

# RichTextBox for WPF

2021.12.21 更新

グレープシティ株式会社

## 目次

<a href="#">製品の概要</a>	4
<a href="#">主な特長</a>	5-6
<a href="#">クイックスタート</a>	7
<a href="#">手順1: WPF アプリケーションの作成</a>	7-8
<a href="#">手順2: スペルチェック機能の追加</a>	8-10
<a href="#">手順3: 実行時のコントロールの使用</a>	10-13
<a href="#">XAML クイックリファレンス</a>	14-15
<a href="#">RichTextBox for WPF の使い方</a>	16
<a href="#">C1RichTextBox の概念と主要なプロパティ</a>	16-18
<a href="#">C1RichTextBox のコンテンツ</a>	18-19
<a href="#">ロードと保存</a>	19-21
<a href="#">カスタムコマンドバー</a>	21-22
<a href="#">クリップボード機能</a>	22
<a href="#">テキスト揃え機能</a>	22-23
<a href="#">フォント機能</a>	23
<a href="#">書式設定機能</a>	23-24
<a href="#">テキスト選択機能</a>	24-25
<a href="#">ドキュメント履歴機能</a>	25
<a href="#">ハイパーリンク</a>	25-28
<a href="#">レイアウト情報へのアクセス</a>	28-30
<a href="#">ペインタ</a>	30-32
<a href="#">スペルチェック</a>	32-34
<a href="#">構文の色指定</a>	34-38
<a href="#">スタイルのオーバーライド</a>	38-42
<a href="#">ヒットテスト</a>	42-45


<a href="#">HtmlFilter のカスタマイズ</a>	45-47
<a href="#">C1Document オブジェクトの使い方</a>	48
<a href="#">ドキュメントとレポートの作成</a>	48-57
<a href="#">分割表示の実装</a>	57-59
<a href="#">C1Document クラスの使用</a>	59-60
<a href="#">C1TextPointer の理解</a>	60-64
<a href="#">C1RichTextBoxToolbar の使い方</a>	65-66
<a href="#">[編集] グループ</a>	66
<a href="#">[段落] グループ</a>	66-68
<a href="#">[挿入] グループ</a>	68-69
<a href="#">[ツール] グループ</a>	69-70
<a href="#">C1SimplifiedRichTextBoxToolbarの使い方</a>	71-72
<a href="#">RichTextBox Ribbon の使い方</a>	73-76
<a href="#">RichTextBox ToolStrip の使い方</a>	77
<a href="#">RichTextBox でサポートされる要素</a>	78
<a href="#">HTML 要素</a>	78-80
<a href="#">HTML の属性</a>	80-86
<a href="#">CSS2 プロパティ</a>	86-90
<a href="#">CSS2 セレクタ</a>	90-92
<a href="#">RichTextBox for WPF の外観</a>	93
<a href="#">ClearStyle 技術</a>	93
<a href="#">ClearStyle の仕組み</a>	93
<a href="#">C1RichTextBox の ClearStyle プロパティ</a>	93-94
<a href="#">C1RichTextBox のテーマ</a>	94-100
<a href="#">スペルチェック</a>	101-103
<a href="#">タスク別ヘルプ</a>	104
<a href="#">テキストコンテンツの設定</a>	104-105

<a href="#"><u>HTML コンテンツの設定</u></a>	105-106
<a href="#"><u>C1RichTextBoxToolbar を C1RichTextBox に接続する</u></a>	106-107
<a href="#"><u>簡単な書式設定ツールバーの実装</u></a>	107-110
<a href="#"><u>スペルチェックの追加</u></a>	110-112

## 製品の概要

**RichTextBox for WPF** は、WPF 用の市場で最も充実したリッチテキストエディタです。書式設定されたテキストを HTML または RTF ドキュメントとしてロード、編集、および保存します。**C1RichTextBox** コントロールは、さまざまな書式設定、行の自動折り返し、HTML と RTF のインポート/エクスポート、テーブルのサポート、画像、注釈などを提供します。

