti sulle animazioni](#)

# Livelli di rendering della grafica

11/12/2019 • 13 minutes to read • [Edit Online](#)

Un livello di rendering definisce un livello di prestazioni e funzionalità hardware grafiche per un dispositivo che esegue un'applicazione WPF.

## Hardware grafico

Le funzionalità dell'hardware grafico che hanno effetto sui livelli di rendering sono:

- **RAM video** La quantità di memoria video sull'hardware grafico determina le dimensioni e il numero di buffer che possono essere usati per la composizione della grafica.
- **Pixel shader** Un pixel shader è una funzione di elaborazione grafica che calcola gli effetti sui singoli pixel. A seconda della risoluzione della grafica visualizzata, potrebbe essere necessario elaborare diversi milioni di pixel per ogni fotogramma visualizzato.
- **Vertex shader** Un vertex shader è una funzione di elaborazione grafica che esegue operazioni matematiche sui dati dei vertici dell'oggetto.
- **Supporto per più trame** Il supporto per più trame è la possibilità di applicare due o più trame distinte durante un'operazione di fusione su un oggetto grafico 3D. Il grado di supporto per più trame è determinato dal numero di unità a più trame nell'hardware grafico.

## Definizioni dei livelli di rendering

Le funzionalità dell'hardware grafico determinano la capacità di rendering di un'applicazione WPF. Il sistema WPF definisce tre livelli di rendering:

- **Livello di rendering 0** Nessuna accelerazione hardware grafico. Tutte le funzionalità grafiche usano l'accelerazione software. Il livello della versione di DirectX è inferiore alla versione 9,0.
- **Livello di rendering 1** Alcune funzionalità grafiche usano l'accelerazione hardware grafico. Il livello della versione di DirectX è maggiore o uguale alla versione 9,0.
- **Livello di rendering 2** La maggior parte delle funzionalità grafiche usa l'accelerazione hardware grafico. Il livello della versione di DirectX è maggiore o uguale alla versione 9,0.

La proprietà `RenderCapability.Tier` consente di recuperare il livello di rendering in fase di esecuzione dell'applicazione. Il livello di rendering consente di determinare se il dispositivo supporta determinate funzionalità grafiche con accelerazione hardware. L'applicazione può quindi usare percorsi di codice diversi in fase di esecuzione a seconda del livello di rendering supportato dal dispositivo.

### Livello di rendering 0

Il valore 0 del livello di rendering indica che l'accelerazione hardware grafico non è disponibile per l'applicazione sul dispositivo. A questo livello, si deve presupporre che il rendering tutta la grafica verrà eseguito dal software senza accelerazione hardware. Questa funzionalità del livello corrisponde a una versione di DirectX inferiore a 9,0.

### Livello di rendering 1 e livello di rendering 2

#### NOTE

A partire da .NET Framework 4, il livello di rendering 1 è stato ridefinito in modo da includere solo hardware grafico che supporta DirectX 9,0 o versione successiva. L'hardware grafico che supporta DirectX 7 o 8 è ora definito come livello di rendering 0.

Il valore 1 o 2 del livello di rendering indica che la maggior parte delle funzionalità grafiche di WPF userà l'accelerazione hardware se le risorse di sistema necessarie sono disponibili e non sono state esaurite.

Corrisponde a una versione di DirectX maggiore o uguale a 9,0.

La tabella seguente illustra le differenze dei requisiti di hardware grafico tra il livello di rendering 1 e livello di rendering 2:

CARATTERISTICA	LIVELLO 1	LIVELLO 2
Versione DirectX	Deve essere superiore o uguale alla 9,0.	Deve essere superiore o uguale alla 9,0.
RAM video	Deve essere superiore o uguale a 60 MB.	Deve essere superiore o uguale a 120 MB.
Pixel shader	Il livello della versione deve essere superiore o uguale alla 2,0.	Il livello della versione deve essere superiore o uguale alla 2,0.
Vertex shader	Nessun requisito.	Il livello della versione deve essere superiore o uguale alla 2,0.
Unità a più trame	Nessun requisito.	Il numero di unità deve essere superiore o uguale a 4.

Le funzionalità e capacità seguenti sono con accelerazione hardware per il livello di rendering 1 e livello di rendering 2:

CARATTERISTICA	NOTE
Rendering 2D	È supportata la maggior parte del rendering 2D.
Rasterizzazione 3D	È supportata la maggior parte delle rasterizzazioni 3D.
Filtro anisotropico 3D	WPF prova a usare il filtro anisotropico quando viene eseguito il rendering del contenuto 3D. Il filtro anisotropico consente di migliorare la qualità delle trame di un'immagine su superfici lontane e molto inclinate rispetto alla fotocamera.
Mapping MIP 3D	WPF prova a usare il mapping MIP quando viene eseguito il rendering del contenuto 3D. Il mapping MIP migliora la qualità del rendering della trama quando una trama occupa un campo di visualizzazione più piccolo in una <a href="#">Viewport3D</a> .
Sfumature radiali	Sebbene sia supportato, evitare l'uso di <a href="#">RadialGradientBrush</a> su oggetti di grandi dimensioni.
Calcoli per l'illuminazione 3D	WPF esegue l'illuminazione per vertice, in cui l'intensità della luce deve essere calcolata in ogni vertice per ogni materiale applicato a un mesh.

CARATTERISTICA	NOTE
Rendering del testo	Il rendering dei tipi di carattere a livello di sub-pixel usa i pixel shader disponibili nell'hardware grafico.

Le funzionalità e capacità seguenti sono con accelerazione hardware solo per il livello di rendering 2:

CARATTERISTICA	NOTE
Anti-aliasing 3D	l'anti-aliasing 3D è supportato solo nei sistemi operativi che supportano Windows Display Driver Model (WDDM), ad esempio Windows Vista e Windows 7.

Le funzionalità e capacità seguenti sono **senza** accelerazione hardware:

CARATTERISTICA	NOTE
Contenuti stampati	Il rendering di tutto il contenuto stampato viene eseguito con la pipeline software WPF.
Contenuto rasterizzato che usa <a href="#">RenderTargetBitmap</a>	Qualsiasi contenuto di cui viene eseguito il rendering tramite il metodo <a href="#">Render</a> di <a href="#">RenderTargetBitmap</a> .
Contenuto affiancato che usa <a href="#">TileBrush</a>	Qualsiasi contenuto affiancato in cui la proprietà <a href="#">TileMode</a> del <a href="#">TileBrush</a> è impostata su <a href="#">Tile</a> .
Superfici che superano le dimensioni massime della trama dell'hardware grafico	Per la maggior parte dell'hardware grafico, le superfici di grandi dimensioni sono pari a 2048x2048 o 4096x4096 pixel.
Qualsiasi operazione il cui requisito per la RAM video supera la memoria dell'hardware grafico	È possibile monitorare l'utilizzo della RAM video dell'applicazione usando lo strumento Perforator incluso nella <a href="#">famiglia di prodotti per l'analisi delle prestazioni WPF</a> in Windows SDK.
Finestre a livelli	<p>Le finestre a livelli consentono alle applicazioni WPF di eseguire il rendering del contenuto sullo schermo in una finestra non rettangolare. Nei sistemi operativi che supportano Windows Display Driver Model (WDDM), ad esempio Windows Vista e Windows 7, le finestre sovrapposte sono con accelerazione hardware. In altri sistemi, ad esempio Windows XP, il rendering delle finestre sovrapposte viene eseguito dal software senza accelerazione hardware.</p> <p>È possibile abilitare le finestre sovrapposte in WPF impostando le proprietà <a href="#">Window</a> seguenti:</p> <ul style="list-style-type: none"> <li>- <a href="#">WindowStyle</a> = <a href="#">None</a></li> <li>- <a href="#">AllowsTransparency</a> = <a href="#">true</a></li> <li>- <a href="#">Background</a> = <a href="#">Transparent</a></li> </ul>

## Altre risorse

Le risorse seguenti consentono di analizzare le caratteristiche delle prestazioni dell'applicazione WPF.

### Impostazioni del Registro di sistema per il rendering della grafica

WPF fornisce quattro impostazioni del Registro di sistema per controllare il rendering WPF:

IMPOSTAZIONE DI	DESCRIZIONE
<b>Opzione di disabilitazione dell'accelerazione hardware</b>	Specifica se l'accelerazione hardware deve essere abilitata.
<b>Valore massimo di multicampionamento</b>	Specifica il grado di campionamento multiplo per l'anti-aliasing del contenuto 3D.
<b>Impostazione Data driver video necessaria</b>	Specifica se il sistema disabilita l'accelerazione hardware per i driver rilasciati prima di novembre 2004.
<b>Opzione per l'uso di unità di rasterizzazione dei riferimenti</b>	Specifica se WPF deve usare l'unità di rasterizzazione dei riferimenti.

A queste impostazioni è possibile accedere tramite qualsiasi utilità di configurazione esterna che possa fare riferimento alle impostazioni del Registro di sistema di WPF. È possibile creare o modificare queste impostazioni anche accedendo direttamente ai valori usando l'editor del registro di sistema di Windows. Per altre informazioni, vedere [Impostazioni del Registro di sistema per il rendering della grafica](#).

### Strumenti per la profilatura delle prestazioni WPF

WPF include una suite di strumenti per la profilatura delle prestazioni che consentono di analizzare il comportamento dell'applicazione in fase di esecuzione e di determinare i tipi di ottimizzazioni delle prestazioni che è possibile applicare. Nella tabella seguente sono elencati gli strumenti di profilatura delle prestazioni inclusi nello strumento Windows SDK, WPF Performance Suite:

STRUMENTO	DESCRIZIONE
Perforator	Da usare per l'analisi del comportamento di rendering.
Visual Profiler	Da usare per la profilatura dell'uso dei servizi WPF, come gestione del layout e degli eventi, per gli elementi nella struttura ad albero visuale.

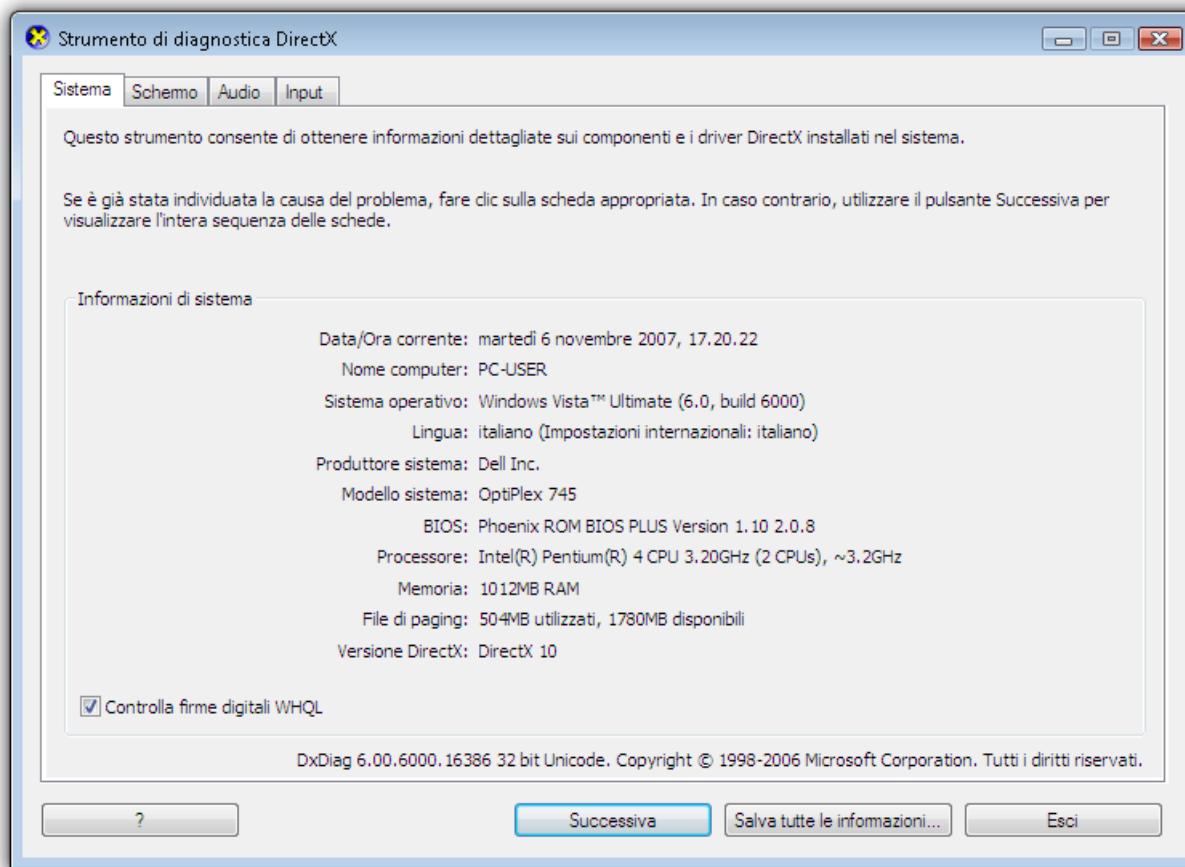
WPF Performance Suite offre una visualizzazione grafica avanzata dei dati sulle prestazioni. Per altre informazioni sugli strumenti per le prestazioni WPF, vedere [Famiglia di prodotti per l'analisi delle prestazioni WPF](#).

### Strumento di diagnostica DirectX

Lo strumento di diagnostica DirectX, Dxdiag. exe, è progettato per semplificare la risoluzione dei problemi correlati a DirectX. La cartella di installazione predefinita per lo strumento di diagnostica DirectX è la seguente:

`~\Windows\System32`

Quando si esegue lo strumento di diagnostica DirectX, la finestra principale contiene un set di schede che consentono di visualizzare e diagnosticare le informazioni correlate a DirectX. Ad esempio, la scheda **sistema** fornisce informazioni di sistema sul computer e specifica la versione di DirectX installata nel computer.



Finestra principale dello strumento di diagnostica DirectX

## Vedere anche

- [RenderCapability](#)
- [RenderOptions](#)
- [Ottimizzazione delle prestazioni di applicazioni WPF](#)
- [Famiglia di prodotti per l'analisi delle prestazioni WPF](#)
- [Impostazioni del Registro di sistema per il rendering della grafica](#)
- [Suggerimenti sulle animazioni](#)

# Ottimizzazione delle prestazioni di applicazioni WPF

03/02/2020 • 2 minutes to read • [Edit Online](#)

Questa sezione è destinata a un riferimento per Windows Presentation Foundation (WPF) sviluppatori di applicazioni che stanno cercando modi per migliorare le prestazioni delle applicazioni. Per gli sviluppatori che non hanno familiarità con il Framework Microsoft .NET e WPF, è necessario acquisire familiarità con entrambe le piattaforme. Questa sezione presuppone la conoscenza del funzionamento di entrambi e viene scritta per i programmati che conoscono già abbastanza per far funzionare le proprie applicazioni.

## NOTE

I dati sulle prestazioni forniti in questa sezione si basano su WPF applicazioni in esecuzione su un PC a 2,8 GHz con RAM 512 e una scheda grafica ATI Radeon 9700.

## Contenuto della sezione

[Pianificazione delle prestazioni dell'applicazione](#)

[Sfruttare appieno l'hardware](#)

[Ottimizzazione delle prestazioni: layout e progettazione](#)

[Grafica bidimensionale e creazione di immagini](#)

[Comportamento dell'oggetto](#)

[Risorse di applicazioni](#)

[Testo](#)

[Data binding](#)

[Controlli](#)

[Altri suggerimenti relativi alle prestazioni](#)

[Tempo di avvio delle applicazioni](#)

## Vedere anche

- [RenderOptions](#)
- [RenderCapability](#)
- [Livelli di rendering della grafica](#)
- [Cenni preliminari sul rendering della grafica WPF](#)
- [Layout](#)
- [Strutture ad albero in WPF](#)
- [Cenni preliminari sugli oggetti Drawing](#)
- [Uso degli oggetti DrawingVisual](#)
- [Cenni preliminari sulle proprietà di dipendenza](#)
- [Cenni preliminari sugli oggetti Freezable](#)
- [Risorse XAML](#)

- Documenti in WPF
- Disegno di testo formattato
- Funzionalità tipografiche di WPF
- Panoramica sul data binding
- Cenni preliminari sulla navigazione
- Suggerimenti sulle animazioni
- Procedura dettagliata: Memorizzazione dei dati di un'applicazione nella cache di un'applicazione WPF

# Pianificazione delle prestazioni dell'applicazione

23/10/2019 • 5 minutes to read • [Edit Online](#)

La possibilità di raggiungere gli obiettivi di prestazioni dipende dalla capacità di sviluppare una strategia di prestazioni. La pianificazione è la prima fase nello sviluppo di qualsiasi prodotto. Questo argomento descrive alcune regole molto semplici per sviluppare una strategia di prestazioni ottimali.

## Pensare in termini di scenari

Scenari consentono di concentrarsi sui componenti critici dell'applicazione. Scenari a livello generale sono derivati dai clienti, nonché i prodotti competitivi. Sempre studiare i tuoi clienti e scoprire ciò che rende li entusiasti al prodotto e i prodotti della concorrenza. Commenti e suggerimenti dei clienti possono aiutare a determinare uno scenario principale dell'applicazione. Ad esempio, se si sta sviluppando un componente che verrà usato all'avvio, è probabile che il componente verrà chiamato una sola volta, all'avvio dell'applicazione. Tempo di avvio diventa il principale scenario. Altri esempi di scenari chiave potrebbero essere la frequenza dei fotogrammi desiderati per le sequenze di animazione, o il working set massimo consentito per l'applicazione.

## Definire gli obiettivi

Obiettivi di aiutano a determinare se un'applicazione sta effettuando più veloce o lenta. È consigliabile definire obiettivi per tutti gli scenari. Tutti gli obiettivi di prestazioni che definiscono devono basarsi sulle aspettative dei clienti. Potrebbe essere difficile per le prestazioni dei set del ciclo di obiettivi sin dall'inizio per lo sviluppo di applicazioni, se sono presenti problemi non risolti molti. Tuttavia, è preferibile definire un obiettivo iniziale e rivederla successiva rispetto a non a essere un obiettivo del tutto.

## Comprendere la piattaforma in uso

Mantenere sempre il ciclo di misurazione, analisi, perfezionando/correggere durante il ciclo di sviluppo dell'applicazione. A partire dall'inizio alla fine del ciclo di sviluppo, è necessario misurare le prestazioni dell'applicazione in un ambiente stabile e affidabile. È consigliabile evitare la variabilità causata da fattori esterni. Ad esempio, durante il test delle prestazioni, si dovrebbe disabilitare antivirus o qualsiasi altro aggiornamento automatico, ad esempio SMS, in modo da non influenzare i risultati del test. Dopo aver misurato le prestazioni dell'applicazione, è necessario identificare le modifiche che comporterà i miglioramenti più significativi. Dopo aver modificato l'applicazione, avviare nuovamente il ciclo.

## Creare un processo iterativo di ottimizzazione delle prestazioni

È necessario conoscere il costo relativo di ogni funzione che si userà. Ad esempio, l'uso della reflection in Microsoft .NET Framework è in genere prestazioni elevate in termini di risorse di calcolo, in modo che si desidera utilizzarlo con cautela. Ciò non significa evitare l'uso della reflection, solo che è necessario prestare attenzione a bilanciare i requisiti di prestazioni dell'applicazione con le esigenze di prestazioni delle funzionalità usate.

## Compilare verso ricchezza con interfaccia grafica

Una tecnica fondamentale per la creazione di un approccio scalabile per il raggiungimento di WPF le prestazioni dell'applicazione consiste nel compilare verso la ricchezza di con interfaccia grafica e complessità. Iniziare sempre con l'uso minimo con utilizzo intensivo delle risorse di prestazioni per raggiungere gli obiettivi di scenario. Una volta raggiungere questi obiettivi, compilare verso ricchezza grafico usando più funzionalità con utilizzo intensivo di prestazioni, sempre tenendo presenti gli obiettivi dello scenario. Tenere presente che WPF è una piattaforma

molto avanzata e offre numerose funzionalità grafiche. Uso delle prestazioni a elevato utilizzo di funzionalità può influire negativamente sulle prestazioni dell'applicazione complessiva.

WPF i controlli sono intrinsecamente estensibili, consentendo la diffusa personalizzazione dell'aspetto, mentre non modifica il comportamento di controllo. Grazie all'uso di stili, modelli di dati e modelli di controllo, è possibile creare e sviluppare in modo incrementale un personalizzabile interfaccia utente che si adatta ai requisiti di prestazioni.

## Vedere anche

- [Ottimizzazione delle prestazioni di applicazioni WPF](#)
- [Sfruttare appieno l'hardware](#)
- [Ottimizzazione delle prestazioni: layout e progettazione](#)
- [Grafica bidimensionale e creazione di immagini](#)
- [Comportamento dell'oggetto](#)
- [Risorse di applicazioni](#)
- [per](#)
- [Data binding](#)
- [Altri suggerimenti relativi alle prestazioni](#)

# Ottimizzazione delle prestazioni: Sfruttare appieno l'hardware

23/10/2019 • 7 minutes to read • [Edit Online](#)

L'architettura interna di WPF dispone di due pipeline di rendering, hardware e software. In questo argomento vengono fornite informazioni su queste pipeline di rendering che consentono di prendere decisioni in merito alle ottimizzazioni delle prestazioni delle applicazioni.

## Pipeline di rendering hardware

Uno dei fattori più importanti per determinare le prestazioni è che è associato al rendering, maggiore è il numero di pixel di cui è necessario eseguire il rendering, maggiore è il costo delle prestazioni. Tuttavia, maggiore è il rendering che è possibile scaricare nell'unità di elaborazione grafica (GPU), maggiori sono i vantaggi in termini di prestazioni che è possibile ottenere. La WPF pipeline di rendering hardware dell'applicazione sfrutta appieno le funzionalità di Microsoft DirectX su hardware che supporta almeno Microsoft DirectX versione 7,0. Ulteriori ottimizzazioni possono essere ottenute con hardware che supporta le funzionalità di Microsoft DirectX versione 7,0 e PixelShader 2,0 +.

## Pipeline di rendering software

La WPF pipeline di rendering software è interamente associata alla CPU. WPF sfrutta i set di istruzioni SSE e SSE2 nella CPU per implementare un'rasterizzatore software ottimizzato e con funzionalità complete. Il fallback al software è facile ogni volta che non è possibile eseguire il rendering della funzionalità dell'applicazione utilizzando la pipeline di rendering hardware.

Il problema di prestazioni più grande che si verificherà quando il rendering in modalità software è correlato alla velocità di riempimento, definita come il numero di pixel di cui si esegue il rendering. Se si è interessati alle prestazioni in modalità di rendering software, provare a ridurre al minimo il numero di volte in cui un pixel viene ridisegnato. Se, ad esempio, si dispone di un'applicazione con uno sfondo blu, che quindi esegue il rendering di un'immagine leggermente trasparente su di essa, si eseguirà il rendering di tutti i pixel nell'applicazione due volte. Di conseguenza, il rendering dell'applicazione con l'immagine verrà eseguito il doppio rispetto a quando si dispone solo dello sfondo blu.

### Livelli di rendering della grafica

Potrebbe essere molto difficile prevedere la configurazione hardware in cui verrà eseguita l'applicazione. Tuttavia, potrebbe essere opportuno prendere in considerazione una progettazione che consenta all'applicazione di cambiare facilmente le funzionalità quando vengono eseguite su hardware diverso, in modo che sia possibile sfruttare appieno tutte le diverse configurazioni hardware.

A tale scopo, WPF fornisce la funzionalità per determinare la funzionalità grafica di un sistema in fase di esecuzione. La funzionalità grafica è determinata dalla categorizzazione della scheda video come uno dei tre livelli di funzionalità di rendering. WPF espone un'API che consente a un'applicazione di eseguire query sul livello di funzionalità di rendering. L'applicazione può quindi usare percorsi di codice diversi in fase di esecuzione a seconda del livello di rendering supportato dall'hardware.