リファレンス	
<b>C1.WPF.RichTextBox.4.5.2 アセンブリ</b>	<b>C1.WPF.RichTextBox アセンブリ</b>

 **RichTextBox** コントロールは、**.NET4.5.2** および **.NET5Framework** の両方と互換性があります。

## 主な特長

RichTextBox for WPF を使用すると、機能豊富でカスタマイズされたアプリケーションを作成できます。次のような主要機能を利用して、RichTextBox for WPF を最大限に活用してください。

- **インポートとエクスポートの形式**

RichTextBox for WPF は、RTF、HTML、およびプレーンテキストのインポートとエクスポートをサポートします。既存のリッチテキストまたは HTML を **C1RichTextBox** コントロールにロードし、ドキュメントを編集し、RTF または HTML に再度エクスポートします。

- **さまざまな書式の適用**

複数のフォント、装飾、色、テーブル、画像、リストなどを含むテキストを編集して書式設定できます。

- **言語のサポート**

**C1RichTextBox** は、中国語、日本語、韓国語の各言語をサポートするようになりました。

- **C1RichTextBoxToolbar**

一般的な機能をすべて備えた **C1RichTextBoxToolbar** コントロールを使用して即座に開始することも、自分でカスタムのツールバーを作成することもできます。このコントロールは、フォントファミリー、フォントサイズ、フォント拡大、フォント縮小、斜体、下線、大文字小文字の変更、下付き、上付き、テキストの色、テキスト強調表示の色、左寄せ、中央寄せ、右寄せ、均等割り付け、箇条書き、番号付き、テキスト折り返し、罫線の太さ、罫線の色、段落の色、マージン、パディング、画像の挿入、記号の挿入、ハイパーリンクの挿入、ハイパーリンクの削除、切り取り、コピー、貼り付け、元に戻す、やり直す、検索と置換、スペルチェックなどのアクションを提供します。**RichTextBoxToolbar** は、**C1Toolbar** コントロールを使用して完全なカスタマイズが可能です。詳細については、「[C1RichTextBoxToolbar の使い方](#)」を参照してください。

- **ページのズーム**

RichTextBox は、印刷レイアウトでも下書きビューでもページズームをサポートします。

- **テキストのスペルチェック**

RichTextBox は、**C1SpellChecker** コンポーネントを使用して次の2種類のスペルチェックをサポートします。

- **モーダルスペルチェック**：[スペル] ダイアログボックスを表示して、ドキュメント内のスペルミスを選択します。スペルミスを見逃したり、正しいスペルを入力するか候補リストから選択してミスを修正したり、その単語を辞書に追加することができます。詳細については、「[モーダルスペルチェック](#)」を参照してください。
- **入力中スペルチェック**：入力中にスペルミスは通常は赤い波線の下線で強調表示します。ドキュメント内のスペルミスをクリックすると、見逃す、辞書に追加する、スペル候補から選択してスペルミスを自動的に修正するなどのオプションメニューが表示されます。詳細については、「[スペルチェック](#)」を参照してください。

- **元に戻す/やり直しのサポート**

RichTextBox 内のデータを安心して編集できます。ボタンをクリックして、簡単に変更を元に戻したり、やり直すことができます。

- **クリップボードのサポート**

**C1RichTextBox** は、クリップボードを完全にサポートします。RichTextBox 内での切り取り/コピー/貼り付けの操作を実装します。

- **注釈**

C1RichTextBox を使用して、ドキュメントに強調表示や注釈を追加できます。注釈は、Web ドキュメントの一部にアタッチできるコメント、メモ、備考、または説明です。

- **PDF形式に保存**

直接印刷する他、**C1RichTextBox** の内容が PDF 形式にエクスポートすることもできます。

- **充実したドキュメントオブジェクトモデル**

WPF の **Document** クラスを下敷きにした RichTextBox の充実したドキュメントオブジェクトモデル (DOM) は、画像、リスト、ハイパーリンク、境界、テキスト範囲の背景色/前景色などをサポートします。この豊富な機能を持つ DOM を使用して、プログラムでドキュメントを作成および変更できます。詳細については、「[C1Document オブジェクトの使い方](#)」を参照してください。

- **超高速**

RichTextBox では、極めて高速にドキュメントをロードし、直ちに編集を開始できます。

- **Silverlight Toolkit テーマのサポート**

ExpressionDark、ExpressionLight、WhistlerBlue、RainierOrange、ShinyBlue、BureauBlack など、最もよく使用されている Microsoft Silverlight Toolkit テーマが組み込みでサポートされており、それを使用して UI にスタイルを追加できます。

## クイックスタート

このクイックスタートでは、最初に、Visual Studioで WPF アプリケーションを作成し、新しいアプリケーションに**C1RichTextBox** コントロールと**C1RichTextBoxToolBar** コントロールを追加し、アプリケーションをカスタマイズするコードを追加し、アプリケーションを実行して実行時に可能な操作を確認します。

## 手順1: WPF アプリケーションの作成

以下のセクションでは、.NET4.5.2および.NET5バージョンのRichTextBoxコントロールの使用を開始します。

### .NET 4.5.2

この手順では、新しい WPF アプリケーションを作成し、XAMLで **C1RichTextBox** コントロールと**C1RichTextBoxToolBar** コントロールを追加します。この手順を完了すると、主要な機能を備えたリッチテキストエディタができあがります。

### デザインビューで

デザインビューでは、WPFアプリケーションにRichTextBoxコントロールを追加するには、次の手順に従います。

1. Visual Studioで新しいWPFアプリケーションプロジェクトを作成します。
2. ツールボックスに移動し、**C1RichTextBox** コントロールと**C1RichTextBoxToolBar** コントロールを探します。
3. **C1RichTextBox** と**C1RichTextBoxToolBar** アイコンをダブルクリックして、**MainWindow** に追加します。
4. XAML ビューで **MainWindow** タグを編集し、ページにコントロールの名前空間を追加します。

XAML	copyCode
<pre>xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml" x:Class="RTBQuickStart.MainWindow" Title="MainWindow" Height="350" Width="525"&gt;</pre>	

5. カーソルを <Grid> タグと </Grid> タグの間に置き、1回クリックします。<Grid>タグの中に次のマークアップを追加して、**StackPanel** パネルを追加します。

XAML	copyCode
<pre>&lt;StackPanel HorizontalAlignment="Left" Margin="0,10,0,0" x:Name="SP" VerticalAlignment="Top" Height="418" Width="645" Grid.ColumnSpan="2" Grid.Column="1"/&gt;</pre>	

6. XAML ビューで **StackPanel** のタグの間をクリックし、次のマークアップを追加して、**C1RichTextBoxToolBar** コントロールと **C1RichTextBox** コントロールを追加します。

XAML
<pre>&lt;c1:C1RichTextBox Name="c1RichTextBox1" Margin="0,127,0,10"/&gt; &lt;c1:C1RichTextBoxToolBar RichTextBox="{Binding ElementName= c1RichTextBox1}" Name="C1RTBTB" Margin="3,0,-3,197" /&gt;</pre>



アプリケーションを実行すると、ほぼ完全な機能を備えた **C1RichTextBox** アプリケーションが表示されます。C1RichTextBox コントロールにテキストを入力し、**C1RichTextBoxToolbar** のオプションを使用してテキストを編集できます。次の手順では、スペルチェックを設定し、アプリケーションをさらにカスタマイズします。

## .NET 5

この手順では、新しいWPFコアアプリケーションを作成し、XAMLで C1RichTextBox コントロールを追加します。この手順を完了すると、主要な機能を備えたリッチテキストエディタができあがります。

### デザインビューで

デザインビューでは、WPFコアアプリケーションにRichTextBoxコントロールを追加するには、次の手順に従います。

1. Visual Studioで新しいWPFコアアプリケーションプロジェクトを作成します。
2. ツールボックスに移動し、**C1RichTextBox** コントロールを探します。
3. C1RichTextBox アイコンをダブルクリックして、MainWindow に追加します。
4. XAML ビューで MainWindow タグを編集し、ページにコントロールの名前空間を追加します。