Le funzionalità dell'hardware grafico che hanno effetto sui livelli di rendering sono:

- **RAM video** La quantità di memoria video sull'hardware grafico determina le dimensioni e il numero di buffer che possono essere usati per la composizione della grafica.

- **Pixel shader** Un pixel shader è una funzione di elaborazione grafica che calcola gli effetti sui singoli pixel. A seconda della risoluzione della grafica visualizzata, potrebbe essere necessario elaborare diversi milioni di pixel per ogni fotogramma visualizzato.
- **Vertex shader** Un vertex shader è una funzione di elaborazione grafica che esegue operazioni matematiche sui dati dei vertici dell'oggetto.
- **Supporto per più trame** Il supporto per più trame è la possibilità di applicare due o più trame distinte durante un'operazione di fusione su un oggetto grafico 3D. Il grado di supporto per più trame è determinato dal numero di unità a più trame nell'hardware grafico.

Le funzionalità pixel shader, vertex shader e multitexture vengono usate per definire livelli di versione DirectX specifici, che, a loro volta, vengono usati per definire i diversi livelli di rendering in WPF.

Le funzionalità dell'hardware grafico determinano la capacità di rendering di un'applicazione WPF. Il sistema WPF definisce tre livelli di rendering:

- **Livello di rendering 0** Nessuna accelerazione hardware grafico. Il livello della versione di DirectX è inferiore alla versione 7,0.
- **Livello di rendering 1** Accelerazione hardware grafica parziale. Il livello della versione di DirectX è maggiore o uguale alla versione 7,0 e **inferiore** alla versione 9,0.
- **Livello di rendering 2** La maggior parte delle funzionalità grafiche usa l'accelerazione hardware grafico. Il livello della versione di DirectX è maggiore o uguale alla versione 9,0.

Per altre informazioni sui WPF livelli di rendering, vedere [livelli di rendering della grafica](#).

## Vedere anche

- [Ottimizzazione delle prestazioni di applicazioni WPF](#)
- [Pianificazione delle prestazioni dell'applicazione](#)
- [Ottimizzazione delle prestazioni: layout e progettazione](#)
- [Grafica bidimensionale e creazione di immagini](#)
- [Comportamento dell'oggetto](#)
- [Risorse di applicazioni](#)
- [Text](#)
- [Data binding](#)
- [Altri suggerimenti relativi alle prestazioni](#)

# Ottimizzazione delle prestazioni: Layout e progettazione

23/10/2019 • 7 minutes to read • [Edit Online](#)

La progettazione dell'applicazione WPF può influire sulle relative prestazioni, poiché crea un sovraccarico non necessario durante il calcolo del layout e la convalida dei riferimenti agli oggetti. La costruzione di oggetti, soprattutto in fase di runtime, può influire sulle caratteristiche di prestazioni dell'applicazione.

Questo argomento offre utili suggerimenti per il miglioramento delle prestazioni in queste aree.

## Layout

Il termine "passaggio di layout" descrive il processo di misurazione e disposizione dei membri di un **Panel**-insieme di elementi figlio e di disegno sullo schermo dell'oggetto derivato. Il passaggio di layout è un processo molto complesso dal punto di vista matematico: più alto è il numero di elementi figlio presenti nella raccolta, maggiore è il numero di calcoli necessari. Ad esempio, ogni volta che un elemento figlio **UIElement** oggetto nella raccolta cambia posizione, ha la possibilità di attivare un nuovo passaggio dal sistema di layout. Data la stretta correlazione tra le caratteristiche dell'oggetto e il comportamento del layout, è importante capire i tipi di eventi che possono richiamare il sistema di layout. Le prestazioni dell'applicazione risulteranno migliori se si riuscirà a ridurre al massimo le chiamate non necessarie al passaggio di layout.

Per ogni membro figlio di una raccolta vengono completati due passaggi: un passaggio di misurazione e uno di disposizione. Ogni oggetto figlio offre la propria implementazione sottoposta a override dei **Measure** e **Arrange** metodi per fornire il proprio comportamento di layout specifico. Nella forma più semplice, il layout è un sistema ricorsivo che fa sì che un elemento venga ridimensionato, posizionato e disegnato sullo schermo.

- Un elemento figlio **UIElement** oggetto inizia il processo di layout con la misurazione delle relative proprietà principali.
- L'oggetto **FrameworkElement** le proprietà correlate alle dimensioni, ad esempio **Width**, **Height**, e **Margin**, vengono valutati.
- **Panel**-viene applicata logica specifica, ad esempio la **Dock** proprietà del **DockPanel**, o il **Orientation** proprietà del **StackPanel**.
- Il contenuto viene disposto, o posizionato, dopo la misurazione di tutti gli oggetti figlio.
- La raccolta di oggetti figlio viene disegnata sullo schermo.

Il passaggio di layout viene nuovamente richiamato qualora si verifichi una delle azioni seguenti:

- Un oggetto figlio viene aggiunto alla raccolta.
- Oggetto **LayoutTransform** viene applicato all'oggetto figlio.
- Il **UpdateLayout** viene chiamato per l'oggetto figlio.
- Se viene apportata una modifica al valore di una proprietà di dipendenza contrassegnata con metadati che incidono sul passaggio di misurazione o disposizione.

### Usare il pannello più efficiente, se possibile

La complessità del processo di layout dipende direttamente dal comportamento di layout di **Panel**-degli elementi è utilizzare derivati. Ad esempio, un **Grid** oppure **StackPanel** controllo offre molte più funzionalità rispetto a un

[Canvas](#) controllo. All'aumento delle funzionalità corrisponde, tuttavia, un maggiore dispendio in termini di prestazioni. Tuttavia, se la funzionalità non è necessario che un [Grid](#) fornisce controllo, è necessario usare le alternative meno costose, ad esempio un [Canvas](#) o un pannello personalizzato.

Per altre informazioni, vedere [Cenni preliminari sugli elementi Panel](#).

### Aggiornare anziché sostituire una proprietà RenderTransform

È possibile aggiornare una [Transform](#) anziché sostituirlo come valore di un [RenderTransform](#) proprietà. Questa possibilità può essere attuata soprattutto negli scenari con animazioni. Aggiornando un oggetto esistente [Transform](#), si evita di attivare un calcolo dell'oggetto layout non necessari.

### Compilare la struttura ad albero dall'alto in basso

Quando si aggiunge o si rimuove un nodo dall'albero logico, le convalide di proprietà vengono annullate sull'elemento padre e su tutti gli elementi figlio del nodo. Di conseguenza, è sempre consigliabile seguire un pattern di costruzione dall'alto in basso per evitare il costo di annullamenti di convalide non necessari su nodi già convalidati. La tabella seguente illustra la differenza in velocità di esecuzione tra la creazione di una struttura ad albero dall'alto in basso rispetto a basso in alto, in cui la struttura ad albero presenta 150 livelli di profondità con un unico [TextBlock](#) e [DockPanel](#) a ogni livello.

AZIONE	COMPILAZIONE DELLA STRUTTURA AD ALBERO (IN MS)	RENDERING: INCLUDE LA COMPILAZIONE DELLA STRUTTURA AD ALBERO (IN MS)
Dal basso in alto	366	454
Dall'alto in basso	11	96

L'esempio di codice seguente illustra come creare una struttura ad albero dall'alto in basso.

```
private void OnBuildTreeTopDown(object sender, RoutedEventArgs e)
{
 TextBlock textBlock = new TextBlock();
 textBlock.Text = "Default";

 DockPanel parentPanel = new DockPanel();
 DockPanel childPanel;

 myCanvas.Children.Add(parentPanel);
 myCanvas.Children.Add(textBlock);

 for (int i = 0; i < 150; i++)
 {
 textBlock = new TextBlock();
 textBlock.Text = "Default";
 parentPanel.Children.Add(textBlock);

 childPanel = new DockPanel();
 parentPanel.Children.Add(childPanel);
 parentPanel = childPanel;
 }
}
```

```
Private Sub OnBuildTreeTopDown(ByVal sender As Object, ByVal e As RoutedEventArgs)
 Dim textBlock As New TextBlock()
 textBlock.Text = "Default"

 Dim parentPanel As New DockPanel()
 Dim childPanel As DockPanel

 myCanvas.Children.Add(parentPanel)
 myCanvas.Children.Add(textBlock)

 For i As Integer = 0 To 149
 textBlock = New TextBlock()
 textBlock.Text = "Default"
 parentPanel.Children.Add(textBlock)

 childPanel = New DockPanel()
 parentPanel.Children.Add(childPanel)
 parentPanel = childPanel
 Next i
End Sub
```

Per altre informazioni sull'albero logico, vedere [Strutture ad albero in WPF](#).

## Vedere anche

- [Ottimizzazione delle prestazioni di applicazioni WPF](#)
- [Pianificazione delle prestazioni dell'applicazione](#)
- [Sfruttare appieno l'hardware](#)
- [Grafica bidimensionale e creazione di immagini](#)
- [Comportamento dell'oggetto](#)
- [Risorse di applicazioni](#)
- [per](#)
- [Data binding](#)
- [Altri suggerimenti relativi alle prestazioni](#)
- [Layout](#)

# Ottimizzazione delle prestazioni: Grafica 2D e creazione di immagini

23/10/2019 • 12 minutes to read • [Edit Online](#)

WPF offre un'ampia gamma di funzionalità di grafica 2D e creazione di immagini che possono essere ottimizzate in base ai requisiti dell'applicazione. Questo argomento fornisce informazioni sull'ottimizzazione delle prestazioni in queste aree.

## Disegno e forme

WPF fornisce oggetti [Shape](#) e per rappresentare il contenuto del disegno grafico. [Drawing](#) Tuttavia, [Drawing](#) gli oggetti sono costrutti più semplici [Shape](#) rispetto agli oggetti e offrono caratteristiche di prestazioni migliori.

Un [Shape](#) oggetto consente di disegnare una forma grafica sullo schermo. Poiché derivano dalla [FrameworkElement](#) classe, [Shape](#) gli oggetti possono essere utilizzati all'interno di pannelli e la maggior parte dei controlli.

WPF offre diversi livelli di accesso alla grafica e ai servizi di rendering. Al livello superiore, [Shape](#) gli oggetti sono facili da usare e offrono numerose funzionalità utili, ad esempio il layout e la gestione degli eventi. WPF offre anche una serie di oggetti Shape pronti all'uso. Tutti gli oggetti Shape ereditano [Shape](#) dalla classe. Gli oggetti Shape disponibili [Ellipse](#) includono [Line](#), [Path](#), [Polygon](#), [Polyline](#), e [Rectangle](#).

[Drawing](#) gli oggetti, d'altra parte, non derivano dalla [FrameworkElement](#) classe e forniscono un'implementazione più leggera per il rendering di forme, immagini e testo.

Sono disponibili quattro tipi di [Drawing](#) oggetti:

- [GeometryDrawing](#) Disegna una forma.
- [ImageDrawing](#) Disegna un'immagine.
- [GlyphRunDrawing](#) Disegna il testo.
- [DrawingGroup](#) Disegna altri disegni. Usare un gruppo di disegni per combinare altri disegni in un unico disegno composto.

L' [GeometryDrawing](#) oggetto viene utilizzato per eseguire il rendering del contenuto della geometria. La [Geometry](#) classe e le classi concrete che derivano da esso, [CombinedGeometry](#) ad [EllipseGeometry](#) esempio, [PathGeometry](#) e, forniscono un mezzo per il rendering di immagini 2D, oltre a fornire il supporto per l'hit testing e il ritaglio. Gli oggetti Geometry possono essere usati per definire, ad esempio, l'area di un controllo o l'area di ritaglio da applicare a un'immagine e possono essere semplici aree, quali rettangoli o cerchi, oppure aree composite create con due o più oggetti Geometry. È possibile creare aree geometriche più [PathSegment](#) complesse combinando oggetti derivati da, [BezierSegment](#), [ArcSegment](#) ad esempio [QuadraticBezierSegment](#), e.

Sulla superficie, la [Geometry](#) classe e la [Shape](#) classe sono molto simili. Entrambi vengono usati nel rendering di grafica 2D ed entrambi hanno classi concrete simili che derivano da esse, ad esempio [EllipseGeometry](#) e [Ellipse](#). Esistono tuttavia importanti differenze tra questi due set di classi. Per uno, la [Geometry](#) classe non dispone di alcune funzionalità [Shape](#) della classe, ad esempio la possibilità di disegnarla. Per disegnare un oggetto [Geometry](#), è necessario usare un'altra classe, ad esempio [DrawingContext](#), [Drawing](#) o [Path](#) (notare che [Path](#) è un oggetto [Shape](#)) per poter eseguire l'operazione di disegno. Le proprietà di rendering, tra cui il riempimento, il tratto e lo spessore del tratto, si trovano nella classe che consente di disegnare l'oggetto [Geometry](#), ma sono

contenute direttamente in un oggetto Shape. In altre parole, un oggetto Geometry definisce un'area, ad esempio un cerchio, mentre un oggetto Shape definisce un'area, il modo in cui questa viene riempita e delineata e partecipa al sistema di layout.

Poiché [Shape](#) gli oggetti derivano [FrameworkElement](#) dalla classe, l'uso di tali oggetti può aumentare significativamente il consumo di memoria nell'applicazione. Se non sono necessarie le [FrameworkElement](#) funzionalità per il contenuto grafico, provare a usare gli [Drawing](#) oggetti più semplici.

Per altre informazioni sugli [Drawing](#) oggetti, vedere [Cenni preliminari sugli oggetti Drawing](#).

## Oggetto StreamGeometry

L' [StreamGeometry](#) oggetto è un'alternativa semplice a [PathGeometry](#) per la creazione di forme geometriche. Usare un [StreamGeometry](#) oggetto quando è necessario descrivere una geometria complessa. [StreamGeometry](#) è ottimizzato per la [PathGeometry](#) gestione di molti oggetti e offre prestazioni migliori rispetto all' [PathGeometry](#) utilizzo di molti singoli oggetti.

Nell'esempio seguente viene usata la sintassi degli attributi per creare [StreamGeometry](#) un XAML triangolare in.

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
 <StackPanel>

 <Path Data="F0 M10,100 L100,100 100,50Z"
 StrokeThickness="1" Stroke="Black"/>

 </StackPanel>
</Page>
```

Per altre informazioni sugli [StreamGeometry](#) oggetti, vedere [creare una forma usando un StreamGeometry](#).

## Oggetti DrawingVisual

L' [DrawingVisual](#) oggetto è una classe di disegno semplificata utilizzata per il rendering di forme, immagini o testo. Questa classe è considerata semplice perché non offre la gestione di layout o eventi, con un conseguente aumento delle prestazioni. Per questo motivo, i disegni sono ideali per sfondi e ClipArt. Per altre informazioni, vedere [Uso degli oggetti DrawingVisual](#).

## Immagini

WPF la creazione di immagini offre un miglioramento significativo rispetto alle funzionalità di creazione di immagini nelle versioni precedenti di Windows. Le funzionalità per la creazione di immagini, come la visualizzazione di una bitmap o l'uso di un'immagine per un controllo comune, venivano gestite inizialmente dalle API (Application Programming Interface) Graphics Device Interface (GDI) o GDI+ di Microsoft Windows. Queste API offrivano funzionalità di base per la creazione di immagini, ma erano sprovviste di funzionalità come il supporto per l'estendibilità dei codec o per immagini ad alta fedeltà. Le API per la creazione di immagini di WPF sono state riprogettate per colmare le carenze di GDI e GDI+ e offrire un nuovo set di API per visualizzare e usare le immagini all'interno delle applicazioni.

Per ottenere prestazioni migliori durante l'uso di immagini, seguire questi consigli:

- Se l'applicazione richiede la visualizzazione di immagini di anteprima, creare una versione ridotta dell'immagine. Per impostazione predefinita, WPF carica l'immagine e la decodifica con le dimensioni originali. Se si vuole soltanto un'immagine di anteprima, WPF decodifica comunque l'immagine con le dimensioni originali e successivamente la riduce alle dimensioni di un'anteprima. Per evitare questo inutile sovraccarico, è possibile richiedere a WPF di decodificare l'immagine con le dimensioni dell'anteprima

oppure richiedere a WPF di caricare direttamente l'immagine di anteprima.

- Decodificare sempre l'immagine con le dimensioni desiderate e non con quelle predefinite. Richiedere quindi a WPF di eseguire la codifica secondo le dimensioni desiderate e non in base a quelle predefinite. In questo modo è possibile ridurre non solo il working set dell'applicazione ma anche la velocità di esecuzione.
- Se possibile, combinare le immagini in un'unica immagine, come una pellicola cinematografica composta da più immagini.
- Per altre informazioni, vedere [Panoramica della creazione dell'immagine](#).

### BitmapScalingMode

Quando si aggiunge un'animazione alla scala di una bitmap, è possibile che l'algoritmo di ricampionamento delle immagini di alta qualità determini un consumo di risorse di sistema tale da rallentare la frequenza dei fotogrammi, provocando lo stuttering delle animazioni. Impostando la **BitmapScalingMode** proprietà **RenderOptions** dell'oggetto su **LowQuality** è possibile creare un'animazione più uniforme durante il ridimensionamento di una bitmap. **LowQuality** mode indica WPF al motore di rendering di passare da un algoritmo ottimizzato per la qualità a un algoritmo ottimizzato per la velocità durante l'elaborazione di immagini.

Nell'esempio seguente viene illustrato come impostare **BitmapScalingMode** per un oggetto **Image**.

```
// Set the bitmap scaling mode for the image to render faster.
RenderOptions.SetBitmapScalingMode(MyImage, BitmapScalingMode.LowQuality);
```

```
' Set the bitmap scaling mode for the image to render faster.
RenderOptions.SetBitmapScalingMode(MyImage, BitmapScalingMode.LowQuality)
```

### CachingHint

Per impostazione predefinita WPF, non memorizza nella cache il contenuto **TileBrush** sottoposto a rendering **DrawingBrush** degli **VisualBrush** oggetti, ad esempio e. Negli scenari statici in cui non è stato modificato né il **TileBrush** contenuto né l'uso di nella scena, questo ha senso, poiché consente di conservare la memoria video. Non è molto utile quando un **TileBrush** oggetto con contenuto statico viene usato in modo non statico, ad esempio quando viene eseguito il mapping di un oggetto statico **VisualBrush DrawingBrush** o alla superficie di un oggetto 3D rotante. Il comportamento predefinito di WPF prevede di eseguire nuovamente il rendering dell'intero contenuto dell' **DrawingBrush** oggetto **VisualBrush** o per ogni frame, anche se il contenuto non è in corso di modifica.

Impostando la **CachingHint** proprietà **RenderOptions** dell'oggetto su **Cache** è possibile migliorare le prestazioni utilizzando le versioni memorizzate nella cache degli oggetti pennello affiancati.

I **CacheInvalidationThresholdMinimum** valori **CacheInvalidationThresholdMaximum** delle proprietà sono valori di dimensione relativa che determinano quando l' **TileBrush** oggetto deve essere rigenerato a causa di modifiche della scala. Se ad esempio si imposta la **CacheInvalidationThresholdMaximum** proprietà su 2,0, la cache **TileBrush** per deve essere rigenerata solo quando la dimensione supera il doppio della dimensione della cache corrente.

Nell'esempio seguente viene illustrato come utilizzare l'opzione relativa all'hint di memorizzazione **DrawingBrush** nella cache per un oggetto.

```
DrawingBrush drawingBrush = new DrawingBrush();

// Set the caching hint option for the brush.
RenderOptions.SetCachingHint(drawingBrush, CachingHint.Cache);

// Set the minimum and maximum relative sizes for regenerating the tiled brush.
// The tiled brush will be regenerated and re-cached when its size is
// 0.5x or 2x of the current cached size.
RenderOptions.SetCacheInvalidationThresholdMinimum(drawingBrush, 0.5);
RenderOptions.SetCacheInvalidationThresholdMaximum(drawingBrush, 2.0);
```

```
Dim drawingBrush As New DrawingBrush()

' Set the caching hint option for the brush.
RenderOptions.SetCachingHint(drawingBrush, CachingHint.Cache)

' Set the minimum and maximum relative sizes for regenerating the tiled brush.
' The tiled brush will be regenerated and re-cached when its size is
' 0.5x or 2x of the current cached size.
RenderOptions.SetCacheInvalidationThresholdMinimum(drawingBrush, 0.5)
RenderOptions.SetCacheInvalidationThresholdMaximum(drawingBrush, 2.0)
```

## Vedere anche

- [Ottimizzazione delle prestazioni di applicazioni WPF](#)
- [Pianificazione delle prestazioni dell'applicazione](#)
- [Sfruttare appieno l'hardware](#)
- [Ottimizzazione delle prestazioni: layout e progettazione](#)
- [Comportamento dell'oggetto](#)
- [Risorse di applicazioni](#)
- [Text](#)
- [Data binding](#)
- [Altri suggerimenti relativi alle prestazioni](#)
- [Suggerimenti sulle animazioni](#)

# Ottimizzazione delle prestazioni: comportamento degli oggetti

10/02/2020 • 14 minutes to read • [Edit Online](#)

La comprensione del comportamento intrinseco degli oggetti WPF consente di trovare più facilmente il compromesso ideale tra funzionalità e prestazioni.

## La mancata rimozione di gestori eventi dagli oggetti può mantenere gli oggetti attivi

Il delegato passato da un oggetto al relativo evento è di fatto un riferimento all'oggetto. I gestori eventi, quindi, possono mantenere gli oggetti attivi più a lungo del previsto. Quando si esegue la pulitura di un oggetto registrato per restare in ascolto di un evento dell'oggetto, è essenziale rimuovere il delegato prima di rilasciare l'oggetto. Mantenere attivi oggetti non necessari aumenta il consumo di memoria dell'applicazione, soprattutto se l'oggetto è la radice di un albero logico o di una struttura ad albero visuale.

In WPF è stato introdotto un modello di listener di eventi debole per eventi che possono rivelarsi utili in situazioni in cui è difficile tenere traccia delle relazioni di durata degli oggetti tra l'origine e il listener. Alcuni eventi WPF esistenti usano già questo modello, che può essere utile soprattutto per implementare oggetti con eventi personalizzati. Per informazioni dettagliate, vedere [Modelli di eventi deboli](#).

Sono disponibili vari strumenti, ad esempio il profiler CLR e il visualizzatore del working set, in grado di fornire informazioni sul consumo di memoria di un determinato processo. Il profiler CLR include alcune visualizzazioni del profilo di allocazione molto utili, tra cui un istogramma dei tipi allocati, grafici delle allocazioni e delle chiamate, una cronologia delle operazioni di Garbage Collection di varie generazioni e lo stato dell'heap gestito che ne deriva e una struttura ad albero delle chiamate che mostra le allocazioni per metodo e i caricamenti degli assembly. Per altre informazioni, vedere [Prestazioni](#).

## Proprietà e oggetti di dipendenza

In generale, l'accesso a una proprietà di dipendenza di un [DependencyObject](#) non è più lento dell'accesso a una proprietà CLR. Sebbene si verifichi un lieve sovraccarico delle prestazioni per l'impostazione di un valore di proprietà, il recupero di un valore è rapido quanto il recupero del valore da una proprietà CLR. Il sovraccarico delle prestazioni, infatti, è compensato dalla capacità delle proprietà di dipendenza di supportare funzionalità avanzate come il data binding, l'animazione, l'ereditarietà e l'applicazione di stili. Per altre informazioni, vedere [Panoramica sulle proprietà di dipendenza](#).

### Ottimizzazioni di DependencyProperty

È necessario definire le proprietà di dipendenza nell'applicazione con estrema attenzione. Se il [DependencyProperty](#) interessa solo le opzioni dei metadati del tipo di rendering, anziché altre opzioni di metadati, ad esempio [AffectsMeasure](#), è necessario contrassegnarlo come tale eseguendo l'override dei relativi metadati. Per altre informazioni sull'override dei metadati delle proprietà o su come ottenere i metadati delle proprietà, vedere [Metadati delle proprietà di dipendenza](#).

Nel caso in cui non tutte le modifiche alle proprietà influiscano effettivamente sulla misura, la disposizione e il rendering, può essere più vantaggioso avere un gestore delle modifiche alle proprietà che consenta di invalidare manualmente la misura, la disposizione e i passaggi di rendering. È possibile, ad esempio, decidere di eseguire nuovamente il rendering di uno sfondo solo se un valore è superiore a un limite impostato. In questo caso, il rendering verrebbe invalidato dal gestore delle modifiche alle proprietà solo se il valore supera il limite

impostato.

### Problemi legati alla possibilità di rendere ereditabile un oggetto DependencyProperty

Per impostazione predefinita, le proprietà di dipendenza registrate non sono ereditabili. È possibile tuttavia rendere ereditabile qualsiasi proprietà in modo esplicito. Sebbene possa essere utile, la conversione di una proprietà per renderla ereditabile incide sulle prestazioni, poiché aumenta la durata della procedura di annullamento della convalida della proprietà.

### Uso corretto di RegisterClassHandler

Quando si chiama [RegisterClassHandler](#) consente di salvare lo stato dell'istanza, è importante tenere presente che il gestore viene chiamato su ogni istanza, che può causare problemi di prestazioni. Usare [RegisterClassHandler](#) solo quando l'applicazione richiede di salvare lo stato dell'istanza.

### Impostazione del valore predefinito di un oggetto DependencyProperty durante le registrazioni

Quando si crea un [DependencyProperty](#) che richiede un valore predefinito, impostare il valore utilizzando i metadati predefiniti passati come parametro al metodo [Register](#) della [DependencyProperty](#). Usare questa tecnica, anziché impostare il valore della proprietà, in un costruttore o in ogni istanza di un elemento.

### Impostazione del valore PropertyMetadata usando Register

Quando si crea una [DependencyProperty](#), è possibile impostare l'[PropertyMetadata](#) utilizzando i metodi [Register](#) o [OverrideMetadata](#). Sebbene l'oggetto possa avere un costruttore statico per chiamare [OverrideMetadata](#), questa non è la soluzione ottimale e avrà un effetto sulle prestazioni. Per ottenere prestazioni ottimali, impostare il [PropertyMetadata](#) durante la chiamata al [Register](#).

## Oggetti Freezable

Un [Freezable](#) è un tipo speciale di oggetto con due stati: non bloccato e bloccato. Bloccare gli oggetti ogni volta che è possibile migliora le prestazioni dell'applicazione e ne riduce il working set. Per altre informazioni, vedere [Cenni preliminari sugli oggetti Freezable](#).

Ogni [Freezable](#) dispone di un evento [Changed](#) che viene generato ogni volta che viene modificato. Le notifiche di modifica, tuttavia, sono molto dispendiose in termini di prestazioni dell'applicazione.

Si consideri l'esempio seguente in cui ogni [Rectangle](#) utilizza lo stesso oggetto [Brush](#):