XAML	copyCode
<pre>xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml" x:Class="RTBQuickStart.MainWindow" Title="MainWindow" Height="350" Width="525"&gt;</pre>	

5. カーソルを <Grid> タグと </Grid> タグの間に置き、1回クリックします。<Grid>タグの中に次のマークアップを追加して、**StackPanel** パネルを追加します。

XAML	copyCode
<pre>&lt;StackPanel HorizontalAlignment="Left" Margin="0,10,0,0" x:Name="SP" VerticalAlignment="Top" Height="418" Width="645" Grid.ColumnSpan="2" Grid.Column="1"/&gt;</pre>	

6. XAML ビューで **StackPanel** のタグの間をクリックし、次のマークアップを追加して、**C1RichTextBox** コントロールを追加します。

XAML
<pre>&lt;c1:C1RichTextBox Name="c1RichTextBox1" Margin="0,127,0,10"/&gt;</pre>

アプリケーションを実行すると、ほぼ完全な機能を備えた **C1RichTextBox** アプリケーションが表示されます。C1RichTextBox コントロールにテキストを入力し、オプションでテキストを編集できます。.NET 5 RichTextBoxで使用可能な編集機能の詳細については、それぞれの機能のトピックを参照してください。

## 手順2: スペルチェック機能の追加

前の手順では、新しいWPFアプリケーションを作成し、アプリケーションに **C1RichTextBox** コントロール

# RichTextBox for WPF

と **C1RichTextBoxToolBar** コントロールを追加しました。実行時にツールバーの **[スペルチェック]** ボタンをクリックすると、現在はスペルチェックが設定されていないというメッセージが表示されます。この手順では、アプリケーションをさらにカスタマイズして、アプリケーションにスペルチェック機能を追加します。

アプリケーションにスペルチェック機能を追加するには、次の手順に従います。

1. ソリューションエクスプローラで、プロジェクトを右クリックし、**[追加]** → **[既存の項目]** を選択します。
2. **[既存項目の追加]** ダイアログボックスで、**RichTextBoxSamples** サンプルフォルダ内にある **C1Spell\_en-US.dct** ファイルを見つけます。デフォルトでは、**ComponentOne Samples\WPF\C1.WPF.RichTextBox\RichTextBoxSamples\RichTextBoxSamples.Web** または **ComponentOne Samples\Silverlight 4.0\C1.Silverlight.RichTextBox\RichTextBoxSamples\RichTextBoxSamples.Web** の **Documents** フォルダにインストールされます。  
これはアメリカ英語の辞書ファイルです。代わりに別のファイルを追加する場合は、適切なコードを使用して以下の手順を変更できます。
3. ソリューションエクスプローラで、**MainPage.xaml** ファイルを右クリックし、**[コードの表示]** を選択して、コードファイルを開きます。
4. コードエディタで、次のコードを追加して次の名前空間をインポートします。

## VisualBasic

```
Imports C1.WPF.RichTextBox
Imports C1.WPF.SpellChecker
```

## C#

```
using C1.WPF.RichTextBox;
using C1.WPF.SpellChecker;
```

5. RichTextBoxコントロールでスペルチェック機能を追加するには、次のコードを **MainPage** コンストラクタに追加します。

## VisualBasic

```
Public Sub New()
    InitializeComponent()
    Dim spell As New C1SpellChecker()
    spell.MainDictionary.LoadAsync("C1Spell_en-US.dct")
    Me.C1RTB.SpellChecker = spell
End Sub
```

## C#

```
public MainPage()
{
    InitializeComponent();
    var spell = new C1SpellChecker();
    spell.MainDictionary.LoadAsync("C1Spell_en-US.dct");
    this.C1RTB.SpellChecker = spell;
}
```

6. **MainPage** コンストラクタの先ほど追加したコードの下に、次のコードを追加します。

## VisualBasic

```
Me.c1RichTextBox1.Text = "Hello World! Weelcome to the most complete rich text editor availible for WPF. Load, edit, and save formattted text as HTML or RTF documents with RichTextBox for WPF. The C1RichTextBox control provids rich formatting, automatic line wrapping, HTML and RTF import/export, table support, images, anotations, and more."
```

## C#

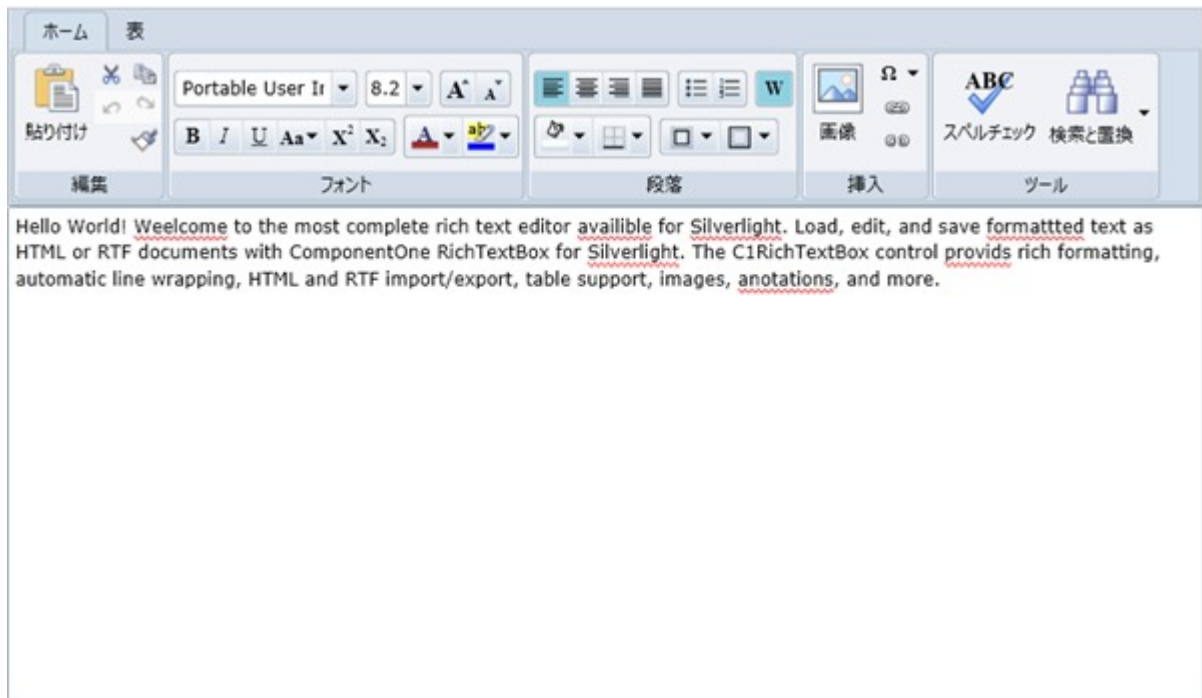
```
this.c1RichTextBox1.Text = "Hello World! Weelcome to the most complete rich text editor availible for WPF. Load, edit, and save formattted text as HTML or RTF documents with RichTextBox for WPF. The C1RichTextBox control provids rich formatting, automatic line wrapping, HTML and RTF import/export, table support, images, anotations, and more.";
```