```
rectangle_1.Fill = myBrush;
rectangle_2.Fill = myBrush;
rectangle_3.Fill = myBrush;
// ...
rectangle_10.Fill = myBrush;
```

```
rectangle_1.Fill = myBrush
rectangle_2.Fill = myBrush
rectangle_3.Fill = myBrush
...
rectangle_10.Fill = myBrush
```

Per impostazione predefinita, WPF fornisce un gestore eventi per l'evento [Changed](#) dell'oggetto [SolidColorBrush](#) per invalidare la proprietà [Fill](#) dell'oggetto [Rectangle](#). In questo caso, ogni volta che il [SolidColorBrush](#) deve generare l'evento [Changed](#), è necessario richiamare la funzione di callback per ogni [Rectangle](#): l'accumulo di queste chiamate di funzione di callback impone una riduzione significativa delle prestazioni. Ma non solo: per aggiungere e rimuovere gestori in questa fase, l'applicazione deve scorrere tutto l'elenco, con un considerevole dispendio di prestazioni. Se lo scenario dell'applicazione non modifica mai il [SolidColorBrush](#), si pagherà il costo della gestione inutilmente dei gestori di eventi [Changed](#).

Il blocco di un [Freezable](#) può migliorare le prestazioni, perché non è più necessario spendere risorse per la gestione delle notifiche di modifica. Nella tabella seguente viene illustrata la dimensione di una semplice [SolidColorBrush](#) quando la relativa proprietà [IsFrozen](#) è impostata su `true`, rispetto a quando non lo è. In questo modo si presuppone l'applicazione di un pennello alla proprietà [Fill](#) di dieci oggetti [Rectangle](#).

STATE	DIMENSIONE
<a href="#">SolidColorBrush</a> bloccati	212 byte
<a href="#">SolidColorBrush</a> non bloccati	972 byte

Nell'esempio di codice seguente viene dimostrato questo concetto:

```
Brush frozenBrush = new SolidColorBrush(Colors.Blue);
frozenBrush.Freeze();
Brush nonFrozenBrush = new SolidColorBrush(Colors.Blue);

for (int i = 0; i < 10; i++)
{
 // Create a Rectangle using a non-frozed Brush.
 Rectangle rectangleNonFrozen = new Rectangle();
 rectangleNonFrozen.Fill = nonFrozenBrush;

 // Create a Rectangle using a frozed Brush.
 Rectangle rectangleFrozen = new Rectangle();
 rectangleFrozen.Fill = frozenBrush;
}
```

```
Dim frozenBrush As Brush = New SolidColorBrush(Colors.Blue)
frozenBrush.Freeze()
Dim nonFrozenBrush As Brush = New SolidColorBrush(Colors.Blue)

For i As Integer = 0 To 9
 ' Create a Rectangle using a non-frozed Brush.
 Dim rectangleNonFrozen As New Rectangle()
 rectangleNonFrozen.Fill = nonFrozenBrush

 ' Create a Rectangle using a frozed Brush.
 Dim rectangleFrozen As New Rectangle()
 rectangleFrozen.Fill = frozenBrush
Next i
```

### Gestori modificati su oggetti Freezable non bloccati possono mantenere gli oggetti attivi

Il delegato passato da un oggetto a un evento [Changed](#) dell'oggetto [Freezable](#) è effettivamente un riferimento a tale oggetto. I gestori eventi [Changed](#) possono pertanto mantengono gli oggetti attivi più a lungo del previsto. Quando si esegue la pulizia di un oggetto che è stato registrato per restare in ascolto di un evento [Changed](#) dell'oggetto [Freezable](#), è essenziale rimuovere il delegato prima di rilasciare l'oggetto.

WPF associa anche gli eventi [Changed](#) internamente. Ad esempio, tutte le proprietà di dipendenza che accettano [Freezable](#) come valore ascolteranno automaticamente [Changed](#) eventi. La proprietà [Fill](#), che accetta un [Brush](#), illustra questo concetto.

```
Brush myBrush = new SolidColorBrush(Colors.Red);
Rectangle myRectangle = new Rectangle();
myRectangle.Fill = myBrush;
```

```
Dim myBrush As Brush = New SolidColorBrush(Colors.Red)
Dim myRectangle As New Rectangle()
myRectangle.Fill = myBrush
```

Nell'assegnazione di `myBrush` `myRectangle.Fill`, un delegato che fa riferimento all'oggetto `Rectangle` verrà aggiunto all'evento `Changed` dell'oggetto `SolidColorBrush`. Nel codice seguente, quindi, `myRect` non viene reso idoneo per operazioni di Garbage Collection:

```
myRectangle = null;
```

```
myRectangle = Nothing
```

In questo caso `myBrush` mantiene sempre `myRectangle` attivo e richiamerà il computer quando genera il relativo evento di `Changed`. Si noti che l'assegnazione di `myBrush` alla proprietà `Fill` di un nuovo `Rectangle` aggiungerà semplicemente un altro gestore eventi a `myBrush`.

Il metodo consigliato per pulire questi tipi di oggetti consiste nel rimuovere il `Brush` dalla proprietà `Fill`, che a sua volta rimuoverà il gestore dell'evento `Changed`.

```
myRectangle.Fill = null;
myRectangle = null;
```

```
myRectangle.Fill = Nothing
myRectangle = Nothing
```

## Virtualizzazione dell'interfaccia utente

WPF fornisce inoltre una variante dell'elemento `StackPanel` che "virtualizza" automaticamente il contenuto figlio associato a dati. In questo contesto il termine virtualizzare si riferisce a una tecnica grazie alla quale, a partire da un numero più elevato di elementi dati, viene generato un subset di oggetti in base agli elementi visibili sullo schermo. La generazione di un elevato numero di elementi dell'interfaccia utente, quando solo alcuni possono essere visualizzati sullo schermo in un momento specifico, richiede un consumo intensivo di risorse sia in termini di memoria che di processore. `VirtualizingStackPanel` (tramite la funzionalità fornita da `VirtualizingPanel`) calcola gli elementi visibili e utilizza il `ItemContainerGenerator` da un `ItemsControl` (ad esempio `ListBox` o `ListView`) per creare solo elementi per gli elementi visibili.

Per ottimizzare le prestazioni, gli oggetti visivi per questi elementi vengono generati o mantenuti attivi solo se sono visibili sullo schermo. Quando non si trovano più nell'area visualizzabile del controllo, gli oggetti visibili possono essere rimossi. Questa operazione non deve essere confusa con la virtualizzazione dei dati, in cui gli oggetti dati non sono tutti presenti nella raccolta locale, ma trasmessi a seconda delle esigenze.

La tabella seguente mostra il tempo trascorso per l'aggiunta e il rendering di 5000 elementi `TextBlock` a una `StackPanel` e a un `VirtualizingStackPanel`. In questo scenario, le misurazioni rappresentano il tempo che intercorre tra la connessione di una stringa di testo alla proprietà `ItemsSource` di un oggetto `ItemsControl` al momento in cui gli elementi del pannello visualizzano la stringa di testo.

PANNELLO HOST	TEMPO DI RENDERING (MS)
<code>StackPanel</code>	3210

PANNELLO HOST	TEMPO DI RENDERING (MS)
VirtualizingStackPanel	46

## Vedere anche

- [Ottimizzazione delle prestazioni di applicazioni WPF](#)
- [Pianificazione delle prestazioni dell'applicazione](#)
- [Sfruttare appieno l'hardware](#)
- [Ottimizzazione delle prestazioni: layout e progettazione](#)
- [Grafica bidimensionale e creazione di immagini](#)
- [Risorse di applicazioni](#)
- [Text](#)
- [Data binding](#)
- [Altri suggerimenti relativi alle prestazioni](#)

# Ottimizzazione delle prestazioni: risorse di applicazioni

04/11/2019 • 5 minutes to read • [Edit Online](#)

WPF consente di condividere le risorse dell'applicazione in modo che sia possibile supportare un aspetto o un comportamento coerente in elementi tipizzati simili. In questo argomento vengono fornite alcune raccomandazioni in questa area che consentono di migliorare le prestazioni delle applicazioni.

Per altre informazioni sulle risorse, vedere [Risorse XAML](#).

## Condivisione di risorse

Se l'applicazione usa controlli personalizzati e definisce le risorse in un [ResourceDictionary](#) (nodo risorse XAML), è consigliabile definire le risorse a livello di [Application](#) o [Window](#) oppure definirle nel tema predefinito per i controlli personalizzati. La definizione delle risorse nel [ResourceDictionary](#) di un controllo personalizzato impone un effetto sulle prestazioni per ogni istanza di tale controllo. Se, ad esempio, si dispone di operazioni pennello a elevato utilizzo di prestazioni definite come parte della definizione di risorsa di un controllo personalizzato e di molte istanze del controllo personalizzato, l'working set dell'applicazione aumenterà significativamente.

Per illustrare questo punto, tenere presente quanto segue. Supponiamo che si stia sviluppando un gioco di carte usando WPF. Per la maggior parte dei giochi di carte sono necessarie 52 schede con 52 visi differenti. Si decide di implementare un controllo personalizzato di una scheda e si definiscono 52 pennelli (ognuno dei quali rappresenta un volto di scheda) nelle risorse del controllo personalizzato della scheda. Nell'applicazione principale si creano inizialmente 52 istanze di questo controllo personalizzato della scheda. Ogni istanza del controllo personalizzato della scheda genera 52 istanze di [Brush](#) oggetti, che fornisce un totale di  $52 * 52$  [Brush](#) oggetti nell'applicazione. Spostando i pennelli fuori dalle risorse di controllo personalizzato per il [Application](#) o [Window](#) a livello di oggetto o definendoli nel tema predefinito per il controllo personalizzato, si riduce il working set dell'applicazione, perché ora si condividono i pennelli 52 tra 52 istanze del controllo scheda.

## Condivisione di un pennello senza copia

Se si dispone di più elementi che utilizzano lo stesso oggetto [Brush](#), definire il pennello come risorsa e farvi riferimento, anziché definire il pennello inline in XAML. Questo metodo creerà un'istanza e la riutilizzerà, mentre la definizione dei pennelli inline in XAML crea una nuova istanza per ogni elemento.

Nell'esempio di markup seguente viene illustrato questo punto:

```

<StackPanel.Resources>
 <LinearGradientBrush x:Key="myBrush" StartPoint="0,0.5" EndPoint="1,0.5" Opacity="0.5">
 <LinearGradientBrush.GradientStops>
 <GradientStopCollection>
 <GradientStop Color="GoldenRod" Offset="0" />
 <GradientStop Color="White" Offset="1" />
 </GradientStopCollection>
 </LinearGradientBrush.GradientStops>
 </LinearGradientBrush>
</StackPanel.Resources>

<!-- Non-shared Brush object. -->
<Label>
 Label 1
 <Label.Background>
 <LinearGradientBrush StartPoint="0,0.5" EndPoint="1,0.5" Opacity="0.5">
 <LinearGradientBrush.GradientStops>
 <GradientStopCollection>
 <GradientStop Color="GoldenRod" Offset="0" />
 <GradientStop Color="White" Offset="1" />
 </GradientStopCollection>
 </LinearGradientBrush.GradientStops>
 </LinearGradientBrush>
 </Label.Background>
</Label>

<!-- Shared Brush object. -->
<Label Background="{StaticResource myBrush}">Label 2</Label>
<Label Background="{StaticResource myBrush}">Label 3</Label>

```

## Usare risorse statiche quando possibile

Una risorsa statica fornisce un valore per qualsiasi attributo di proprietà XAML cercando un riferimento a una risorsa già definita. Il comportamento di ricerca per tale risorsa è analogo alla ricerca in fase di compilazione.

Una risorsa dinamica, d'altra parte, creerà un'espressione temporanea durante la compilazione iniziale e quindi rinvia la ricerca delle risorse finché il valore della risorsa richiesta non è effettivamente necessario per costruire un oggetto. Il comportamento di ricerca per tale risorsa è analogo alla ricerca in fase di esecuzione, che impone un effetto sulle prestazioni. Usare le risorse statiche laddove possibile nell'applicazione, usando risorse dinamiche solo quando necessario.

L'esempio di markup seguente illustra l'uso di entrambi i tipi di risorse:

```

<StackPanel.Resources>
 <SolidColorBrush x:Key="myBrush" Color="Teal"/>
</StackPanel.Resources>

<!-- StaticResource reference -->
<Label Foreground="{StaticResource myBrush}">Label 1</Label>

<!-- DynamicResource reference -->
<Label Foreground="{DynamicResource {x:Static SystemColors.ControlBrushKey}}">Label 2</Label>

```

## Vedere anche

- [Ottimizzazione delle prestazioni di applicazioni WPF](#)
- [Pianificazione delle prestazioni dell'applicazione](#)
- [Sfruttare appieno l'hardware](#)
- [Ottimizzazione delle prestazioni: layout e progettazione](#)

- Grafica bidimensionale e creazione di immagini
- Comportamento dell'oggetto
- Testo
- Data binding
- Altri suggerimenti relativi alle prestazioni

# Ottimizzazione delle prestazioni: testo

10/01/2020 • 16 minutes to read • [Edit Online](#)

WPF include il supporto per la presentazione di contenuto di testo tramite l'uso di controlli avanzati della interfaccia utente. In generale, è possibile suddividere il rendering del testo in tre livelli:

1. Utilizzando direttamente gli oggetti [Glyphs](#) e [GlyphRun](#).
2. Utilizzando l'oggetto [FormattedText](#).
3. Uso di controlli di alto livello, ad esempio gli oggetti [TextBlock](#) e [FlowDocument](#).

Questo argomento offre utili suggerimenti sulle prestazioni del rendering del testo.

## Rendering del testo a livello di glifo

Windows Presentation Foundation (WPF) offre supporto avanzato per il testo, incluso il markup a livello di glifo, con accesso diretto ai [Glyphs](#) per i clienti che desiderano intercettare e mantenere il testo dopo la formattazione. Queste funzionalità forniscono il supporto fondamentale per i diversi requisiti di rendering del testo in ognuno degli scenari seguenti.

- Visualizzazione di documenti a formato fisso.
- Scenari di stampa.
  - Extensible Application Markup Language (XAML) come linguaggio della stampante.
  - Microsoft XPS Document Writer.
  - Driver della stampante precedenti, output delle applicazioni Win32 nel formato fisso.
  - Formato dello spooling di stampa.
- Rappresentazione di documenti a formato fisso, inclusi i client per le versioni precedenti di Windows e altri dispositivi di elaborazione.

### NOTE

[Glyphs](#) e [GlyphRun](#) sono progettati per scenari di presentazione e stampa di documenti a formato fisso. Windows Presentation Foundation (WPF) fornisce diversi elementi per il layout generale e scenari di interfaccia utente, ad esempio [Label](#) e [TextBlock](#). Per altre informazioni sugli scenari di layout e dell'Interfaccia utente, vedere [Funzionalità tipografiche di WPF](#).

Negli esempi seguenti viene illustrato come definire le proprietà per un oggetto [Glyphs](#) in Extensible Application Markup Language (XAML). L'oggetto [Glyphs](#) rappresenta l'output di un [GlyphRun](#) in XAML. Negli esempi si presuppone che i tipi di carattere Arial, Courier New e Times New Roman siano installati nella cartella **C:\WINDOWS\Fonts** nel computer locale.

```

<!-- The example shows how to use a Glyphs object. -->
<Page
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
>

 <StackPanel Background="PowderBlue">

 <Glyphs
 FontUri = "C:\WINDOWS\Fonts\TIMES.TTF"
 FontRenderingEmSize = "100"
 StyleSimulations = "BoldSimulation"
 UnicodeString = "Hello World!"
 Fill = "Black"
 OriginX = "100"
 OriginY = "200"
 />

 </StackPanel>
</Page>

```

## Uso di DrawGlyphRun

Se si dispone di un controllo personalizzato e si desidera eseguire il rendering dei glifi, utilizzare il metodo [DrawGlyphRun](#).

WPF fornisce anche servizi di livello inferiore per la formattazione di testo personalizzata tramite l'utilizzo dell'oggetto [FormattedText](#). Il modo più efficiente per eseguire il rendering del testo in Windows Presentation Foundation (WPF) consiste nel generare contenuto di testo a livello di glifo utilizzando [Glyphs](#) e [GlyphRun](#). Tuttavia, il costo di questa efficienza è la perdita di una formattazione del testo RTF facile da usare, ovvero funzionalità predefinite dei controlli Windows Presentation Foundation (WPF), ad esempio [TextBlock](#) e [FlowDocument](#).

## Oggetto FormattedText

L'oggetto [FormattedText](#) consente di creare testo su più righe, in cui ogni carattere del testo può essere formattato singolarmente. Per altre informazioni, vedere [Disegno di testo formattato](#).

Per creare un testo formattato, chiamare il costruttore [FormattedText](#) per creare un oggetto [FormattedText](#). Dopo aver creato la stringa di testo formattato iniziale, è possibile applicare una gamma di stili di formattazione. Se l'applicazione desidera implementare il proprio layout, l'oggetto [FormattedText](#) è preferibile rispetto all'uso di un controllo, ad esempio [TextBlock](#). Per ulteriori informazioni sull'oggetto [FormattedText](#), vedere [disegno di testo formattato](#).

L'oggetto [FormattedText](#) fornisce funzionalità di formattazione del testo di basso livello. È possibile applicare vari stili di formattazione a uno o più caratteri. È ad esempio possibile chiamare entrambi i metodi [SetFontStyle](#) e [SetForegroundBrush](#) per modificare la formattazione dei primi cinque caratteri del testo.

Nell'esempio di codice seguente viene creato un oggetto [FormattedText](#) e ne viene eseguito il rendering.

```
protected override void OnRender(DrawingContext drawingContext)
{
 string testString = "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor";

 // Create the initial formatted text string.
 FormattedText formattedText = new FormattedText(
 testString,
 CultureInfo.GetCultureInfo("en-us"),
 FlowDirection.LeftToRight,
 new Typeface("Verdana"),
 32,
 Brushes.Black);

 // Set a maximum width and height. If the text overflows these values, an ellipsis "..." appears.
 formattedText.MaxTextWidth = 300;
 formattedText.MaxTextHeight = 240;

 // Use a larger font size beginning at the first (zero-based) character and continuing for 5 characters.
 // The font size is calculated in terms of points -- not as device-independent pixels.
 formattedText.SetFontSize(36 * (96.0 / 72.0), 0, 5);

 // Use a Bold font weight beginning at the 6th character and continuing for 11 characters.
 formattedText.SetFontWeight(FontWeights.Bold, 6, 11);

 // Use a linear gradient brush beginning at the 6th character and continuing for 11 characters.
 formattedText.SetForegroundBrush(
 new LinearGradientBrush(
 Colors.Orange,
 Colors.Teal,
 90.0),
 6, 11);

 // Use an Italic font style beginning at the 28th character and continuing for 28 characters.
 formattedTextSetFontStyle(FontStyles.Italic, 28, 28);

 // Draw the formatted text string to the DrawingContext of the control.
 drawingContext.DrawText(formattedText, new Point(10, 0));
}
```

```

Protected Overrides Sub OnRender(ByVal drawingContext As DrawingContext)
 Dim testString As String = "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor"

 ' Create the initial formatted text string.
 Dim formattedText As New FormattedText(testString, CultureInfo.GetCultureInfo("en-us"),
 FlowDirection.LeftToRight, New Typeface("Verdana"), 32, Brushes.Black)

 ' Set a maximum width and height. If the text overflows these values, an ellipsis "..." appears.
 formattedText.MaxTextWidth = 300
 formattedText.MaxTextHeight = 240

 ' Use a larger font size beginning at the first (zero-based) character and continuing for 5 characters.
 ' The font size is calculated in terms of points -- not as device-independent pixels.
 formattedText.SetFontSize(36 * (96.0 / 72.0), 0, 5)

 ' Use a Bold font weight beginning at the 6th character and continuing for 11 characters.
 formattedText.SetFontWeight(FontWeights.Bold, 6, 11)

 ' Use a linear gradient brush beginning at the 6th character and continuing for 11 characters.
 formattedText.SetForegroundBrush(New LinearGradientBrush(Colors.Orange, Colors.Teal, 90.0), 6, 11)

 ' Use an Italic font style beginning at the 28th character and continuing for 28 characters.
 formattedTextSetFontStyle(FontStyles.Italic, 28, 28)

 ' Draw the formatted text string to the DrawingContext of the control.
 drawingContext.DrawText(formattedText, New Point(10, 0))
End Sub

```

## Controlli FlowDocument, TextBlock e Label

WPF include più controlli per la creazione di testo sullo schermo. Ogni controllo è destinato a uno scenario diverso e dispone di un proprio elenco di funzionalità e limitazioni.

### **Un elemento FlowDocument influenza sulle prestazioni più di un elemento TextBlock o Label**

In generale, l'elemento [TextBlock](#) deve essere utilizzato quando è necessario il supporto di testo limitato, ad esempio una breve frase in una interfaccia utente. [Label](#) può essere utilizzato quando è richiesto un supporto di testo minimo. L'elemento [FlowDocument](#) è un contenitore per i documenti riflussi che supportano la presentazione dettagliata del contenuto e, di conseguenza, ha un maggiore effetto sulle prestazioni rispetto all'uso dei controlli [TextBlock](#) o [Label](#).

Per altre informazioni su [FlowDocument](#), vedere [Cenni preliminari sui documenti dinamici](#).

### **Evitare di usare TextBlock in FlowDocument**

L'elemento [TextBlock](#) deriva da [UIElement](#). L'elemento [Run](#) deriva da [TextElement](#), che è meno costoso da usare rispetto a un oggetto derivato da [UIElement](#). Quando possibile, usare [Run](#) anziché [TextBlock](#) per visualizzare il contenuto di testo in un [FlowDocument](#).

L'esempio di markup seguente illustra due modi per impostare il contenuto di testo all'interno di un [FlowDocument](#):

```

<FlowDocument>

 <!-- Text content within a Run (more efficient). -->
 <Paragraph>
 <Run>Line one</Run>
 </Paragraph>

 <!-- Text content within a TextBlock (less efficient). -->
 <Paragraph>
 <TextBlock>Line two</TextBlock>
 </Paragraph>

</FlowDocument>

```

### Evitare di usare un oggetto Run per impostare proprietà di testo

In generale, l'uso di un [Run](#) all'interno di una [TextBlock](#) è più intenso rispetto all'uso di un oggetto [Run](#) esplicito. Se si utilizza una [Run](#) per impostare le proprietà del testo, impostare le proprietà direttamente nel [TextBlock](#).

Nell'esempio di markup seguente vengono illustrate queste due modalità di impostazione di una proprietà di testo, in questo caso la proprietà [FontWeight](#):

```

<!-- Run is used to set text properties. -->
<TextBlock>
 <Run FontWeight="Bold">Hello, world</Run>
</TextBlock>

<!-- TextBlock is used to set text properties, which is more efficient. -->
<TextBlock FontWeight="Bold">
 Hello, world
</TextBlock>

```

La tabella seguente mostra il costo della visualizzazione di 1000 [TextBlock](#) oggetti con e senza una [Run](#) esplicita.

TIPO DI TEXTBLOCK	TEMPO DI CREAZIONE (MS)	TEMPO DI RENDERING (MS)
Proprietà di testo impostate con Run	146	540
Proprietà di testo impostate con TextBlock	43	453

### Evitare di eseguire il data binding alla proprietà Label.Content

Si immagini uno scenario in cui si dispone di un oggetto [Label](#) aggiornato di frequente da un'origine [String](#). Quando data binding la proprietà [Content](#) dell'elemento [Label](#) all'oggetto di origine [String](#), è possibile che si verifichi una riduzione delle prestazioni. Ogni volta che viene aggiornata la [String](#) di origine, l'oggetto [String](#) precedente viene eliminato e viene ricreato un nuovo [String](#), perché un oggetto [String](#) non è modificabile e non può essere modificato. Questo, a sua volta, fa sì che il [ContentPresenter](#) dell'oggetto [Label](#) elimini il contenuto precedente e rigeneri il nuovo contenuto per visualizzare il nuovo [String](#).

La soluzione di questo problema è semplice. Se il [Label](#) non è impostato su un valore [ContentTemplate](#) personalizzato, sostituire il [Label](#) con una [TextBlock](#) e associare i dati alla relativa proprietà [Text](#) nella stringa di origine.

PROPRIETÀ CON DATA BINDING	TEMPO DI AGGIORNAMENTO (MS)
Label.Content	835

PROPRIETÀ CON DATA BINDING	TEMPO DI AGGIORNAMENTO (MS)
TextBlock.Text	242

## Collegamento ipertestuale

L'oggetto [Hyperlink](#) è un elemento di contenuto del flusso di livello inline che consente di ospitare collegamenti ipertestuali all'interno del contenuto del flusso.

### Combinare collegamenti ipertestuali in un oggetto TextBlock

È possibile ottimizzare l'utilizzo di più elementi [Hyperlink](#) raggruppando questi elementi all'interno della stessa [TextBlock](#). In questo modo, è possibile ridurre al minimo il numero di oggetti creati nell'applicazione. È possibile, ad esempio, che si voglia visualizzare più collegamenti ipertestuali, come illustrato di seguito:

Home page MSN | MSN

L'esempio di markup seguente mostra più elementi [TextBlock](#) utilizzati per visualizzare i collegamenti ipertestuali:

```
<!-- Hyperlinks in separate TextBlocks. -->
<TextBlock>
 <Hyperlink TextDecorations="None" NavigateUri="http://www.msn.com">MSN Home</Hyperlink>
</TextBlock>

<TextBlock Text=" | "/>

<TextBlock>
 <Hyperlink TextDecorations="None" NavigateUri="http://my.msn.com">My MSN</Hyperlink>
</TextBlock>
```

L'esempio di markup seguente mostra un modo più efficiente per visualizzare i collegamenti ipertestuali, questa volta usando una singola [TextBlock](#):

```
<!-- Hyperlinks combined in the same TextBlock. -->
<TextBlock>
 <Hyperlink TextDecorations="None" NavigateUri="http://www.msn.com">MSN Home</Hyperlink>

 <Run Text=" | " />

 <Hyperlink TextDecorations="None" NavigateUri="http://my.msn.com">My MSN</Hyperlink>
</TextBlock>
```

### Visualizzazione delle sottolineature nei collegamenti ipertestuali solo in caso di eventi MouseEnter

Un oggetto [TextDecoration](#) è un ornamento visivo che è possibile aggiungere al testo; Tuttavia, può essere un'attività che richiede un utilizzo intensivo delle prestazioni. Se si utilizza in modo esteso gli elementi di [Hyperlink](#), è consigliabile visualizzare una sottolineatura solo quando si attiva un evento, ad esempio l'evento [MouseEnter](#). Per altre informazioni, vedere [Specificare se un collegamento ipertestuale è sottolineato](#).

Nell'immagine seguente viene illustrato il modo in cui l'evento [MouseEnter](#) attiva il collegamento ipertestuale sottolineato:



Nell'esempio di markup seguente viene illustrato un [Hyperlink](#) definito con e senza sottolineatura:

```
<!-- Hyperlink with default underline. -->
<Hyperlink NavigateUri="http://www.msn.com">
 MSN Home
</Hyperlink>

<Run Text=" | " />

<!-- Hyperlink with no underline. -->
<Hyperlink Name="myHyperlink" TextDecorations="None"
 MouseEnter="OnMouseEnter"
 MouseLeave="OnMouseLeave"
 NavigateUri="http://www.msn.com">
 My MSN
</Hyperlink>
```

La tabella seguente mostra il costo in termini di prestazioni della visualizzazione degli elementi di 1000 [Hyperlink](#) con e senza una sottolineatura.

COLLEGAMENTO IPERTESTUALE	TEMPO DI CREAZIONE (MS)	TEMPO DI RENDERING (MS)
Con sottolineatura	289	1130
Senza sottolineatura	299	776

## Funzionalità di formattazione del testo

WPF fornisce servizi di formattazione RTF come le sillabazioni automatiche. Questi servizi possono influire sulle prestazioni dell'applicazione e devono essere usati solo se strettamente necessario.

### Evitare l'uso non necessario della sillabazione

La sillabazione automatica trova i punti di interruzione del segno meno per le righe di testo e consente posizioni di interruzione aggiuntive per le linee in [TextBlock](#) e [FlowDocument](#) oggetti. Per impostazione predefinita, la funzionalità di sillabazione automatica è disattivata in questi oggetti. È possibile attivare questa funzionalità impostando la proprietà [IsHyphenationEnabled](#) dell'oggetto su `true`. Tuttavia, l'abilitazione di questa funzionalità comporta l'avvio dell'interoperabilità di Component Object Model (COM) da WPF, che può influisce sulle prestazioni dell'applicazione. È consigliabile quindi usare la sillabazione automatica solo se necessario.

### Usare con attenzione gli elementi [Figure](#)

Un elemento [Figure](#) rappresenta una parte di contenuto dinamico che può essere posizionata in modo assoluto all'interno di una pagina di contenuto. In alcuni casi, una [Figure](#) può causare la riformattazione automatica di un'intera pagina se la posizione è in conflitto con il contenuto già definito. È possibile ridurre al minimo la possibilità di riformattazione non necessaria raggruppando gli elementi [Figure](#) uno accanto all'altro o dichiarando tali elementi nella parte superiore del contenuto in uno scenario di dimensioni di pagina fisse.

### Paragrafo ottimale

La funzionalità di paragrafo ottimale dell'oggetto [FlowDocument](#) dispone i paragrafi, in modo che gli spazi vuoti vengano distribuiti nel modo più uniforme possibile. Per impostazione predefinita, la funzionalità di paragrafo ottimale è disattivata. È possibile abilitare questa funzionalità impostando la proprietà [IsOptimalParagraphEnabled](#) dell'oggetto su `true`. L'attivazione di questa funzionalità, tuttavia, influisce sulle prestazioni dell'applicazione. È consigliabile quindi usare la funzionalità di paragrafo ottimale solo se necessario.

## Vedere anche

- Ottimizzazione delle prestazioni di applicazioni WPF
- Pianificazione delle prestazioni dell'applicazione
- Sfruttare appieno l'hardware
- Ottimizzazione delle prestazioni: layout e progettazione
- Grafica bidimensionale e creazione di immagini
- Comportamento dell'oggetto
- Risorse di applicazioni
- Data binding
- Altri suggerimenti relativi alle prestazioni

# Ottimizzazione delle prestazioni: associazione dati

08/11/2019 • 10 minutes to read • [Edit Online](#)

Il data binding di Windows Presentation Foundation (WPF) rappresenta per le applicazioni un modo semplice e coerente di presentare i dati e interagire con essi. Gli elementi possono essere associati a dati da un'ampia gamma di origini dati sotto forma di oggetti CLR e XML.

Questo argomento offre utili suggerimenti sulle prestazioni del data binding.

## Risoluzione dei riferimenti di data binding

Prima di trattare i problemi di prestazioni del data binding, è opportuno scoprire come vengono risolti dal motore di data binding di Windows Presentation Foundation (WPF) i riferimenti agli oggetti per il binding.

L'origine di un Windows Presentation Foundation (WPF) data binding può essere qualsiasi oggetto CLR. È possibile eseguire l'associazione a proprietà, sottoproprietà o indicizzatori di un oggetto CLR. I riferimenti di associazione vengono risolti tramite Microsoft .NET Reflection del Framework o una [ICustomTypeDescriptor](#). Di seguito vengono descritti i tre metodi disponibili per risolvere riferimenti a oggetti per il data binding.

Il primo metodo prevede l'uso della reflection. In questo caso, l'oggetto  [PropertyInfo](#) viene usato per individuare gli attributi della proprietà e fornisce l'accesso ai metadati della proprietà. Quando si usa l'interfaccia [ICustomTypeDescriptor](#), il motore di data binding usa questa interfaccia per accedere ai valori delle proprietà. L'interfaccia [ICustomTypeDescriptor](#) è particolarmente utile nei casi in cui l'oggetto non dispone di un set statico di proprietà.

Le notifiche di modifica delle proprietà possono essere fornite implementando l'interfaccia [INotifyPropertyChanged](#) o utilizzando le notifiche di modifica associate al [TypeDescriptor](#). Tuttavia, la strategia consigliata per l'implementazione delle notifiche delle modifiche delle proprietà consiste nell'utilizzare [INotifyPropertyChanged](#).

Se l'oggetto di origine è un oggetto CLR e la proprietà di origine è una proprietà CLR, il motore di Windows Presentation Foundation (WPF) data binding deve innanzitutto utilizzare la reflection sull'oggetto di origine per ottenere il [TypeDescriptor](#), quindi eseguire una query per una [PropertyDescriptor](#). Questa sequenza di operazioni di reflection richiede potenzialmente molto tempo da un punto di vista delle prestazioni.

Il secondo metodo per la risoluzione dei riferimenti a oggetti comporta un oggetto di origine CLR che implementa l'interfaccia [INotifyPropertyChanged](#) e una proprietà di origine che è una proprietà CLR. In questo caso, il motore di data binding usa direttamente la reflection sul tipo di origine e ottiene la proprietà necessaria. Sebbene presenti requisiti del working set inferiori rispetto al primo metodo, non si tratta ancora del metodo ottimale.

Il terzo metodo per la risoluzione dei riferimenti a oggetti prevede l'uso di un oggetto di origine che è un [DependencyObject](#) e una proprietà di origine [DependencyProperty](#). In questo caso, non è necessario che il motore di data binding usi la reflection: il motore della proprietà e il motore di data binding risolvono il riferimento alla proprietà in modo indipendente. Si tratta del metodo ottimale per la risoluzione dei riferimenti a oggetti usati per il data binding.

Nella tabella seguente viene confrontata la velocità di data binding la proprietà [Text](#) di elementi [TextBlock](#) 1000 usando questi tre metodi.

BINDING DELLA PROPRIETÀ TEXT DI UN TEXTBLOCK	TEMPO DI BINDING (MS)	TEMPO DI RENDERING: INCLUDE IL BINDING (MS)
A una proprietà di un oggetto CLR	115	314
A una proprietà di un oggetto CLR che implementa <a href="#">INotifyPropertyChanged</a>	115	305
A un <a href="#">DependencyProperty</a> di un <a href="#">DependencyObject</a> .	90	263

## Binding a oggetti CLR di grandi dimensioni

Quando si esegue l'associazione dati a un singolo oggetto CLR con migliaia di proprietà, si verifica un notevole impatto sulle prestazioni. È possibile ridurre questo effetto dividendo il singolo oggetto in più oggetti CLR con un minor numero di proprietà. La tabella mostra i tempi di binding e di rendering per data binding a un singolo oggetto CLR di grandi dimensioni rispetto a più oggetti più piccoli.

DATA BINDING DI 1000 OGGETTI TEXTBLOCK	TEMPO DI BINDING (MS)	TEMPO DI RENDERING: INCLUDE IL BINDING (MS)
A un oggetto CLR con proprietà 1000	950	1200
Per 1000 oggetti CLR con una proprietà	115	314

## Binding a una proprietà [ItemsSource](#)

Si consideri uno scenario in cui è presente un oggetto [List<T>](#) CLR che contiene un elenco di dipendenti che si desidera visualizzare in una [ListBox](#). Per creare una corrispondenza tra questi due oggetti, è necessario associare l'elenco dei dipendenti alla proprietà [ItemsSource](#) della [ListBox](#). Si supponga ora che un nuovo dipendente si unisca al gruppo. Si potrebbe pensare che, per inserire questo nuovo utente nei valori [ListBox](#) associati, è sufficiente aggiungere tale persona all'elenco dei dipendenti e prevedere che questa modifica venga riconosciuta automaticamente dal motore di data binding. Tale presupposto risulterebbe falso; in realtà, la modifica non verrà riflessa automaticamente nel [ListBox](#). Questo è dovuto al fatto che l'oggetto [List<T>](#) CLR non genera automaticamente un evento di modifica della raccolta. Per fare in modo che il [ListBox](#) rilevi le modifiche, è necessario ricreare l'elenco dei dipendenti e ricollegarlo alla proprietà [ItemsSource](#) dell'[ListBox](#). Questa soluzione funziona, ma incide considerevolmente sulle prestazioni. Ogni volta che si riassegna la [ItemsSource](#) di [ListBox](#) a un nuovo oggetto, il [ListBox](#) prima di tutto genera gli elementi precedenti e rigenera l'intero elenco. L'effetto sulle prestazioni viene ingrandito se il [ListBox](#) viene mappato a un [DataTemplate](#) complesso.

Una soluzione molto efficiente per questo problema consiste nel fare in modo che il dipendente elenchi un [ObservableCollection<T>](#). Un oggetto [ObservableCollection<T>](#) genera una notifica di modifica che può essere ricevuta dal motore di data binding. L'evento aggiunge o rimuove un elemento da un [ItemsControl](#) senza la necessità di rigenerare l'intero elenco.

La tabella seguente mostra il tempo necessario per aggiornare la [ListBox](#) (con la virtualizzazione dell'interfaccia utente disattivata) quando viene aggiunto un elemento. Il numero nella prima riga rappresenta il tempo trascorso quando l'oggetto CLR [List<T>](#) viene associato al [ItemsSource](#) di [ListBox](#) elemento. Il numero nella seconda riga rappresenta il tempo trascorso quando un [ObservableCollection<T>](#) viene associato al [ItemsSource](#) dell'elemento del [ListBox](#). Si noti il notevole risparmio di tempo con la strategia di data binding [ObservableCollection<T>](#).

DATA BINDING DELLA PROPRIETÀ ITEMSOURCE	TEMPO DI AGGIORNAMENTO PER 1 ELEMENTO (MS)
A un oggetto <code>List&lt;T&gt;</code> CLR	1656
A un <code>ObservableCollection&lt;T&gt;</code>	20

## Binding di `IList` a `ItemsControl` non `IEnumerable`

Se è possibile scegliere tra associare un `IList<T>` o un `IEnumerable` a un oggetto `ItemsControl`, scegliere l'oggetto `IList<T>`. Il binding `IEnumerable` a un `ItemsControl` impone WPF di creare un wrapper `IList<T>` oggetto, il che significa che le prestazioni sono influenzate dal sovraccarico superfluo di un secondo oggetto.

## Non convertire oggetti CLR in XML solo per il data binding

WPF consente di associare dati al contenuto XML; Tuttavia, data binding al contenuto XML è più lento rispetto data binding agli oggetti CLR. Non convertire i dati dell'oggetto CLR in XML se l'unico scopo è data binding.

## Vedere anche

- [Ottimizzazione delle prestazioni di applicazioni WPF](#)
- [Pianificazione delle prestazioni dell'applicazione](#)
- [Sfruttare appieno l'hardware](#)
- [Ottimizzazione delle prestazioni: layout e progettazione](#)
- [Grafica bidimensionale e creazione di immagini](#)
- [Comportamento dell'oggetto](#)
- [Risorse di applicazioni](#)
- [Testo](#)
- [Altri suggerimenti relativi alle prestazioni](#)
- [Panoramica sul data binding](#)
- [Procedura dettagliata: Memorizzazione dei dati di un'applicazione nella cache di un'applicazione WPF](#)

# Ottimizzazione delle prestazioni: controlli

28/01/2020 • 9 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) include molti componenti dell'interfaccia utente comuni utilizzati nella maggior parte delle applicazioni Windows. Questo argomento illustra le tecniche da adottare per migliorare le prestazioni dell'interfaccia utente.

## Visualizzazione di set di dati di grandi dimensioni

Per visualizzare elenchi di elementi in un'applicazione, vengono utilizzati controlli WPF come [ListView](#) e [ComboBox](#). Se l'elenco da visualizzare è molto lungo, le prestazioni dell'applicazione possono risentirne. Questo perché il sistema di layout standard crea un contenitore di layout per ogni elemento associato al controllo elenco e ne calcola le dimensioni di layout e la posizione. In genere, non è necessario visualizzare tutti gli elementi contemporaneamente, ma ne viene visualizzato un sottoinsieme e l'utente scorre l'elenco. In questo caso è utile usare la *virtualizzazione* dell'interfaccia utente, ovvero la generazione del contenitore dell'elemento e il calcolo del layout associato sono rinviati al momento in cui l'elemento risulta visibile.

La virtualizzazione dell'interfaccia utente è un aspetto importante dei controlli elenco e non deve essere confusa con la virtualizzazione dei dati. Con la virtualizzazione dell'interfaccia utente vengono archiviati in memoria solo gli elementi visibili, mentre in uno scenario di associazione dati viene archiviata in memoria l'intera struttura dei dati. Con la virtualizzazione dei dati, al contrario, vengono archiviati in memoria solo gli elementi dati visibili sullo schermo.

Per impostazione predefinita, la virtualizzazione dell'interfaccia utente è abilitata per i controlli [ListView](#) e [ListBox](#) quando i relativi elementi elenco sono associati ai dati. [TreeView](#) virtualizzazione può essere abilitata impostando la proprietà associata [VirtualizingStackPanel.IsVirtualizing](#) su `true`. Se si vuole abilitare la virtualizzazione dell'interfaccia utente per i controlli personalizzati che derivano da [ItemsControl](#) o da controlli elemento esistenti che usano la classe [StackPanel](#), ad esempio [ComboBox](#), è possibile impostare il [ItemsPanel](#) su [VirtualizingStackPanel](#) e impostare [IsVirtualizing](#) su `true`. Purtroppo è possibile che la virtualizzazione dell'interfaccia utente venga inavvertitamente disabilitata per questi controlli. Di seguito sono elencate le condizioni che disabilitano la virtualizzazione dell'interfaccia utente.

- I contenitori di elementi vengono aggiunti direttamente al [ItemsControl](#). Se, ad esempio, un'applicazione aggiunge in modo esplicito [ListBoxItem](#) oggetti a una [ListBox](#), il [ListBox](#) non virtualizza gli oggetti [ListBoxItem](#).
- I contenitori di elementi nel [ItemsControl](#) sono di tipi diversi. Un [Menu](#) che utilizza [Separator](#) oggetti, ad esempio, non è in grado di implementare il riciclo degli elementi perché il [Menu](#) contiene oggetti di tipo [Separator](#) e [MenuItem](#).
- Impostazione [CanContentScroll](#) su `false`.
- Impostazione [IsVirtualizing](#) su `false`.

Un aspetto importante di cui tener conto quando si virtualizzano contenitori di elementi è la disponibilità di informazioni aggiuntive sullo stato associate a un contenitore di elementi appartenente all'elemento. Nel caso in cui queste informazioni siano disponibili, è necessario salvare lo stato aggiuntivo. Ad esempio, si potrebbe avere un elemento contenuto in un controllo [Expander](#) e lo stato [IsExpanded](#) è associato al contenitore dell'elemento e non all'elemento stesso. Quando il contenitore viene riutilizzato per un nuovo elemento, viene utilizzato il valore corrente di [IsExpanded](#) per il nuovo elemento. Inoltre, l'elemento precedente perde il valore [IsExpanded](#) corretto.

Attualmente nessun controllo WPF offre il supporto incorporato per la virtualizzazione dei dati.

## Riciclo del contenitore

Un'ottimizzazione per la virtualizzazione dell'interfaccia utente aggiunta in .NET Framework 3,5 SP1 per i controlli che ereditano da [ItemsControl](#) è il *riciclo del contenitore*, che può anche migliorare le prestazioni di scorrimento. Quando viene popolato un [ItemsControl](#) che usa la virtualizzazione dell'interfaccia utente, viene creato un contenitore di elementi per ogni elemento che scorre nella visualizzazione ed Elimina il contenitore di elementi per ogni elemento che scorre fuori dalla visualizzazione. Il *riciclo del contenitore* consente al controllo di riutilizzare i contenitori di elementi esistenti per elementi di dati diversi, in modo che i contenitori di elementi non vengano creati ed eliminati continuamente quando l'utente scorre il [ItemsControl](#). È possibile scegliere di abilitare il riciclo degli elementi impostando la proprietà associata [VirtualizationMode](#) su [Recycling](#).

Eventuali [ItemsControl](#) che supportano la virtualizzazione possono usare il riciclo dei contenitori. Per un esempio di come abilitare il riciclo del contenitore in un [ListBox](#), vedere [migliorare le prestazioni di scorrimento di un controllo ListBox](#).

## Supporto della virtualizzazione bidirezionale

[VirtualizingStackPanel](#) offre supporto incorporato per la virtualizzazione dell'interfaccia utente in una direzione, orizzontalmente o verticalmente. Se si vuole usare la virtualizzazione bidirezionale per i controlli, è necessario implementare un pannello personalizzato che estende la classe [VirtualizingStackPanel](#). La classe [VirtualizingStackPanel](#) espone metodi virtuali come [OnViewportSizeChanged](#), [LineUp](#), [PageUp](#) e [MouseWheelUp](#). Questi metodi virtuali consentono di rilevare una modifica nella parte visibile di un elenco e di gestirla di conseguenza.

## Ottimizzazione di modelli

La struttura ad albero visuale contiene tutti gli elementi visivi di un'applicazione. Oltre agli oggetti creati direttamente, contiene anche oggetti generati dall'espansione del modello. Ad esempio, quando si crea una [Button](#), si ottengono anche [ClassicBorderDecorator](#) e [ContentPresenter](#) oggetti nella struttura ad albero visuale. Se non si è provveduto a ottimizzare i modelli di controllo, quindi, è possibile che si creino molti oggetti in più non necessari nella struttura ad albero visuale. Per altre informazioni sulla struttura ad albero visuale, vedere [Cenni preliminari sul rendering della grafica WPF](#).

## Scorrimento posticipato

Per impostazione predefinita, quando l'utente trascina il cursore su una barra di scorrimento, la visualizzazione del contenuto viene continuamente aggiornata. Se lo scorrimento è lento nel controllo, valutare la possibilità di usare lo scorrimento posticipato, in cui il contenuto viene aggiornato solo quando l'utente rilascia il cursore.

Per implementare lo scorrimento posticipato, impostare la proprietà [IsDeferredScrollingEnabled](#) su `true`. [IsDeferredScrollingEnabled](#) è una proprietà associata e può essere impostata su [ScrollViewer](#) e qualsiasi controllo con un [ScrollViewer](#) nel relativo modello di controllo.

## Controlli che implementano le funzionalità delle prestazioni

La tabella seguente elenca i controlli comuni per la visualizzazione dei dati e ne indica il tipo di supporto per le funzionalità relative alle prestazioni. Vedere le sezioni precedenti per informazioni su come abilitare queste funzionalità.

CONTROL	VIRTUALIZZAZIONE	RICICLO DEL CONTENITORE	SCORRIMENTO POSTICIPATO
<a href="#">ComboBox</a>	Può essere abilitato	Può essere abilitato	Può essere abilitato

CONTROL	VIRTUALIZZAZIONE	RICICLO DEL CONTENITORE	SCORRIMENTO POSTICIPATO
ContextMenu	Può essere abilitato	Può essere abilitato	Può essere abilitato
DocumentViewer	Non disponibile	Non disponibile	Può essere abilitato
ListBox	Default	Può essere abilitato	Può essere abilitato
ListView	Default	Può essere abilitato	Può essere abilitato
TreeView	Può essere abilitato	Può essere abilitato	Può essere abilitato
ToolBar	Non disponibile	Non disponibile	Può essere abilitato

#### NOTE

Per un esempio di come abilitare la virtualizzazione e il riciclo del contenitore in un [TreeView](#), vedere [migliorare le prestazioni di un controllo TreeView](#).

## Vedere anche

- [Layout](#)
- [Ottimizzazione delle prestazioni: layout e progettazione](#)
- [Data binding](#)
- [Controlli](#)
- [Applicazione di stili e modelli](#)
- [Procedura dettagliata: Memorizzazione dei dati di un'applicazione nella cache di un'applicazione WPF](#)

# Ottimizzazione delle prestazioni: altri suggerimenti

08/01/2020 • 6 minutes to read • [Edit Online](#)

Questo argomento offre suggerimenti sulle prestazioni aggiuntivi rispetto a quelli descritti negli argomenti della sezione [Ottimizzazione delle prestazioni di applicazioni WPF](#).

Di seguito sono elencate le diverse sezioni di questo argomento:

- [Pennelli opachi ed elementi opachi](#)
- [Navigazione a un oggetto](#)
- [Hit testing su superfici 3D ampie](#)
- [Evento CompositionTarget.Rendering](#)
- [Evitare l'uso di ScrollBarVisibility=Auto](#)
- [Configurare il servizio Cache tipi di carattere per ridurre i tempi di avvio](#)

## Pennelli opachi ed elementi opachi

Quando si usa un [Brush](#) per impostare la [Fill](#) o [Stroke](#) di un elemento, è preferibile impostare il valore di [Brush.Opacity](#) anziché l'impostazione della proprietà [Opacity](#) dell'elemento. La modifica della proprietà [Opacity](#) di un elemento può provocare WPF la creazione di una superficie temporanea.

## Navigazione a un oggetto

L'oggetto [NavigationWindow](#) deriva da [Window](#) e lo estende con il supporto per la navigazione del contenuto, principalmente aggregando [NavigationService](#) e [Journal](#). È possibile aggiornare l'area client di [NavigationWindow](#) specificando un URI (Uniform Resource Identifier) o un oggetto. L'esempio seguente illustra entrambi i metodi:

```
private void buttonGoToUri(object sender, RoutedEventArgs args)
{
 navWindow.Source = new Uri("NewPage.xaml", UriKind.RelativeOrAbsolute);
}

private void buttonGoNewObject(object sender, RoutedEventArgs args)
{
 NewPage nextPage = new NewPage();
 nextPage.InitializeComponent();
 navWindow.Content = nextPage;
}
```

```
Private Sub buttonGoToUri(ByVal sender As Object, ByVal args As RoutedEventArgs)
 navWindow.Source = New Uri("NewPage.xaml", UriKind.RelativeOrAbsolute)
End Sub

Private Sub buttonGoNewObject(ByVal sender As Object, ByVal args As RoutedEventArgs)
 Dim nextPage As New NewPage()
 nextPage.InitializeComponent()
 navWindow.Content = nextPage
End Sub
```

Ogni oggetto [NavigationWindow](#) dispone di un journal che registra la cronologia di navigazione dell'utente in tale finestra. Uno degli scopi del journal è quello di consentire agli utenti di rintracciare i passaggi eseguiti.

Quando si esegue la navigazione utilizzando un URI (Uniform Resource Identifier), nel journal viene archiviato solo il riferimento URI (Uniform Resource Identifier). Ogni volta che viene rivisitata, la pagina viene ricostruita in modo dinamico; questa operazione può richiedere molto tempo, a seconda della complessità della pagina. In questo caso, il costo di archiviazione nel journal è basso, ma il tempo necessario per la ricostruzione della pagina è potenzialmente lungo.

Con la navigazione tramite un oggetto, nel journal viene archiviata l'intera struttura ad albero visiva dell'oggetto. Ogni volta che la pagina viene rivisitata, quindi, ne viene immediatamente eseguito il rendering, senza che debba essere ricostruita. In questo caso, il costo di archiviazione nel journal è alto, ma il tempo necessario per la ricostruzione della pagina è ridotto.

Quando si utilizza l'oggetto [NavigationWindow](#), è necessario tenere presente il modo in cui il supporto per il Journal influisce sulle prestazioni dell'applicazione. Per altre informazioni, vedere [Cenni preliminari sulla navigazione](#).

## Hit testing su superfici 3D ampie

L'hit testing su superfici 3D ampie presenta requisiti particolarmente elevati in termini di consumo della CPU, soprattutto se alla superficie 3D viene aggiunta un'animazione. Se l'hit testing su queste superfici non è necessario, quindi, è opportuno disabilitarlo. Gli oggetti derivati da [UIElement](#) possono disabilitare l'hit testing impostando la proprietà [IsHitTestVisible](#) su `false`.

## Evento [CompositionTarget.Rendering](#)

L'evento [CompositionTarget.Rendering](#) causa l'animazione continua di WPF. Se si usa questo evento, disconnetterlo ogni volta che è possibile.

## Evitare l'uso di [ScrollBarVisibility=Auto](#)

Quando possibile, evitare di utilizzare il valore [ScrollBarVisibility.Auto](#) per le proprietà [HorizontalScrollBarVisibility](#) e [VerticalScrollBarVisibility](#). Queste proprietà sono definite per gli oggetti [RichTextBox](#), [ScrollViewer](#) e [TextBox](#) e come proprietà associata per l'oggetto [ListBox](#). Impostare invece [ScrollBarVisibility](#) su [Disabled](#), [Hidden](#) o [Visible](#).

Il valore [Auto](#) è destinato ai casi in cui lo spazio è limitato e le barre di scorrimento devono essere visualizzate solo quando necessario. Ad esempio, può essere utile usare questo valore [ScrollBarVisibility](#) con una [ListBox](#) di 30 elementi, anziché un [TextBox](#) con centinaia di righe di testo.

## Configurare il servizio Cache tipi di carattere per ridurre il tempo di avvio

Il servizio cache dei tipi di carattere WPF condivide i dati di tipo carattere tra le applicazioni WPF. La prima applicazione WPF eseguita avvia il servizio se il servizio non è già in esecuzione. Se si utilizza Windows Vista, è possibile impostare il servizio "Windows Presentation Foundation (WPF) font cache 3.0.0.0" da "Manual" (impostazione predefinita) a "Automatic (avvio ritardato)" per ridurre il tempo di avvio iniziale delle applicazioni WPF.

## Vedere anche

- [Pianificazione delle prestazioni dell'applicazione](#)
- [Sfruttare appieno l'hardware](#)

- Ottimizzazione delle prestazioni: layout e progettazione
- Grafica bidimensionale e creazione di immagini
- Comportamento dell'oggetto
- Risorse di applicazioni
- per
- Data binding
- Suggerimenti sulle animazioni

# Tempo di avvio delle applicazioni

23/10/2019 • 15 minutes to read • [Edit Online](#)

La quantità di tempo necessaria per avviare un'applicazione WPF può variare notevolmente. In questo argomento vengono descritte varie tecniche per ridurre il tempo di avvio percepito ed effettivo per un'applicazione Windows Presentation Foundation (WPF).

## Informazioni su avvio a freddo e avvio a caldo

L'avvio a freddo si verifica quando un'applicazione viene avviata per la prima volta dopo un riavvio del sistema, o quando si avvia l'applicazione, la si chiude e quindi la si riavvia dopo un lungo periodo di tempo. All'avvio dell'applicazione, se le pagine necessarie (codice, dati statici, registro e così via) non sono presenti nell'elenco di standby del gestore della memoria di Windows, si verificano errori di pagina. L'accesso al disco è necessario per inserire le pagine in memoria.

L'avvio a caldo si verifica quando la maggior parte delle pagine per i componenti principali di Common Language Runtime (CLR) sono già caricata in memoria, il che consente di risparmiare molto tempo per l'accesso al disco. Ecco perché un'applicazione gestita si avvia più rapidamente quando viene eseguita per la seconda volta.

## Implementare la schermata iniziale

Nei casi in cui c'è un ritardo significativo e inevitabile tra l'avvio di un'applicazione e la visualizzazione della prima interfaccia utente, è possibile ottimizzare il tempo di avvio percepito usando una *schermata iniziale*. Questo approccio consente di visualizzare un'immagine quasi immediatamente dopo l'avvio dell'applicazione da parte dell'utente. Quando l'applicazione è pronta per la visualizzazione della prima interfaccia utente, la schermata iniziale si dissolve. A partire da .NET Framework 3.5 SP1, è possibile usare il [SplashScreen](#) classe per implementare la schermata iniziale. Vedere [Aggiungere una schermata iniziale in un'applicazione WPF](#).

È anche possibile implementare la schermata iniziale con grafica Win32 nativa. Visualizzare l'implementazione prima di [Run](#) viene chiamato il metodo.

## Analizzare il codice di avvio

Determinare il motivo di un avvio a freddo lento. La causa potrebbe essere il disco I/O, ma non sempre. In generale, è necessario ridurre l'uso di risorse esterne, ad esempio rete, servizi Web o disco.

Prima di eseguire il test, verificare che nessun'altra applicazione o servizio in esecuzione usi il codice gestito o WPF.

Avviare l'applicazione WPF immediatamente dopo un riavvio e determinare il tempo richiesto per la visualizzazione. Se tutti gli avvii successivi dell'applicazione (avvio a caldo) sono molto più veloci, il problema di avvio a freddo probabilmente è causato da I/O.

Se il problema di avvio a freddo dell'applicazione non è correlato all'I/O, è probabile che l'applicazione esegua una lunga inizializzazione o un calcolo, attenda il completamento di un evento o richieda una notevole quantità di compilazione JIT all'avvio. Le sezioni seguenti descrivono alcune di queste situazioni in maggior dettaglio.

## Ottimizzare il modulo di caricamento

Usare strumenti come Esplora processi (Procexp.exe) e Tlist.exe per determinare quali moduli l'applicazione carica. Il comando `Tlist <pid>` mostra tutti i moduli caricati da un processo.

Ad esempio, se non ci si sta connettendo al Web e si nota che System.Web.dll è stato caricato, vi è un modulo nell'applicazione che fa riferimento a questo assembly. Verificare che il riferimento sia necessario.

Se l'applicazione dispone di più moduli, unirli in un unico modulo. Questo approccio richiede un minor sovraccarico di caricamento di assembly del CLR. Un minor numero di assembly significa inoltre che CLR gestisce un minor numero di stati.

## Rinviare le operazioni di inizializzazione

È consigliabile posticipare il codice di inizializzazione dopo aver eseguito il rendering della finestra principale dell'applicazione.

Tenere presente che l'inizializzazione può essere eseguita all'interno di un costruttore di classe, e se il codice di inizializzazione fa riferimento ad altre classi, può causare un effetto a catena in cui vengono eseguiti molti costruttori di classi.

## Evitare la configurazione dell'applicazione

È consigliabile evitare la configurazione dell'applicazione. Ad esempio, se un'applicazione ha requisiti di configurazione semplici e tempi di avvio stretti, le voci del registro o di un semplice file INI potrebbero essere un'alternativa di avvio più veloce.

## Usare il GAC

Se un assembly non è installato nella Global Assembly Cache (GAC), ci sono ritardi causati dalla verifica hash di assembly con nome sicuro e convalida dell'immagine Ngen se un'immagine nativa per tale assembly è disponibile nel computer. La verifica del nome sicuro viene ignorata per tutti gli assembly installati nella GAC. Per altre informazioni, vedere [Gacutil.exe \(Global Assembly Cache Tool\)](#) (Strumento Global Assembly Cache, Gacutil.exe).

## Usare Ngen.exe

È consigliabile usare il generatore di immagini native (Ngen.exe) nell'applicazione. L'uso di Ngen.exe indica la negoziazione del consumo di CPU per un maggiore accesso al disco perché l'immagine nativa generata da Ngen.exe è probabilmente più grande dell'immagine MSIL.

Per migliorare il tempo di avvio a caldo, si deve usare sempre Ngen.exe sull'applicazione, poiché ciò consente di evitare il costo della CPU per la compilazione JIT del codice dell'applicazione.

In alcuni scenari di avvio a freddo, l'uso di Ngen.exe può essere utile, perché il compilatore JIT (mscorjit.dll) non deve essere caricato.

La presenza di entrambi i moduli Ngen e JIT può avere effetti negativi, perché è necessario caricare mscorejit.dll e quando il compilatore JIT opera sul codice, l'accesso a molte pagine nelle immagini Ngen deve avvenire quando il compilatore JIT legge i metadati degli assembly.

### ClickOnce e Ngen

Anche il modo in cui si prevede di distribuire l'applicazione può fare la differenza in fase di caricamento. Distribuzione di applicazioni ClickOnce non supporta Ngen. Se si decide di usare Ngen.exe per l'applicazione, è necessario usare un altro meccanismo di distribuzione, ad esempio Windows Installer.

Per altre informazioni, vedere [Ngen.exe \(Native Image Generator\)](#).

### Riassegnazione e conflitti di indirizzi di DLL

Se si usa Ngen.exe, tenere presente che la riassegnazione può verificarsi quando le immagini native vengono caricate in memoria. Se una DLL non viene caricata all'indirizzo di base preferito perché tale intervallo di indirizzi è già stato allocato, il caricatore di Windows lo caricherà in un altro indirizzo. Questa può essere un'operazione

impegnativa.

È possibile usare lo strumento Virtual Address Dump (Vadump.exe) per verificare se sono presenti moduli in cui tutte le pagine sono private. In questo caso, il modulo potrebbe stato riassegnato a un indirizzo diverso. Di conseguenza, le pagine non possono essere condivise.

Per altre informazioni su come impostare l'indirizzo di base, vedere [Ngen.exe \(Native Image Generator\)](#).

## Ottimizzare Authenticode

La verifica di Authenticode si aggiunge al tempo di avvio. Gli assembly con firma Authenticode devono essere verificati con l'autorità di certificazione (CA). Questa verifica può richiedere tempi lunghi, in quanto può essere necessario connettersi alla rete più volte per scaricare gli elenchi di revoca dei certificati attuali. Consente inoltre la presenza di una catena completa di certificati validi nel percorso a una fonte attendibile. Questo può causare molti secondi di ritardo, mentre l'assembly viene caricato.

Si consiglia di installare il certificato CA nel computer client oppure di evitare di usare Authenticode quando è possibile. Se per l'applicazione non è richiesta la prova del server di pubblicazione, non è necessario pagare il costo della verifica della firma.

A partire da .NET Framework 3.5, è presente un'opzione di configurazione che consente di ignorare la verifica Authenticode. A tale scopo, aggiungere la seguente impostazione nel file app.exe.config:

```
<configuration>
 <runtime>
 <generatePublisherEvidence enabled="false"/>
 </runtime>
</configuration>
```

Per ulteriori informazioni, vedere [Elemento <generatePublisherEvidence>](#).

## Confrontare le prestazioni in Windows Vista

Il gestore della memoria in Windows Vista è una tecnologia denominata SuperFetch. SuperFetch analizza i modelli di uso della memoria nel tempo per determinare il contenuto della memoria ottimale per un utente specifico. Funziona in modo continuo per garantire la disponibilità del contenuto in qualsiasi momento.

Questo approccio è diverso dalla tecnica di recupero preliminare usata in Windows XP, che consente di precaricare i dati in memoria senza analisi dei modelli di uso. Nel corso del tempo, se l'utente usa spesso l'applicazione WPF in Windows Vista, il tempo di avvio a freddo dell'applicazione può migliorare.

## Usare in modo efficiente AppDomain

Se possibile, caricare gli assembly in un'area di codice indipendente dal dominio per assicurarsi che l'immagine nativa, se presente, venga usata in tutti gli AppDomain creati nell'applicazione.

Per prestazioni ottimali, applicare una comunicazione efficiente tra domini, riducendo le chiamate tra domini. Quando possibile, usare le chiamate senza argomenti oppure con argomenti di tipo primitivo.

## Usare l'attributo NeutralResourcesLanguage

Usare la [NeutralResourcesLanguageAttribute](#) per specificare le impostazioni cultura neutrali per il [ResourceManager](#). Questo approccio impedisce ricerche di assembly non riuscite.

## Usare la classe BinaryFormatter per la serializzazione

Se è necessario utilizzare la serializzazione, usare il [BinaryFormatter](#) classe anziché il [XmlSerializer](#) classe. Il [BinaryFormatter](#) classe è implementata nella libreria di classe di Base (BCL) nell'assembly mscorlib.dll. Il [XmlSerializer](#) viene implementata nell'assembly XML.dll, che potrebbe essere una DLL aggiuntiva da caricare.

Se è necessario usare il [XmlSerializer](#) (classe), è possibile ottenere prestazioni migliori se si genera prima l'assembly di serializzazione.

## Configurare ClickOnce per verificare gli aggiornamenti in seguito all'avvio

Se l'applicazione utilizza ClickOnce, evitare l'accesso alla rete all'avvio configurando ClickOnce per verificare il sito di distribuzione degli aggiornamenti dopo l'avvio dell'applicazione.

Se si usa il modello XAML browser application (XBAP), tenere presente che ClickOnce controlla il sito di distribuzione degli aggiornamenti anche se l'applicazione XBAP è già nella cache di ClickOnce. Per altre informazioni, vedere [ClickOnce Security and Deployment](#).

## Configurare l'avvio automatico del servizio PresentationFontCache

La prima applicazione WPF da eseguire dopo un riavvio è il servizio PresentationFontCache. Il servizio memorizza nella cache i tipi di carattere del sistema, migliora l'accesso al tipo di carattere e le prestazioni complessive. Si verifica un sovraccarico all'avvio del servizio e in alcuni ambienti controllati, si consiglia di configurare l'avvio automatico del servizio al riavvio del sistema.

## Impostare il data binding a livello di codice

Anziché usare XAML per impostare il [DataContext](#) in modo dichiarativo per la finestra principale, è consigliabile impostarlo a livello di codice nel [OnActivated](#) (metodo).

## Vedere anche

- [SplashScreen](#)
- [AppDomain](#)
- [NeutralResourcesLanguageAttribute](#)
- [ResourceManager](#)
- [Aggiungere una schermata iniziale in un'applicazione WPF](#)
- [Ngen.exe \(generatore di immagini native\)](#)
- [Elemento <generatePublisherEvidence>](#)

# Procedura dettagliata: memorizzazione dei dati di un'applicazione nella cache di un'applicazione WPF

03/02/2020 • 18 minutes to read • [Edit Online](#)

La memorizzazione nella cache consente di inserire i dati in memoria per l'accesso rapido. Quando accedono nuovamente ai dati, le applicazioni possono recuperarli dalla cache anziché dall'origine. In questo modo si possono ottenere migliori prestazioni e scalabilità. Inoltre, se si memorizzano i dati nella cache, questi sono accessibili anche quando l'origine dati è temporaneamente non disponibile.

Il .NET Framework fornisce classi che consentono di usare la memorizzazione nella cache nelle applicazioni .NET Framework. Queste classi si trovano nello spazio dei nomi [System.Runtime.Caching](#).

## NOTE

Lo spazio dei nomi [System.Runtime.Caching](#) è nuovo in .NET Framework 4. Questo spazio dei nomi rende disponibile la memorizzazione nella cache per tutte le applicazioni .NET Framework. Nelle versioni precedenti del .NET Framework, la memorizzazione nella cache era disponibile solo nello spazio dei nomi [System.Web](#) e pertanto richiedeva una dipendenza dalle classi ASP.NET.

In questa procedura dettagliata viene illustrato come utilizzare la funzionalità di memorizzazione nella cache disponibile nel .NET Framework come parte di un'applicazione Windows Presentation Foundation (WPF). Nella procedura dettagliata viene memorizzato nella cache il contenuto di un file di testo.

Di seguito vengono illustrate le attività incluse nella procedura dettagliata:

- Creazione di un progetto di applicazione WPF.
- Aggiunta di un riferimento al .NET Framework 4.
- Inizializzazione di una cache.
- Aggiunta di una voce della cache che contiene il contenuto di un file di testo.
- Fornire un criterio di rimozione per la voce della cache.
- Monitoraggio del percorso del file memorizzato nella cache e notifica all'istanza della cache le modifiche apportate all'elemento monitorato.

## Prerequisiti

Per completare questa procedura dettagliata, è necessario:

- Visual Studio 2010.
- Un file di testo che contiene una piccola quantità di testo. Il contenuto del file di testo viene visualizzato in una finestra di messaggio. Il codice illustrato nella procedura dettagliata presuppone che si stia utilizzando il file seguente:

c:\cache\cacheText.txt

Tuttavia, è possibile usare qualsiasi file di testo e apportare piccole modifiche al codice in questa procedura dettagliata.

# Creazione di un progetto di applicazione WPF

Si inizierà creando un progetto di applicazione WPF.

## Per creare un'applicazione WPF

1. Avviare Visual Studio.
2. Scegliere **nuovo** dal menu **file**, quindi fare clic su **nuovo progetto**.

La finestra di dialogo **Nuovo progetto** viene visualizzata.

3. In **modelli installati** selezionare il linguaggio di programmazione che si desidera utilizzare (**Visual Basic** o **oggetto C#visivo** ).
4. Nella finestra di dialogo **nuovo progetto** selezionare **applicazione WPF**.

### NOTE

Se il modello **applicazione WPF** non è visibile, assicurarsi che la destinazione sia una versione del .NET Framework che supporta WPF. Nella finestra di dialogo **nuovo progetto** selezionare .NET Framework 4 nell'elenco.

5. Nella casella di testo **nome** immettere un nome per il progetto. Ad esempio, è possibile immettere **WPFCaching**.
6. Selezionare la casella di controllo **Crea directory per soluzione**.
7. Fare clic su **OK**.

WPF Designer viene aperto in visualizzazione **progettazione** e visualizza il file MainWindow. XAML.

Visual Studio crea la cartella del **progetto**, il file Application. XAML e il file MainWindow. XAML.

## Destinazione della .NET Framework e aggiunta di un riferimento agli assembly di memorizzazione nella cache

Per impostazione predefinita, le applicazioni WPF sono destinate al profilo client di .NET Framework 4. Per utilizzare lo spazio dei nomi [System.Runtime.Caching](#) in un'applicazione WPF, è necessario che l'applicazione sia destinata al .NET Framework 4 (non al profilo client .NET Framework 4) e che includa un riferimento allo spazio dei nomi.

Il passaggio successivo consiste quindi nel modificare la destinazione .NET Framework e aggiungere un riferimento allo spazio dei nomi [System.Runtime.Caching](#).

### NOTE

La procedura per la modifica della destinazione .NET Framework è diversa in un progetto Visual Basic e in un C# progetto visuale.

### Per modificare la .NET Framework di destinazione in Visual Basic

1. In **Esplora soluzioni** fare clic con il pulsante destro del mouse sul nome del progetto, quindi scegliere **Proprietà**.

Verrà visualizzata la finestra proprietà per l'applicazione.

2. Fare clic sulla scheda **Compila**.
3. Nella parte inferiore della finestra fare clic su **Opzioni di compilazione avanzate**.

Viene visualizzata la finestra di dialogo **impostazioni del compilatore avanzate**.

4. Nell'elenco **Framework di destinazione (tutte le configurazioni)** selezionare .NET Framework 4. (Non selezionare .NET Framework 4 profilo client).

5. Fare clic su **OK**.

Viene visualizzata la finestra di dialogo **Modifica del framework di destinazione**.

6. Nella finestra di dialogo **modifica Framework di destinazione** fare clic su **Sì**.

Il progetto viene chiuso e quindi riaperto.

7. Aggiungere un riferimento all'assembly di Caching attenendosi alla procedura seguente:

a. In **Esplora soluzioni** fare clic con il pulsante destro del mouse sul nome del progetto, quindi scegliere **Aggiungi riferimento**.

b. Selezionare la scheda **.NET**, selezionare **System.Runtime.Caching** e quindi fare clic su **OK**.

**Per modificare la .NET Framework di destinazione in un C# progetto Visual**

1. In **Esplora soluzioni** fare clic con il pulsante destro del mouse sul nome del progetto, quindi scegliere **Proprietà**.

Verrà visualizzata la finestra proprietà per l'applicazione.

2. Fare clic sulla scheda **Applicazione**.

3. Nell'elenco **Framework di destinazione** selezionare .NET Framework 4. (Non selezionare **.NET Framework 4 profilo client**).

4. Aggiungere un riferimento all'assembly di Caching attenendosi alla procedura seguente:

a. Fare clic con il pulsante destro del mouse sulla cartella **riferimenti**, quindi scegliere **Aggiungi riferimento**.

b. Selezionare la scheda **.NET**, selezionare **System.Runtime.Caching** e quindi fare clic su **OK**.

## Aggiunta di un pulsante alla finestra WPF

Successivamente, si aggiungerà un controllo Button e si creerà un gestore eventi per l'evento **Click** del pulsante.

Successivamente, quando si fa clic sul pulsante viene aggiunto il codice, il contenuto del file di testo viene memorizzato nella cache e visualizzato.

**Per aggiungere un controllo Button**

1. In **Esplora soluzioni** fare doppio clic sul file MainWindow.xaml per aprirlo.

2. Dalla **casella degli strumenti**, sotto **controlli WPF comuni** trascinare un controllo **Button** nella finestra di **MainWindow**.

3. Nella finestra **Proprietà** impostare la proprietà **Content** del controllo **Button** per **ottenere la cache**.

## Inizializzazione della cache e memorizzazione nella cache di una voce

Successivamente, verrà aggiunto il codice per eseguire le attività seguenti:

- Creare un'istanza della classe cache, ovvero si creerà un'istanza di un nuovo oggetto **MemoryCache**.
- Consente di specificare che la cache utilizza un oggetto **HostFileChangeMonitor** per monitorare le modifiche nel file di testo.
- Leggere il file di testo e memorizzare nella cache il relativo contenuto come voce della cache.

- Visualizza il contenuto del file di testo memorizzato nella cache.

**Per creare l'oggetto cache**

1. Fare doppio clic sul pulsante appena aggiunto per creare un gestore eventi nel file MainWindow.xaml.cs o MainWindow. XAML. vb.
2. Nella parte superiore del file (prima della dichiarazione di classe) aggiungere le seguenti istruzioni **Imports** (Visual Basic) o **using** (C#):