このコードは、**C1RichTextBox** コントロールにコンテンツを追加します。テキストに意図的にスペルミスを入れてください。

## 手順3: 実行時のコントロールの使用

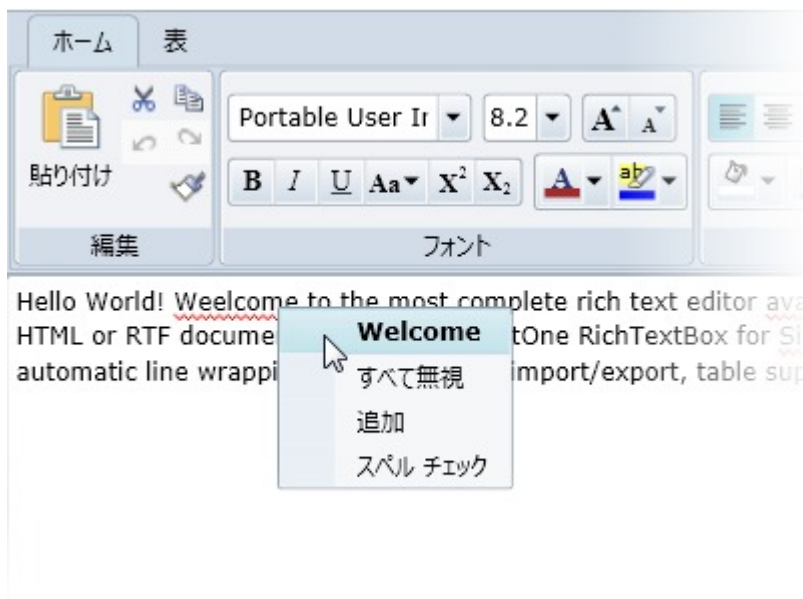
前の手順では、新しい WPF アプリケーションを作成し、アプリケーションに **C1RichTextBox** コントロールと **C1RichTextBoxToolbar** コントロールを追加し、スペルチェック機能を追加しました。アプリケーションを実行し、実行時に可能な操作を確認します。