```
using System.Runtime.Caching;
using System.IO;
```

```
Imports System.Runtime.Caching
Imports System.IO
```

3. Nel gestore eventi aggiungere il codice seguente per creare un'istanza dell'oggetto cache:

```
ObjectCache cache = MemoryCache.Default;
```

```
Dim cache As ObjectCache = MemoryCache.Default
```

La classe **ObjectCache** è una classe incorporata che fornisce una cache di oggetti in memoria.

4. Aggiungere il codice seguente per leggere il contenuto di una voce della cache denominata **filecontents**:

```
Dim fileContents As String = TryCast(cache("filecontents"), String)
```

```
string fileContents = cache["filecontents"] as string;
```

5. Aggiungere il codice seguente per verificare se la voce della cache denominata **filecontents** esiste:

```
If fileContents Is Nothing Then
End If
```

```
if (fileContents == null)
{
```

Se la voce della cache specificata non esiste, è necessario leggere il file di testo e aggiungerlo come voce della cache alla cache.

6. Nel blocco **if/then** aggiungere il codice seguente per creare un nuovo oggetto **CacheItemPolicy** che specifichi che la voce della cache scade dopo 10 secondi.

```
Dim policy As New CacheItemPolicy()
policy.AbsoluteExpiration = DateTimeOffset.Now.AddSeconds(10.0)
```

```
CacheItemPolicy policy = new CacheItemPolicy();
policy.AbsoluteExpiration = DateTimeOffset.Now.AddSeconds(10.0);
```

Se non viene fornita alcuna informazione di rimozione o scadenza, il valore predefinito è [InfiniteAbsoluteExpiration](#), il che significa che le voci della cache non scadono mai in base a un tempo assoluto. Al contrario, le voci della cache scadono solo quando si verifica un numero eccessivo di richieste di memoria. Come procedura consigliata, è consigliabile specificare sempre in modo esplicito una scadenza assoluta o variabile.

7. All'interno del blocco `if/then` e dopo il codice aggiunto nel passaggio precedente, aggiungere il codice seguente per creare una raccolta per i percorsi di file che si desidera monitorare e per aggiungere il percorso del file di testo alla raccolta:

```
Dim filePaths As New List(Of String)()
filePaths.Add("c:\cache\cacheText.txt")
```

```
List<string> filePaths = new List<string>();
filePaths.Add("c:\\cache\\cacheText.txt");
```

#### NOTE

Se il file di testo che si desidera utilizzare non è `c:\cache\cacheText.txt`, specificare il percorso in cui si desidera utilizzare il file di testo.

8. Dopo il codice aggiunto nel passaggio precedente, aggiungere il codice seguente per aggiungere un nuovo oggetto [HostFileChangeMonitor](#) alla raccolta di monitoraggi delle modifiche per la voce della cache:

```
policy.ChangeMonitors.Add(New HostFileChangeMonitor(filePaths))
```

```
policy.ChangeMonitors.Add(new HostFileChangeMonitor(filePaths));
```

L'oggetto [HostFileChangeMonitor](#) monitora il percorso del file di testo e invia una notifica alla cache se si verificano modifiche. In questo esempio, la voce della cache scadrà se il contenuto del file viene modificato.

9. Dopo il codice aggiunto nel passaggio precedente, aggiungere il codice seguente per leggere il contenuto del file di testo:

```
fileContents = File.ReadAllText("c:\cache\cacheText.txt") & vbCrLf & DateTime.Now.ToString()
```

```
fileContents = File.ReadAllText("c:\\cache\\cacheText.txt") + "\n" + DateTime.Now;
```

Viene aggiunto il timestamp di data e ora in modo che sia possibile visualizzare la scadenza della voce della cache.

10. Dopo il codice aggiunto nel passaggio precedente, aggiungere il codice seguente per inserire il contenuto del file nell'oggetto cache come istanza di [CacheItem](#):

```
cache.Set("filecontents", fileContents, policy)
```

```
cache.Set("filecontents", fileContents, policy);
```

Specificare le informazioni sulla modalità di eliminazione della voce della cache passando l'oggetto [CacheItemPolicy](#) creato in precedenza come parametro.

11. Dopo il blocco `if/then` aggiungere il codice seguente per visualizzare il contenuto del file memorizzato nella cache in una finestra di messaggio:

```
MessageBox.Show(fileContents)
```

```
MessageBox.Show(fileContents);
```

12. Scegliere **Compila WPF Caching** dal menu **Compila** per compilare il progetto.

## Test della memorizzazione nella cache nell'applicazione WPF

È ora possibile eseguire il test dell'applicazione.

### Per testare la memorizzazione nella cache nell'applicazione WPF

1. Premere CTRL+F5 per eseguire l'applicazione.

Viene visualizzata la finestra `MainWindow`.

2. Fare clic su **Ottieni cache**.

Il contenuto memorizzato nella cache del file di testo viene visualizzato in una finestra di messaggio. Si noti il timestamp del file.

3. Chiudere la finestra di messaggio e quindi fare di nuovo clic su **Get cache**.

Il timestamp è invariato. Indica che viene visualizzato il contenuto memorizzato nella cache.

4. Attendere 10 secondi o più, quindi fare di nuovo clic su **Ottieni cache**.

Questa volta viene visualizzato un nuovo timestamp. Ciò indica che i criteri consentono la scadenza della voce della cache e che viene visualizzato il nuovo contenuto memorizzato nella cache.

5. In un editor di testo aprire il file di testo creato. Non apportare alcuna modifica.

6. Chiudere la finestra di messaggio e quindi fare di nuovo clic su **Get cache**.

Si noti di nuovo il timestamp.

7. Apportare una modifica al file di testo e quindi salvare il file.

8. Chiudere la finestra di messaggio e quindi fare di nuovo clic su **Get cache**.

Questa finestra di messaggio contiene il contenuto aggiornato del file di testo e un nuovo timestamp. Ciò indica che il monitoraggio delle modifiche del file host ha rimosso la voce della cache immediatamente quando è stato modificato il file, anche se il periodo di timeout assoluto non è scaduto.

### NOTE

È possibile aumentare il tempo di rimozione a 20 secondi o più per consentire più tempo per apportare modifiche al file.

## Esempio di codice

Al termine di questa procedura dettagliata, il codice per il progetto creato sarà simile all'esempio seguente.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Runtime.Caching;
using System.IO;

namespace WPFCaching
{
 /// <summary>
 /// Interaction logic for MainWindow.xaml
 /// </summary>
 public partial class MainWindow : Window
 {
 public MainWindow()
 {
 InitializeComponent();
 }

 private void button1_Click(object sender, RoutedEventArgs e)
 {

 ObjectCache cache = MemoryCache.Default;
 string fileContents = cache["filecontents"] as string;

 if (fileContents == null)
 {
 CacheItemPolicy policy = new CacheItemPolicy();
 policy.AbsoluteExpiration =
 DateTimeOffset.Now.AddSeconds(10.0);

 List<string> filePaths = new List<string>();
 filePaths.Add("c:\\cache\\cacheText.txt");

 policy.ChangeMonitors.Add(new
 HostFileChangeMonitor(filePaths));

 // Fetch the file contents.
 fileContents = File.ReadAllText("c:\\cache\\cacheText.txt") + "\n" + DateTime.Now.ToString();

 cache.Set("filecontents", fileContents, policy);
 }
 MessageBox.Show(fileContents);
 }
 }
}
```

```
Imports System.Runtime.Caching
Imports System.IO

Class MainWindow

 Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.Windows.RoutedEventArgs)
Handles Button1.Click
 Dim cache As ObjectCache = MemoryCache.Default
 Dim fileContents As String = TryCast(cache("filecontents"), _
 String)

 If fileContents Is Nothing Then
 Dim policy As New CacheItemPolicy()
 policy.AbsoluteExpiration = _
 DateTimeOffset.Now.AddSeconds(10.0)
 Dim filePaths As New List(Of String)()
 filePaths.Add("c:\cache\cacheText.txt")
 policy.ChangeMonitors.Add(New _
 HostFileChangeMonitor(filePaths))

 ' Fetch the file contents.
 fileContents = File.ReadAllText("c:\cache\cacheText.txt") & vbCrLf & DateTime.Now.ToString()
 cache.Set("filecontents", fileContents, policy)
 End If
 MessageBox.Show(fileContents)
 End Sub
End Class
```

## Vedere anche

- [MemoryCache](#)
- [ObjectCache](#)
- [System.Runtime.Caching](#)
- [Memorizzazione nella cache in applicazioni .NET Framework](#)

# Modello di threading

10/02/2020 • 40 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) è stato progettato per semplificare il threading. Di conseguenza, la maggior parte dei WPF sviluppatori non dovranno scrivere un'interfaccia che usa più di un thread. Poiché i programmi con multithreading sono complessi ed è difficile eseguirne il debug, è preferibile evitarli quando sono disponibili soluzioni a thread singolo.

Indipendentemente da quanto ben progettato, tuttavia, nessun framework di Interfaccia utente sarà in grado di fornire una soluzione a thread singolo per ogni tipo di problema. WPF si avvicina, ma esistono ancora situazioni in cui più thread migliorano interfaccia utente la velocità di risposta o le prestazioni dell'applicazione. Dopo aver illustrato alcune nozioni di base, questo documento analizza alcune di queste situazioni per concludere con la descrizione di alcuni aspetti più in dettaglio.

## NOTE

In questo argomento viene illustrato il threading utilizzando il metodo `BeginInvoke` per le chiamate asincrone. È anche possibile effettuare chiamate asincrone chiamando il metodo `InvokeAsync`, che accetta un `Action` o `Func<TResult>` come parametro. Il metodo `InvokeAsync` restituisce un `DispatcherOperation` o `DispatcherOperation<TResult>`, che dispone di una proprietà `Task`. È possibile usare la parola chiave `await` con il `DispatcherOperation` o il `Task` associato. Se è necessario attendere in modo sincrono il `Task` restituito da un `DispatcherOperation` o `DispatcherOperation<TResult>`, chiamare il metodo di estensione `DispatcherOperationWait`. La chiamata di `Task.Wait` comporterà un deadlock. Per ulteriori informazioni sull'utilizzo di un `Task` per eseguire operazioni asincrone, vedere Parallelismo delle attività. Il metodo `Invoke` dispone inoltre di overload che accettano un `Action` o `Func<TResult>` come parametro. È possibile usare il metodo `Invoke` per eseguire chiamate sincrone passando un delegato, `Action` o `Func<TResult>`.

## Panoramica e dispatcher

In genere, WPF applicazioni iniziano con due thread: uno per la gestione del rendering e l'altro per la gestione della Interfaccia utente. Il thread di rendering viene eseguito in modo efficace nascosto in background mentre il thread Interfaccia utente riceve input, gestisce gli eventi, disegna lo schermo ed esegue il codice dell'applicazione. La maggior parte delle applicazioni usa un singolo thread di Interfaccia utente, anche se in alcune situazioni è preferibile usare diversi. Questo aspetto verrà spiegato più avanti con un esempio.

Il thread di Interfaccia utente Accoda gli elementi di lavoro all'interno di un oggetto denominato `Dispatcher`. L'oggetto `Dispatcher` seleziona gli elementi di lavoro in base alla priorità ed esegue ciascuno fino al completamento. Ogni thread di Interfaccia utente deve avere almeno una `Dispatcher`; ogni `Dispatcher` può eseguire elementi di lavoro in un solo thread.

Il trucco per la creazione di applicazioni reattive e intuitive è ottimizzare la velocità effettiva `Dispatcher` mantenendo gli elementi di lavoro ridotti. In questo modo gli elementi non vengono mai aggiornati nella coda `Dispatcher` in attesa di elaborazione. Qualsiasi ritardo percepibile tra input e risposta può causare frustrazione in un utente.

In che modo le applicazioni WPF dovrebbero gestire operazioni di grandi dimensioni? Cosa è necessario fare se il codice prevede calcoli complessi o l'esecuzione di query su un database in un server remoto? In genere, la risposta consiste nel gestire la grande operazione in un thread separato, lasciando il Interfaccia utente thread libero per tendere agli elementi nella coda di `Dispatcher`. Al termine della grande operazione, può segnalare il risultato al thread Interfaccia utente per la visualizzazione.

In passato, Windows consente di accedere agli elementi di Interfaccia utente solo dal thread che li ha creati. Ciò

significava che un thread in background responsabile di un'attività a esecuzione prolungata non poteva aggiornare una casella di testo dopo il suo completamento. Questa operazione viene eseguita da Windows per garantire l'integrità dei componenti di Interfaccia utente. Una casella di riepilogo può apparire strana se il relativo contenuto viene aggiornato da un thread in background mentre viene disegnata.

WPF include un meccanismo predefinito di esclusione reciproca che impone questo coordinamento. La maggior parte delle classi in WPF deriva da [DispatcherObject](#). In fase di costruzione, un [DispatcherObject](#) archivia un riferimento alla [Dispatcher](#) collegata al thread attualmente in esecuzione. In effetti, il [DispatcherObject](#) associa al thread che la crea. Durante l'esecuzione del programma, un [DispatcherObject](#) può chiamare il relativo metodo [VerifyAccess](#) pubblico. [VerifyAccess](#) esamina la [Dispatcher](#) associata al thread corrente e la confronta con il riferimento [Dispatcher](#) archiviato durante la costruzione. Se non corrispondono, [VerifyAccess](#) genera un'eccezione. [VerifyAccess](#) deve essere chiamato all'inizio di ogni metodo appartenente a una [DispatcherObject](#).

Se solo un thread può modificare la Interfaccia utente, in che modo i thread in background interagiscono con l'utente? Un thread in background può richiedere al thread Interfaccia utente di eseguire un'operazione per suo conto. Questa operazione viene eseguita registrando un elemento di lavoro con il [Dispatcher](#) del thread di Interfaccia utente. La classe [Dispatcher](#) fornisce due metodi per la registrazione degli elementi di lavoro: [Invoke](#) e [BeginInvoke](#). Entrambi i metodi pianificano un delegato per l'esecuzione. [Invoke](#) è una chiamata sincrona, ovvero non restituisce un risultato fino a quando il thread di Interfaccia utente non completa effettivamente l'esecuzione del delegato. [BeginInvoke](#) è asincrono e restituisce immediatamente un risultato.

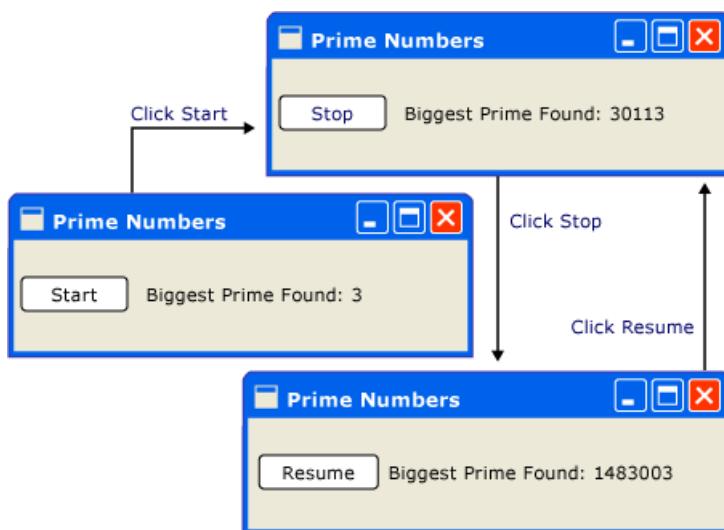
Il [Dispatcher](#) Ordina gli elementi nella coda per priorità. È possibile specificare dieci livelli quando si aggiunge un elemento alla coda [Dispatcher](#). Queste priorità vengono gestite nell'enumerazione [DispatcherPriority](#). Informazioni dettagliate sui livelli di [DispatcherPriority](#) sono disponibili nella documentazione di Windows SDK.

## Thread in azione: esempi

### Applicazione a thread singolo con calcolo a esecuzione prolungata

La maggior parte dell'interfaccia utente grafica (GUI) dedica una grande parte del tempo di inattività durante l'attesa degli eventi generati in risposta alle interazioni dell'utente. Con un'attenta programmazione questo tempo di inattività può essere usato in modo costruttivo, senza influire sulla velocità di risposta del Interfaccia utente. Il modello di threading WPF non consente all'input di interrompere un'operazione che si verifica nel thread di Interfaccia utente. Ciò significa che è necessario assicurarsi di tornare al [Dispatcher](#) periodicamente per elaborare gli eventi di input in sospeso prima di diventare obsoleti.

Prendere in considerazione gli esempi seguenti:



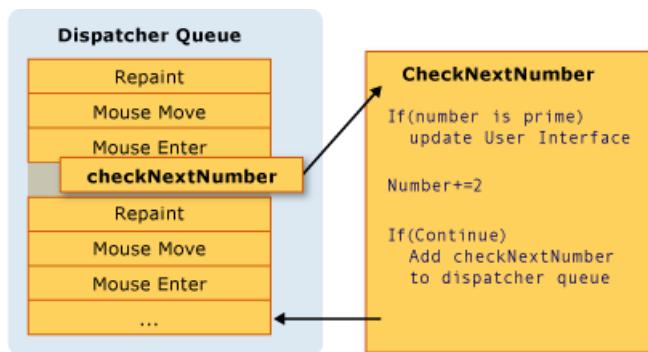
Questa semplice applicazione conta verso l'alto a partire da tre, cercando i numeri primi. Quando l'utente fa clic sul pulsante **Start**, inizia la ricerca. Quando il programma trova un numero primo, aggiorna l'interfaccia utente con il risultato trovato. In qualsiasi momento, l'utente può interrompere la ricerca.

Anche se è abbastanza semplice, la ricerca di numeri primi potrebbe continuare all'infinito e questo comporta alcune difficoltà. Se l'intera ricerca è stata gestita all'interno del gestore dell'evento click del pulsante, non si darebbe mai al thread di Interfaccia utente la possibilità di gestire altri eventi. Il Interfaccia utente non sarà in grado di rispondere ai messaggi di input o di elaborazione. L'interfaccia utente non verrebbe mai ridisegnata e i clic sui pulsanti non riceverebbero alcuna risposta.

È possibile eseguire la ricerca dei numeri primi in un thread separato, ma in questo caso sarebbe necessario affrontare problemi di sincronizzazione. Con un approccio a thread singolo, è possibile aggiornare direttamente l'etichetta che indica il numero primo più grande trovato.

Se si suddivide l'attività di calcolo in blocchi gestibili, è possibile tornare periodicamente al [Dispatcher](#) ed elaborare gli eventi. Possiamo dare WPF la possibilità di ridisegnare ed elaborare l'input.

Il modo migliore per suddividere i tempi di elaborazione tra calcolo e gestione degli eventi consiste nel gestire il calcolo dal [Dispatcher](#). Utilizzando il metodo [BeginInvoke](#), è possibile pianificare i controlli dei numeri primi nella stessa coda da cui vengono tracciati Interfaccia utente eventi. Nell'esempio viene pianificato il controllo di un singolo numero primo per volta. Al termine del controllo del numero primo, viene pianificato immediatamente il controllo successivo. Questo controllo continua solo dopo che sono stati gestiti gli eventi Interfaccia utente in sospeso.



Microsoft Word realizza il controllo ortografico usando questo meccanismo. Il controllo ortografico viene eseguito in background utilizzando il tempo di inattività del thread Interfaccia utente. Di seguito è riportato il codice.

L'esempio seguente mostra il codice XAML che crea l'interfaccia utente.

```

<Window x:Class="SDKSamples.Window1"
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Prime Numbers" Width="260" Height="75"
 >
 <StackPanel Orientation="Horizontal" VerticalAlignment="Center" >
 <Button Content="Start"
 Click="StartOrStop"
 Name="startStopButton"
 Margin="5,0,5,0"
 />
 <TextBlock Margin="10,5,0,0">Biggest Prime Found:</TextBlock>
 <TextBlock Name="bigPrime" Margin="4,5,0,0">3</TextBlock>
 </StackPanel>
</Window>

```

L'esempio seguente mostra il code-behind.

```

using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Threading;
using System.Threading;

```

```

using System.Threading;

namespace SDKSamples
{
 public partial class Window1 : Window
 {
 public delegate void NextPrimeDelegate();

 //Current number to check
 private long num = 3;

 private bool continueCalculating = false;

 public Window1() : base()
 {
 InitializeComponent();
 }

 private void StartOrStop(object sender, EventArgs e)
 {
 if (continueCalculating)
 {
 continueCalculating = false;
 startStopButton.Content = "Resume";
 }
 else
 {
 continueCalculating = true;
 startStopButton.Content = "Stop";
 startStopButton.Dispatcher.BeginInvoke(
 DispatcherPriority.Normal,
 new NextPrimeDelegate(CheckNextNumber));
 }
 }

 public void CheckNextNumber()
 {
 // Reset flag.
 NotAPrime = false;

 for (long i = 3; i <= Math.Sqrt(num); i++)
 {
 if (num % i == 0)
 {
 // Set not a prime flag to true.
 NotAPrime = true;
 break;
 }
 }

 // If a prime number.
 if (!NotAPrime)
 {
 bigPrime.Text = num.ToString();
 }

 num += 2;
 if (continueCalculating)
 {
 startStopButton.Dispatcher.BeginInvoke(
 System.Windows.Threading.DispatcherPriority.SystemIdle,
 new NextPrimeDelegate(this.CheckNextNumber));
 }
 }

 private bool NotAPrime = false;
 }
}

```

```

Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Threading
Imports System.Threading

Namespace SDKSamples
 Partial Public Class MainWindow
 Inherits Window
 Public Delegate Sub NextPrimeDelegate()

 'Current number to check
 Private num As Long = 3

 Private continueCalculating As Boolean = False

 Public Sub New()
 MyBase.New()
 InitializeComponent()
 End Sub

 Private Sub StartOrStop(ByVal sender As Object, ByVal e As EventArgs)
 If continueCalculating Then
 continueCalculating = False
 startStopButton.Content = "Resume"
 Else
 continueCalculating = True
 startStopButton.Content = "Stop"
 startStopButton.Dispatcher.BeginInvoke(DispatcherPriority.Normal, New
NextPrimeDelegate(AddressOf CheckNextNumber))
 End If
 End Sub

 Public Sub CheckNextNumber()
 ' Reset flag.
 NotAPrime = False

 For i As Long = 3 To Math.Sqrt(num)
 If num Mod i = 0 Then
 ' Set not a prime flag to true.
 NotAPrime = True
 Exit For
 End If
 Next

 ' If a prime number.
 If Not NotAPrime Then
 bigPrime.Text = num.ToString()
 End If

 num += 2
 If continueCalculating Then

 startStopButton.Dispatcher.BeginInvoke(System.Windows.Threading.DispatcherPriority.SystemIdle, New
NextPrimeDelegate(AddressOf Me.CheckNextNumber))
 End If
 End Sub

 Private NotAPrime As Boolean = False
 End Class
End Namespace

```

Nell'esempio seguente viene illustrato il gestore eventi per la [Button](#).

```

private void StartOrStop(object sender, EventArgs e)
{
 if (continueCalculating)
 {
 continueCalculating = false;
 startStopButton.Content = "Resume";
 }
 else
 {
 continueCalculating = true;
 startStopButton.Content = "Stop";
 startStopButton.Dispatcher.BeginInvoke(
 DispatcherPriority.Normal,
 new NextPrimeDelegate(CheckNextNumber));
 }
}

```

```

Private Sub StartOrStop(ByVal sender As Object, ByVal e As EventArgs)
 If continueCalculating Then
 continueCalculating = False
 startStopButton.Content = "Resume"
 Else
 continueCalculating = True
 startStopButton.Content = "Stop"
 startStopButton.Dispatcher.BeginInvoke(DispatcherPriority.Normal, New NextPrimeDelegate(AddressOf
CheckNextNumber))
 End If
End Sub

```

Oltre ad aggiornare il testo nella [Button](#), questo gestore è responsabile della pianificazione del primo controllo dei numeri primi mediante l'aggiunta di un delegato alla coda di [Dispatcher](#). A questo punto, dopo che il gestore dell'evento ha completato il proprio lavoro, il [Dispatcher](#) selezionerà questo delegato per l'esecuzione.

Come indicato in precedenza, [BeginInvoke](#) è il membro [Dispatcher](#) usato per pianificare un delegato per l'esecuzione. In questo caso, si sceglie la priorità [SystemIdle](#). Il [Dispatcher](#) eseguirà questo delegato solo quando non ci sono eventi importanti da elaborare. La velocità di risposta dell'Interfaccia utente è più importante del controllo dei numeri. Viene anche passato un nuovo delegato che rappresenta la routine di controllo dei numeri.

```

public void CheckNextNumber()
{
 // Reset flag.
 NotAPrime = false;

 for (long i = 3; i <= Math.Sqrt(num); i++)
 {
 if (num % i == 0)
 {
 // Set not a prime flag to true.
 NotAPrime = true;
 break;
 }
 }

 // If a prime number.
 if (!NotAPrime)
 {
 bigPrime.Text = num.ToString();
 }

 num += 2;
 if (continueCalculating)
 {
 startStopButton.Dispatcher.BeginInvoke(
 System.Windows.Threading.DispatcherPriority.SystemIdle,
 new NextPrimeDelegate(this.CheckNextNumber));
 }
}

private bool NotAPrime = false;

```

```

Public Sub CheckNextNumber()
 ' Reset flag.
 NotAPrime = False

 For i As Long = 3 To Math.Sqrt(num)
 If num Mod i = 0 Then
 ' Set not a prime flag to true.
 NotAPrime = True
 Exit For
 End If
 Next

 ' If a prime number.
 If Not NotAPrime Then
 bigPrime.Text = num.ToString()
 End If

 num += 2
 If continueCalculating Then
 startStopButton.Dispatcher.BeginInvoke(System.Windows.Threading.DispatcherPriority.SystemIdle, New
NextPrimeDelegate(AddressOf Me.CheckNextNumber))
 End If
End Sub

Private NotAPrime As Boolean = False

```

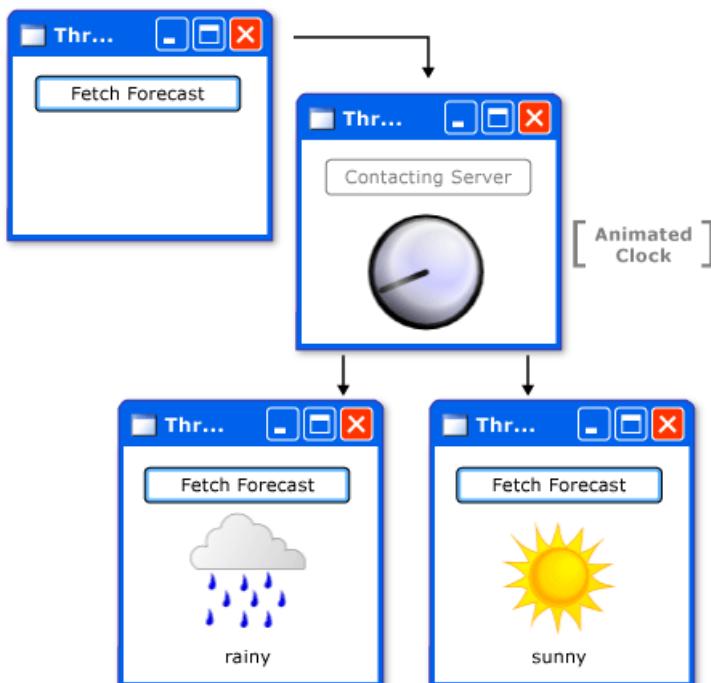
Questo metodo controlla se il numero dispari successivo è un numero primo. Se è primo, il metodo aggiorna direttamente il `bigPrime` `TextBlock` per rifletterne l'individuazione. Ciò è possibile perché il calcolo viene eseguito nello stesso thread usato per creare il componente. Se avessimo scelto di usare un thread separato per il calcolo, avremmo dovuto usare un meccanismo di sincronizzazione più complicato ed eseguire l'aggiornamento nel thread Interfaccia utente. Questa situazione verrà illustrata più avanti.

Per il codice sorgente completo per questo esempio, vedere l' [esempio di applicazione a thread singolo con calcolo a esecuzione prolungata](#)

### Gestione di un'operazione di blocco con un thread in background

La gestione delle operazioni di blocco in un'applicazione grafica può essere complessa. Non è consigliabile chiamare metodi di blocco dai gestori eventi perché l'applicazione risulterebbe bloccata. È possibile usare un thread separato per gestire queste operazioni, ma al termine è necessario eseguire la sincronizzazione con il thread di Interfaccia utente perché non è possibile modificare direttamente l'interfaccia utente grafica dal thread di lavoro. È possibile utilizzare [Invoke](#) o [BeginInvoke](#) per inserire delegati nel [Dispatcher](#) del thread Interfaccia utente. Alla fine, questi delegati verranno eseguiti con l'autorizzazione per modificare Interfaccia utente elementi.

In questo esempio viene simulata una chiamata RPC (Remote Procedure Call) che recupera i dati delle previsioni meteo. Viene usato un thread di lavoro separato per eseguire questa chiamata e si pianifica un metodo di aggiornamento nel [Dispatcher](#) del thread Interfaccia utente al termine dell'operazione.



```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Windows.Threading;
using System.Threading;

namespace SDKSamples
{
 public partial class Window1 : Window
 {
 // Delegates to be used in placing jobs onto the Dispatcher.
 private delegate void NoArgDelegate();
 private delegate void OneArgDelegate(String arg);

 // Storyboards for the animations.
 private Storyboard showClockFaceStoryboard;
 private Storyboard hideClockFaceStoryboard;
 private Storyboard showWeatherImageStoryboard;
 private Storyboard hideWeatherImageStoryboard;
```

```

public Window1(): base()
{
 InitializeComponent();
}

private void Window_Loaded(object sender, RoutedEventArgs e)
{
 // Load the storyboard resources.
 showClockFaceStoryboard =
 (Storyboard)this.Resources["ShowClockFaceStoryboard"];
 hideClockFaceStoryboard =
 (Storyboard)this.Resources["HideClockFaceStoryboard"];
 showWeatherImageStoryboard =
 (Storyboard)this.Resources["ShowWeatherImageStoryboard"];
 hideWeatherImageStoryboard =
 (Storyboard)this.Resources["HideWeatherImageStoryboard"];
}

private void ForecastButtonHandler(object sender, RoutedEventArgs e)
{
 // Change the status image and start the rotation animation.
 fetchButton.IsEnabled = false;
 fetchButton.Content = "Contacting Server";
 weatherText.Text = "";
 hideWeatherImageStoryboard.Begin(this);

 // Start fetching the weather forecast asynchronously.
 NoArgDelegate fetcher = new NoArgDelegate(
 this.FetchWeatherFromServer);

 fetcher.BeginInvoke(null, null);
}

private void FetchWeatherFromServer()
{
 // Simulate the delay from network access.
 Thread.Sleep(4000);

 // Tried and true method for weather forecasting - random numbers.
 Random rand = new Random();
 String weather;

 if (rand.Next(2) == 0)
 {
 weather = "rainy";
 }
 else
 {
 weather = "sunny";
 }

 // Schedule the update function in the UI thread.
 tomorrowsWeather.Dispatcher.BeginInvoke(
 System.Windows.Threading.DispatcherPriority.Normal,
 new OneArgDelegate(UpdateUserInterface),
 weather);
}

private void UpdateUserInterface(String weather)
{
 //Set the weather image
 if (weather == "sunny")
 {
 weatherIndicatorImage.Source = (ImageSource)this.Resources[
 "SunnyImageSource"];
 }
 else if (weather == "rainy")
 {
 weatherIndicatorImage.Source = (ImageSource)this.Resources[

```

```

 "RainingImageSource"];
 }

 //Stop clock animation
 showClockFaceStoryboard.Stop(this);
 hideClockFaceStoryboard.Begin(this);

 //Update UI text
 fetchButton.IsEnabled = true;
 fetchButton.Content = "Fetch Forecast";
 weatherText.Text = weather;
}

private void HideClockFaceStoryboard_Completed(object sender,
 EventArgs args)
{
 showWeatherImageStoryboard.Begin(this);
}

private void HideWeatherImageStoryboard_Completed(object sender,
 EventArgs args)
{
 showClockFaceStoryboard.Begin(this, true);
}
}
}

```

```

Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Media
Imports System.Windows.Media.Animation
Imports System.Windows.Media.Imaging
Imports System.Windows.Shapes
Imports System.Windows.Threading
Imports System.Threading

Namespace SDKSamples
 Partial Public Class Window1
 Inherits Window
 ' Delegates to be used in placing jobs onto the Dispatcher.
 Private Delegate Sub NoArgDelegate()
 Private Delegate Sub OneArgDelegate(ByVal arg As String)

 ' Storyboards for the animations.
 Private showClockFaceStoryboard As Storyboard
 Private hideClockFaceStoryboard As Storyboard
 Private showWeatherImageStoryboard As Storyboard
 Private hideWeatherImageStoryboard As Storyboard

 Public Sub New()
 MyBase.New()
 InitializeComponent()
 End Sub

 Private Sub Window_Loaded(ByVal sender As Object, ByVal e As RoutedEventArgs)
 ' Load the storyboard resources.
 showClockFaceStoryboard = CType(Me.Resources("ShowClockFaceStoryboard"), Storyboard)
 hideClockFaceStoryboard = CType(Me.Resources("HideClockFaceStoryboard"), Storyboard)
 showWeatherImageStoryboard = CType(Me.Resources("ShowWeatherImageStoryboard"), Storyboard)
 hideWeatherImageStoryboard = CType(Me.Resources("HideWeatherImageStoryboard"), Storyboard)
 End Sub

 Private Sub ForecastButtonHandler(ByVal sender As Object, ByVal e As RoutedEventArgs)
 ' Change the status image and start the rotation animation.
 fetchButton.IsEnabled = False
 fetchButton.Content = "Contacting Server"
 End Sub
 End Class

```

```

weatherText.Text = ""
hideWeatherStoryboard.Begin(Me)

' Start fetching the weather forecast asynchronously.
Dim fetcher As New NoArgDelegate(AddressOf Me.FetchWeatherFromServer)

fetcher.BeginInvoke(Nothing, Nothing)
End Sub

Private Sub FetchWeatherFromServer()
 ' Simulate the delay from network access.
 Thread.Sleep(4000)

 ' Tried and true method for weather forecasting - random numbers.
 Dim rand As New Random()
 Dim weather As String

 If rand.Next(2) = 0 Then
 weather = "rainy"
 Else
 weather = "sunny"
 End If

 ' Schedule the update function in the UI thread.
 tomorrowWeather.Dispatcher.BeginInvoke(System.Windows.Threading.DispatcherPriority.Normal, New
OneArgDelegate(AddressOf UpdateUserInterface), weather)
End Sub

Private Sub UpdateUserInterface(ByVal weather As String)
 'Set the weather image
 If weather = "sunny" Then
 weatherIndicatorImage.Source = CType(Me.Resources("SunnyImageSource"), ImageSource)
 ElseIf weather = "rainy" Then
 weatherIndicatorImage.Source = CType(Me.Resources("RainingImageSource"), ImageSource)
 End If

 'Stop clock animation
 showClockFaceStoryboard.Stop(Me)
 hideClockFaceStoryboard.Begin(Me)

 'Update UI text
 fetchButton.IsEnabled = True
 fetchButton.Content = "Fetch Forecast"
 weatherText.Text = weather
End Sub

Private Sub HideClockFaceStoryboard_Completed(ByVal sender As Object, ByVal args As EventArgs)
 showWeatherImageStoryboard.Begin(Me)
End Sub

Private Sub HideWeatherImageStoryboard_Completed(ByVal sender As Object, ByVal args As EventArgs)
 showClockFaceStoryboard.Begin(Me, True)
End Sub
End Class
End Namespace

```

Di seguito sono elencati alcuni dettagli da considerare.

- Creazione del gestore dei pulsanti

```

private void ForecastButtonHandler(object sender, RoutedEventArgs e)
{
 // Change the status image and start the rotation animation.
 fetchButton.IsEnabled = false;
 fetchButton.Content = "Contacting Server";
 weatherText.Text = "";
 hideWeatherImageStoryboard.Begin(this);

 // Start fetching the weather forecast asynchronously.
 NoArgDelegate fetcher = new NoArgDelegate(
 this.FetchWeatherFromServer);

 fetcher.BeginInvoke(null, null);
}

```

```

Private Sub ForecastButtonHandler(ByVal sender As Object, ByVal e As RoutedEventArgs)
 ' Change the status image and start the rotation animation.
 fetchButton.IsEnabled = False
 fetchButton.Content = "Contacting Server"
 weatherText.Text = ""
 hideWeatherImageStoryboard.Begin(Me)

 ' Start fetching the weather forecast asynchronously.
 Dim fetcher As New NoArgDelegate(AddressOf Me.FetchWeatherFromServer)

 fetcher.BeginInvoke(Nothing, Nothing)
End Sub

```

Quando si fa clic sul pulsante, viene visualizzato il disegno dell'orologio e viene avviata l'animazione. Il pulsante viene disabilitato. Il metodo `FetchWeatherFromServer` viene richiamato in un nuovo thread, quindi viene restituito, consentendo al `Dispatcher` di elaborare gli eventi mentre si è in attesa di raccogliere le previsioni meteorologiche.

- Recupero dei dati meteo

```

private void FetchWeatherFromServer()
{
 // Simulate the delay from network access.
 Thread.Sleep(4000);

 // Tried and true method for weather forecasting - random numbers.
 Random rand = new Random();
 String weather;

 if (rand.Next(2) == 0)
 {
 weather = "rainy";
 }
 else
 {
 weather = "sunny";
 }

 // Schedule the update function in the UI thread.
 tomorrowsWeather.Dispatcher.BeginInvoke(
 System.Windows.Threading.DispatcherPriority.Normal,
 new OneArgDelegate(UpdateUserInterface),
 weather);
}

```

```

Private Sub FetchWeatherFromServer()
 ' Simulate the delay from network access.
 Thread.Sleep(4000)

 ' Tried and true method for weather forecasting - random numbers.
 Dim rand As New Random()
 Dim weather As String

 If rand.Next(2) = 0 Then
 weather = "rainy"
 Else
 weather = "sunny"
 End If

 ' Schedule the update function in the UI thread.
 tomorrowsWeather.Dispatcher.BeginInvoke(System.Windows.Threading.DispatcherPriority.Normal, New
OneArgDelegate(AddressOf UpdateUserInterface), weather)
End Sub

```

Per maggiore semplicità, l'esempio non contiene codice di rete. Viene invece simulato il ritardo dell'accesso di rete sospendendo il thread per quattro secondi. In questo periodo, il thread di Interfaccia utente originale è ancora in esecuzione e risponde agli eventi. Per illustrare questa situazione, viene lasciata in esecuzione un'animazione e i pulsanti di riduzione a icona e ingrandimento continuano a funzionare.

Al termine del ritardo e abbiamo selezionato in modo casuale le previsioni meteorologiche, è il momento di riportare al thread Interfaccia utente. Questa operazione viene eseguita pianificando una chiamata a

`UpdateUserInterface` nel thread Interfaccia utente utilizzando `Dispatcher` di tale thread. Viene passata una stringa che descrive le condizioni meteo alla chiamata al metodo pianificata.

- Aggiornamento del Interfaccia utente

```

private void UpdateUserInterface(String weather)
{
 //Set the weather image
 if (weather == "sunny")
 {
 weatherIndicatorImage.Source = (ImageSource)this.Resources[
 "SunnyImageSource"];
 }
 else if (weather == "rainy")
 {
 weatherIndicatorImage.Source = (ImageSource)this.Resources[
 "RainingImageSource"];
 }

 //Stop clock animation
 showClockFaceStoryboard.Stop(this);
 hideClockFaceStoryboard.Begin(this);

 //Update UI text
 fetchButton.IsEnabled = true;
 fetchButton.Content = "Fetch Forecast";
 weatherText.Text = weather;
}

```

```

Private Sub UpdateUserInterface(ByVal weather As String)
 'Set the weather image
 If weather = "sunny" Then
 weatherIndicatorImage.Source = CType(Me.Resources("SunnyImageSource"), ImageSource)
 ElseIf weather = "rainy" Then
 weatherIndicatorImage.Source = CType(Me.Resources("RainingImageSource"), ImageSource)
 End If

 'Stop clock animation
 showClockFaceStoryboard.Stop(Me)
 hideClockFaceStoryboard.Begin(Me)

 'Update UI text
 fetchButton.IsEnabled = True
 fetchButton.Content = "Fetch Forecast"
 weatherText.Text = weather
End Sub

```

Quando il [Dispatcher](#) nel thread Interfaccia utente dispone di tempo, viene eseguita la chiamata pianificata a `UpdateUserInterface`. Questo metodo interrompe l'animazione dell'orologio e sceglie un'immagine per descrivere le condizioni meteo. L'immagine viene visualizzata e il pulsante "Fetch Forecast" viene ripristinato.

### Più finestre, più thread

Alcune applicazioni WPF richiedono più finestre di primo livello. È perfettamente accettabile per una combinazione di thread/[Dispatcher](#) gestire più finestre, ma a volte diversi thread offrono un processo migliore. Ciò è particolarmente vero nel caso in cui ci sia la possibilità che una delle finestre monopolizzi il thread.

Esplora risorse funziona in questo modo. Ogni nuova finestra di Esplora risorse appartiene al processo originale, ma viene creata sotto il controllo di un thread indipendente.

Utilizzando un controllo WPF[Frame](#), è possibile visualizzare le pagine Web. È possibile creare facilmente un sostituto semplice di Internet Explorer. Si inizia con una funzionalità importante: la possibilità di aprire una nuova finestra. Quando l'utente fa clic sul pulsante "Nuova finestra", viene aperta una copia della finestra in un thread separato. In questo modo, le operazioni a esecuzione prolungata o di blocco in una delle finestre non bloccheranno tutte le altre finestre.

In realtà, il modello di browser Web ha un proprio modello di threading complesso. È stato scelto questo esempio perché dovrebbe risultare familiare alla maggior parte dei lettori.

L'esempio seguente mostra il codice.

```
<Window x:Class="SDKSamples.Window1"
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="MultiBrowse"
 Height="600"
 Width="800"
 Loaded="OnLoaded"
 >
<StackPanel Name="Stack" Orientation="Vertical">
 <StackPanel Orientation="Horizontal">
 <Button Content="New Window"
 Click="NewWindowHandler" />
 <TextBox Name="newLocation"
 Width="500" />
 <Button Content="GO!"
 Click="Browse" />
 </StackPanel>
 <Frame Name="placeHolder"
 Width="800"
 Height="550"/>
</StackPanel>
</Window>
```

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Threading;
using System.Threading;

namespace SDKSamples
{
 public partial class Window1 : Window
 {

 public Window1() : base()
 {
 InitializeComponent();
 }

 private void OnLoaded(object sender, RoutedEventArgs e)
 {
 placeHolder.Source = new Uri("http://www.msn.com");
 }

 private void Browse(object sender, RoutedEventArgs e)
 {
 placeHolder.Source = new Uri(newLocation.Text);
 }

 private void NewWindowHandler(object sender, RoutedEventArgs e)
 {
 Thread newWindowThread = new Thread(new ThreadStart(ThreadStartingPoint));
 newWindowThread.SetApartmentState(ApartmentState.STA);
 newWindowThread.IsBackground = true;
 newWindowThread.Start();
 }

 private void ThreadStartingPoint()
 {
 Window1 tempWindow = new Window1();
 tempWindow.Show();
 System.Windows.Threading.Dispatcher.Run();
 }
 }
}
```

```

Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Data
Imports System.Windows.Threading
Imports System.Threading

Namespace SDKSamples
 Partial Public Class Window1
 Inherits Window

 Public Sub New()
 MyBase.New()
 InitializeComponent()
 End Sub

 Private Sub OnLoaded(ByVal sender As Object, ByVal e As RoutedEventArgs)
 placeHolder.Source = New Uri("http://www.msn.com")
 End Sub

 Private Sub Browse(ByVal sender As Object, ByVal e As RoutedEventArgs)
 placeHolder.Source = New Uri(newLocation.Text)
 End Sub

 Private Sub NewWindowHandler(ByVal sender As Object, ByVal e As RoutedEventArgs)
 Dim newWindowThread As New Thread(New ThreadStart(AddressOf ThreadStartingPoint))
 newWindowThread.SetApartmentState(ApartmentState.STA)
 newWindowThread.IsBackground = True
 newWindowThread.Start()
 End Sub

 Private Sub ThreadStartingPoint()
 Dim tempWindow As New Window1()
 tempWindow.Show()
 System.Windows.Threading.Dispatcher.Run()
 End Sub
 End Class
End Namespace

```

I segmenti di threading seguenti del codice sono i più interessanti in questo contesto:

```

private void NewWindowHandler(object sender, RoutedEventArgs e)
{
 Thread newWindowThread = new Thread(new ThreadStart(ThreadStartingPoint));
 newWindowThread.SetApartmentState(ApartmentState.STA);
 newWindowThread.IsBackground = true;
 newWindowThread.Start();
}

```

```

Private Sub NewWindowHandler(ByVal sender As Object, ByVal e As RoutedEventArgs)
 Dim newWindowThread As New Thread(New ThreadStart(AddressOf ThreadStartingPoint))
 newWindowThread.SetApartmentState(ApartmentState.STA)
 newWindowThread.IsBackground = True
 newWindowThread.Start()
End Sub

```

Questo metodo viene chiamato quando si fa clic sul pulsante "Nuova finestra". Il metodo crea un nuovo thread e lo avvia in modo asincrono.

```
private void ThreadStartingPoint()
{
 Window1 tempWindow = new Window1();
 tempWindow.Show();
 System.Windows.Threading.Dispatcher.Run();
}
```

```
Private Sub ThreadStartingPoint()
 Dim tempWindow As New Window1()
 tempWindow.Show()
 System.Windows.Threading.Dispatcher.Run()
End Sub
```

Questo metodo è il punto di partenza per il nuovo thread. Viene creata una nuova finestra sotto il controllo di questo thread. WPF crea automaticamente una nuova [Dispatcher](#) per gestire il nuovo thread. Per rendere funzionale la finestra, è sufficiente avviare il [Dispatcher](#).

## Dettagli tecnici e difficoltà

### Scrittura di componenti usando il threading

La guida per gli sviluppatori di Microsoft .NET Framework descrive un modello per il modo in cui un componente può esporre il comportamento asincrono ai propri client. vedere [Cenni preliminari sul modello asincrono basato su eventi](#). Si supponga, ad esempio, di voler creare il pacchetto del [FetchWeatherFromServer](#) metodo in un componente riutilizzabile e non grafico. Seguendo il modello di Microsoft .NET Framework standard, il risultato sarà simile al seguente.

```
public class WeatherComponent : Component
{
 //gets weather: Synchronous
 public string GetWeather()
 {
 string weather = "";

 //predict the weather

 return weather;
 }

 //get weather: Asynchronous
 public void GetWeatherAsync()
 {
 //get the weather
 }

 public event GetWeatherCompletedEventHandler GetWeatherCompleted;
}

public class GetWeatherCompletedEventArgs : AsyncCompletedEventArgs
{
 public GetWeatherCompletedEventArgs(Exception error, bool canceled,
 object userState, string weather)
 :
 base(error, canceled, userState)
 {
 _weather = weather;
 }

 public string Weather
 {
 get { return _weather; }
 }
 private string _weather;
}

public delegate void GetWeatherCompletedEventHandler(object sender,
 GetWeatherCompletedEventArgs e);
```

```

Public Class WeatherComponent
 Inherits Component
 'gets weather: Synchronous
 Public Function GetWeather() As String
 Dim weather As String = ""

 'predict the weather

 Return weather
 End Function

 'get weather: Asynchronous
 Public Sub GetWeatherAsync()
 'get the weather
 End Sub

 Public Event GetWeatherCompleted As GetWeatherCompletedEventHandler
End Class

Public Class GetWeatherCompletedEventArgs
 Inherits AsyncCompletedEventArgs
 Public Sub New(ByVal [error] As Exception, ByVal canceled As Boolean, ByVal userState As Object, ByVal
weather As String)
 MyBase.New([error], canceled, userState)
 _weather = weather
 End Sub

 Public ReadOnly Property Weather() As String
 Get
 Return _weather
 End Get
 End Property
 Private _weather As String
End Class

Public Delegate Sub GetWeatherCompletedEventHandler(ByVal sender As Object, ByVal e As
GetWeatherCompletedEventArgs)

```

`GetWeatherAsync` userebbe una delle tecniche descritte in precedenza, ad esempio la creazione di un thread in background, per eseguire le attività in modo asincrono, senza bloccare il thread chiamante.

Una delle parti più importanti di questo modello è la chiamata al metodo `methodname Completed` sullo stesso thread che ha chiamato il metodo `MethodName Async` per iniziare. Questa operazione può essere eseguita in modo abbastanza semplice, archiviando `CurrentDispatcher`, ma in WPF questo caso il componente non grafico può essere usato solo in applicazioni WPF, non nei programmi Windows Forms o ASP.NET.

La classe `DispatcherSynchronizationContext` risponde a questa esigenza, ovvero una versione semplificata di `Dispatcher` che funziona anche con altri Interfaccia utente Framework.

```

public class WeatherComponent2 : Component
{
 public string GetWeather()
 {
 return fetchWeatherFromServer();
 }

 private DispatcherSynchronizationContext requestingContext = null;

 public void GetWeatherAsync()
 {
 if (requestingContext != null)
 throw new InvalidOperationException("This component can only handle 1 async request at a time");

 requestingContext = (DispatcherSynchronizationContext)DispatcherSynchronizationContext.Current;

 NoArgDelegate fetcher = new NoArgDelegate(this.fetchWeatherFromServer);

 // Launch thread
 fetcher.BeginInvoke(null, null);
 }

 private void RaiseEvent(GetWeatherCompletedEventArgs e)
 {
 if (GetWeatherCompleted != null)
 GetWeatherCompleted(this, e);
 }

 private string fetchWeatherFromServer()
 {
 // do stuff
 string weather = "";

 GetWeatherCompletedEventArgs e =
 new GetWeatherCompletedEventArgs(null, false, null, weather);

 SendOrPostCallback callback = new SendOrPostCallback(DoEvent);
 requestingContext.Post(callback, e);
 requestingContext = null;

 return e.Weather;
 }

 private void DoEvent(object e)
 {
 //do stuff
 }

 public event GetWeatherCompletedEventHandler GetWeatherCompleted;
 public delegate string NoArgDelegate();
}

```

```

Public Class WeatherComponent2
 Inherits Component
 Public Function GetWeather() As String
 Return fetchWeatherFromServer()
 End Function

 Private requestingContext As DispatcherSynchronizationContext = Nothing

 Public Sub GetWeatherAsync()
 If requestingContext IsNot Nothing Then
 Throw New InvalidOperationException("This component can only handle 1 async request at a time")
 End If

 requestingContext = CType(DispatcherSynchronizationContext.Current,
DispatcherSynchronizationContext)

 Dim fetcher As New NoArgDelegate(AddressOf Me.fetchWeatherFromServer)

 ' Launch thread
 fetcher.BeginInvoke(Nothing, Nothing)
 End Sub

 Private Sub [RaiseEvent](ByVal e As GetWeatherCompletedEventArgs)
 RaiseEvent GetWeatherCompleted(Me, e)
 End Sub

 Private Function fetchWeatherFromServer() As String
 ' do stuff
 Dim weather As String = ""

 Dim e As New GetWeatherCompletedEventArgs(Nothing, False, Nothing, weather)

 Dim callback As New SendOrPostCallback(AddressOf DoEvent)
 requestingContext.Post(callback, e)
 requestingContext = Nothing

 Return e.Weather
 End Function

 Private Sub DoEvent(ByVal e As Object)
 'do stuff
 End Sub

 Public Event GetWeatherCompleted As GetWeatherCompletedEventHandler
 Public Delegate Function NoArgDelegate() As String
End Class

```

## Distribuzione annidata

In alcuni casi non è possibile bloccare completamente il thread di Interfaccia utente. Si prenda in considerazione il metodo [Show](#) della classe [MessageBox](#). [Show](#) non viene restituito fino a quando l'utente non fa clic sul pulsante OK. Crea però una finestra che deve avere un ciclo di messaggi per essere interattiva. Mentre è in attesa del clic dell'utente su OK, la finestra dell'applicazione originale non risponde all'input utente. Continua però a elaborare i messaggi di disegno dell'interfaccia. La finestra originale ridisegna se stessa quando viene coperta e mostrata.



Un thread deve essere responsabile della finestra di messaggio. WPF potrebbe creare un nuovo thread solo per la finestra di messaggio, ma questo thread non sarebbe in grado di disegnare gli elementi disabilitati nella finestra originale (in base a quanto illustrato in precedenza in relazione all'esclusione reciproca). WPF utilizza invece un sistema di elaborazione dei messaggi annidato. La classe [Dispatcher](#) include un metodo speciale denominato [PushFrame](#), che archivia il punto di esecuzione corrente di un'applicazione, quindi avvia un nuovo ciclo di messaggi. Al termine del ciclo di messaggi annidati, l'esecuzione riprende dopo la chiamata di [PushFrame](#) originale.

In questo caso, [PushFrame](#) gestisce il contesto del programma alla chiamata a [MessageBox.Show](#) avvia un nuovo ciclo di messaggi per ridisegnare la finestra di sfondo e gestire l'input della finestra di messaggio. Quando l'utente fa clic su OK e cancella la finestra popup, il ciclo annidato viene chiuso e il controllo riprende dopo la chiamata a [Show](#).

### Eventi indirizzati non aggiornati

Il sistema di eventi indirizzati in WPF notifica a interi alberi quando vengono generati eventi.

```
<Canvas MouseLeftButtonDown="handler1"
 Width="100"
 Height="100"
 >
 <Ellipse Width="50"
 Height="50"
 Fill="Blue"
 Canvas.Left="30"
 Canvas.Top="50"
 MouseLeftButtonDown="handler2"
 />
</Canvas>
```

Quando viene premuto il pulsante sinistro del mouse sull'ellisse, viene eseguito `handler2`. Al termine dell'`handler2`, l'evento viene passato all'oggetto [Canvas](#), che usa `handler1` per elaborarlo. Questa situazione si verifica solo se `handler2` non contrassegna in modo esplicito l'oggetto evento come gestito.

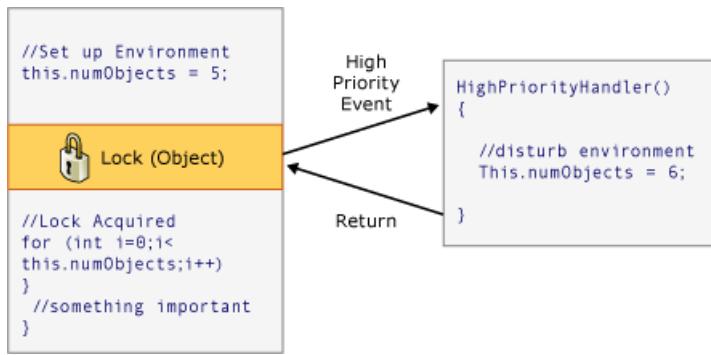
È possibile che `handler2` l'elaborazione di questo evento venga eseguita da molto tempo. `handler2` possibile utilizzare [PushFrame](#) per avviare un ciclo di messaggi annidato che non restituisce per ore. Se `handler2` non contrassegna l'evento come gestito quando il ciclo di messaggi è completo, l'evento viene passato alla struttura ad albero anche se è molto vecchio.

### Reentrancy e blocco

Il meccanismo di blocco del Common Language Runtime (CLR) non si comporta esattamente come si può immaginare; si potrebbe aspettare che un thread smetta completamente l'operazione quando viene richiesto un blocco. In realtà, il thread continua a ricevere ed elaborare i messaggi con priorità alta. Questo consente di evitare i deadlock e rendere le interfacce un minimo reattive, ma introduce la possibilità di bug. Nella maggior parte dei casi non è necessario conoscere questo aspetto, ma in rari casi (in genere che coinvolgono i messaggi della finestra Win32 o i componenti COM STA) può essere utile sapere.

La maggior parte delle interfacce non è compilata tenendo conto di thread safety perché gli sviluppatori

utilizzano il presupposto che non venga mai eseguito l'accesso a una Interfaccia utente da più di un thread. In questo caso, il singolo thread può apportare modifiche ambientali in momenti imprevisti, causando gli effetti negativi che la [DispatcherObject](#) meccanismo di esclusione reciproca dovrebbe risolvere. Si consideri lo pseudocodice seguente:



Nella maggior parte dei casi questo è il momento giusto, ma in WPF casi in cui tali rientranze impreviste possono effettivamente causare problemi. Quindi, in determinati momenti chiave, WPF chiama [DisableProcessing](#), che modifica l'istruzione di blocco per il thread in modo da utilizzare il blocco WPF rientrante, anziché il consueto blocco CLR.

Perché il team CLR ha scelto questo comportamento? La scelta ha a che fare con gli oggetti COM STA e il thread di finalizzazione. Quando un oggetto viene sottoposto a Garbage Collection, il relativo [Finalize](#) metodo viene eseguito sul thread del finalizzatore dedicato, non sul thread Interfaccia utente. Il problema risiede nel fatto che un oggetto COM STA creato nel thread di Interfaccia utente può essere eliminato solo sul thread di Interfaccia utente. CLR esegue l'equivalente di un [BeginInvoke](#) (in questo caso utilizzando Win32 [SendMessage](#)). Tuttavia, se il thread di Interfaccia utente è occupato, il thread del finalizzatore viene bloccato e l'oggetto COM STA non può essere eliminato, il che crea una perdita di memoria grave. Quindi, il team CLR ha fatto la chiamata complessa per far funzionare i blocchi nel modo in cui fanno.

L'attività per WPF consiste nel evitare una rientranza imprevista senza reintrodurre la perdita di memoria, motivo per cui non è possibile bloccare la rientranza in tutto il mondo.

## Vedere anche

- [Esempio di applicazione a thread singolo con calcolo a esecuzione prolungata](#)

# Riferimenti alle API non gestite WPF

03/02/2020 • 2 minutes to read • [Edit Online](#)

Le librerie Windows Presentation Foundation (WPF) espongono una serie di funzioni non gestite destinate solo all'uso interno. Non devono essere chiamati dal codice utente.

## Contenuto della sezione

[Funzione Activate](#)  
[Funzione CreateDispatchSTAForwarder](#)  
[Funzione Deactivate](#)  
[Funzione ForwardTranslateAccelerator](#)  
[Funzione LoadFromHistory](#)  
[Funzione ProcessUnhandledException](#)  
[Funzione SaveToHistory](#)  
[Funzione SetFakeActiveWindow](#)

## Vedere anche

- [Avanzate](#)

# Funzione Activate (riferimenti alle API non gestite WPF)

03/02/2020 • 2 minutes to read • [Edit Online](#)

Questa API supporta l'infrastruttura Windows Presentation Foundation (WPF) e non può essere usata direttamente dal codice.

Utilizzato dall'infrastruttura Windows Presentation Foundation (WPF) per la gestione di Windows.

## Sintassi

```
void Activate(
 const ActivateParameters* pParameters,
 __deref_out_ecount(1) LPUNKNOWN* ppInner,
);
```

## Parametri

`pParameters`

Puntatore ai parametri di attivazione della finestra.

`ppInner`

Puntatore all'indirizzo di un buffer a elemento singolo che contiene un puntatore a un oggetto [IOleDocument](#).

## Requisiti

**Piattaforme:** Vedere [.NET Framework requisiti di sistema](#).

### DLL

Nel .NET Framework 3,0 e 3,5: PresentationHostDLL.dll

In .NET Framework 4 e versioni successive: PresentationHost\_v0400.dll

**Versione .NET Framework:** Disponibile dalla 3,0

## Vedere anche

- [Riferimenti alle API non gestite di WPF](#)

# Funzione CreateDispatchSTAForwarder (riferimenti alle API non gestite WPF)

03/02/2020 • 2 minutes to read • [Edit Online](#)

Questa API supporta l'infrastruttura Windows Presentation Foundation (WPF) e non può essere usata direttamente dal codice.

Utilizzato dall'infrastruttura Windows Presentation Foundation (WPF) per la gestione di thread e finestre.

## Sintassi

```
HRESULT CreateIDispatchSTAForwarder(
 __in IDispatch *pDispatchDelegate,
 __deref_out IDispatch **ppForwarder
)
```

## Parametri

### Valore proprietà/Valore restituito

pDispatchDelegate

Puntatore a un'interfaccia [IDispatch](#).

ppForwarder

Puntatore all'indirizzo di un'interfaccia [IDispatch](#).

## Requisiti

**Piattaforme:** Vedere [.NET Framework requisiti di sistema](#).

### DLL

Nel .NET Framework 3,0 e 3,5: PresentationHostDLL.dll

In .NET Framework 4 e versioni successive: PresentationHost\_v0400.dll

**Versione .NET Framework:** Disponibile dalla 3,0

## Vedere anche

- [Riferimenti alle API non gestite di WPF](#)

# Funzione Deactivate (riferimenti alle API non gestite WPF)

03/02/2020 • 2 minutes to read • [Edit Online](#)

Questa API supporta l'infrastruttura Windows Presentation Foundation (WPF) e non può essere usata direttamente dal codice.

Utilizzato dall'infrastruttura Windows Presentation Foundation (WPF) per la gestione di Windows.

## Sintassi

```
void Deactivate()
```

## Requisiti

**Piattaforme:** Vedere [.NET Framework requisiti di sistema](#).

### DLL

Nel .NET Framework 3,0 e 3,5: PresentationHostDLL.dll

In .NET Framework 4 e versioni successive: PresentationHost\_v0400.dll

**Versione .NET Framework:** Disponibile dalla 3,0

## Vedere anche

- [Riferimenti alle API non gestite di WPF](#)

# Funzione ForwardTranslateAccelerator (riferimenti alle API non gestite WPF)

03/02/2020 • 2 minutes to read • [Edit Online](#)

Questa API supporta l'infrastruttura Windows Presentation Foundation (WPF) e non può essere usata direttamente dal codice.

Utilizzato dall'infrastruttura Windows Presentation Foundation (WPF) per la gestione di Windows.

## Sintassi

```
HRESULT ForwardTranslateAccelerator(
 MSG* pMsg,
 VARIANT_BOOL appUnhandled
)
```

## Parametri

pMsg

Puntatore a un messaggio.

appUnhandled

`true` quando l'app ha già avuto la possibilità di gestire il messaggio di input, ma non lo ha gestito; in caso contrario, `false`.

## Requisiti

**Piattaforme:** Vedere [.NET Framework requisiti di sistema](#).

### DLL

Nel .NET Framework 3,0 e 3,5: PresentationHostDLL.dll

In .NET Framework 4 e versioni successive: PresentationHost\_v0400.dll

**Versione .NET Framework:** Disponibile dalla 3,0

## Vedere anche

- [Riferimenti alle API non gestite di WPF](#)

# Funzione LoadFromHistory (riferimenti alle API non gestite WPF)

03/02/2020 • 2 minutes to read • [Edit Online](#)

Questa API supporta l'infrastruttura Windows Presentation Foundation (WPF) e non può essere usata direttamente dal codice.

Utilizzato dall'infrastruttura Windows Presentation Foundation (WPF) per la gestione di Windows.

## Sintassi

```
HRESULT LoadFromHistory_export(
 IStream* pHistoryStream,
 IBindCtx* pBindCtx
)
```

## Parametri

pHistoryStream

Puntatore a un flusso di informazioni sulla cronologia.

pBindCtx

Puntatore a un contesto di associazione.

## Requisiti

**Piattaforme:** Vedere [.NET Framework requisiti di sistema](#).

### DLL

Nel .NET Framework 3,0 e 3,5: PresentationHostDLL.dll

In .NET Framework 4 e versioni successive: PresentationHost\_v0400.dll

**Versione .NET Framework:** Disponibile dalla 3,0

## Vedere anche

- [Riferimenti alle API non gestite di WPF](#)

# Funzione ProcessUnhandledException (riferimenti alle API non gestite WPF)

03/02/2020 • 2 minutes to read • [Edit Online](#)

Questa API supporta l'infrastruttura Windows Presentation Foundation (WPF) e non può essere usata direttamente dal codice.

Utilizzato dall'infrastruttura Windows Presentation Foundation (WPF) per la gestione delle eccezioni.

## Sintassi

```
void __stdcall ProcessUnhandledException(
 __in_ecount(1) BSTR errorMsg
)
```

## Parametri

errorMsg

Messaggio di errore.

## Requisiti

**Piattaforme:** Vedere [.NET Framework requisiti di sistema](#).

### DLL

Nel .NET Framework 3,0 e 3,5: PresentationHostDLL.dll

In .NET Framework 4 e versioni successive: PresentationHost\_v0400.dll

**Versione .NET Framework:** Disponibile dalla 3,0

## Vedere anche

- [Riferimenti alle API non gestite di WPF](#)

# Funzione SaveToHistory (riferimenti alle API non gestite WPF)

03/02/2020 • 2 minutes to read • [Edit Online](#)

Questa API supporta l'infrastruttura Windows Presentation Foundation (WPF) e non può essere usata direttamente dal codice.

Utilizzato dall'infrastruttura Windows Presentation Foundation (WPF) per la gestione di Windows.

## Sintassi

```
HRESULT SaveToHistory(
 __in_ecount(1) IStream* pHistoryStream
)
```

## Parametri

pHistoryStream

Puntatore al flusso della cronologia.

## Requisiti

### Requisiti

**Piattaforme:** Vedere [.NET Framework requisiti di sistema](#).

#### DLL

Nel .NET Framework 3,0 e 3,5: PresentationHostDLL.dll

In .NET Framework 4 e versioni successive: PresentationHost\_v0400.dll

**Versione .NET Framework:** Disponibile dalla 3,0

## Vedere anche

- [Riferimenti alle API non gestite di WPF](#)

# Funzione SetFakeActiveWindow (riferimenti alle API non gestite WPF)

03/02/2020 • 2 minutes to read • [Edit Online](#)

Questa API supporta l'infrastruttura Windows Presentation Foundation (WPF) e non può essere usata direttamente dal codice.

Utilizzato dall'infrastruttura Windows Presentation Foundation (WPF) per la gestione di Windows.

## Sintassi

```
void __stdcall SetFakeActiveWindow(
 HWND hwnd
)
```

## Parametri

**HWND**

Handle di finestra.

## Requisiti

**Piattaforme:** Vedere [.NET Framework requisiti di sistema](#).

**DLL:** PresentationHost\_v0400.dll

**Versione .NET Framework:** Disponibile dalla 4

## Vedere anche

- [Riferimenti alle API non gestite di WPF](#)