1. **[F5]** キーを押してアプリケーションを実行します。

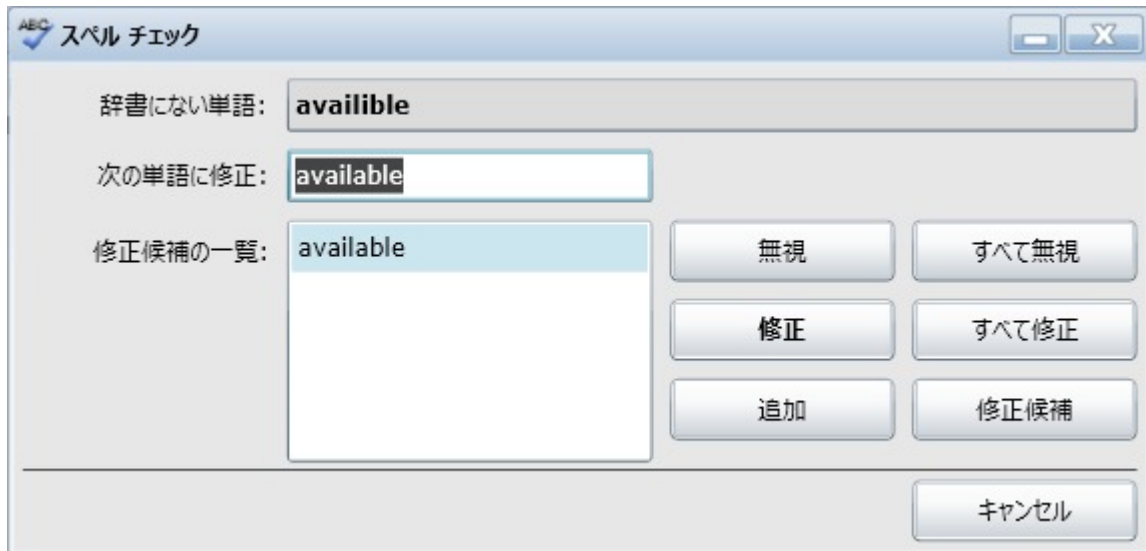


辞書にない単語の下に赤い波線が引かれ、入力時スペルチェックが実装されていることがわかります。

- 最初のスペルミスの単語「Weelcome」を右クリックし、表示されたオプションから正しいスペルを選択します。



- 「ツール」グループの「スペルチェック」ボタンをクリックします。「スペル」ダイアログボックスが表示されます。

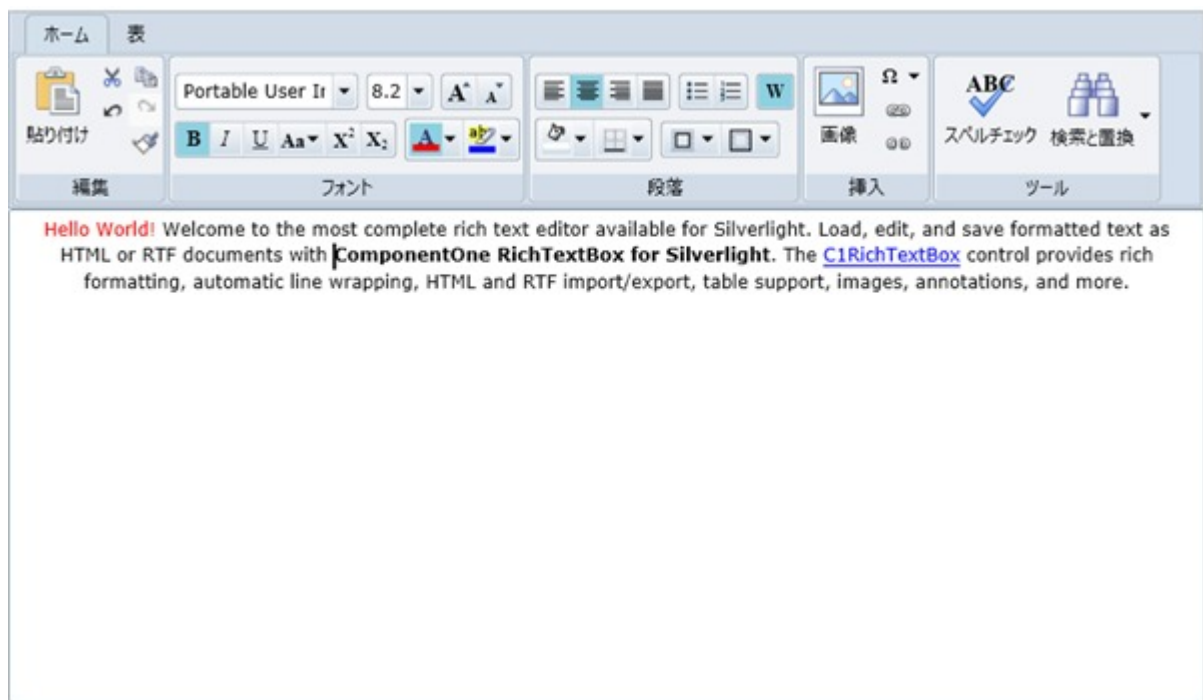


4. **【変更】** をクリックして、提案されたスペルを適用します。ダイアログボックスが次の単語に移動します。
5. **【スペル】** ダイアログボックスの **【追加】** をクリックして、辞書に「WPF」を追加します。
6. 以下の単語でそれぞれ **【変更】** をクリックして、提案されたスペルを適用します。
7. マウスを使用して「RichTextBox for WPF」を強調表示し、**【フォント】** グループの **【太字】** ボタンをクリックして、テキストを太字にします。
8. **【C1RichTextBox】** を強調表示し、**【挿入】** グループの **【ハイパーリンク】** ボタンをクリックして、**【ハイパーリンクの挿入】** ダイアログボックスを開きます。
9. **【ハイパーリンクの挿入】** ダイアログボックスの **【URL】** ボックスに、「<http://www.c1.grapecity.com/>」と入力し、**【OK】** をクリックしてダイアログを閉じます。リンクが追加されます。



このテキストは ComponentOne Web サイトへのリンクになります。

10. テキスト「Hello World!」を強調表示し、**【フォント】** グループの **【フォントの色】** ドロップダウンボックスをクリックし、**【赤】** を選択してテキストを赤にします。
11. 段落全体を強調表示し、**【段落】** グループの **【テキスト中央揃え】** ボタンをクリックして、テキストの配置を設定します。アプリケーションは次のようになります。



## ここまでの成果

おめでとうございます!

このチュートリアルは終了です。これで、**C1RichTextBox** コントロールと **C1RichTextBoxToolbar** コントロールの使い方少し学んで、実行時に可能な操作をいくつか確認しました。

## XAML クイックリファレンス

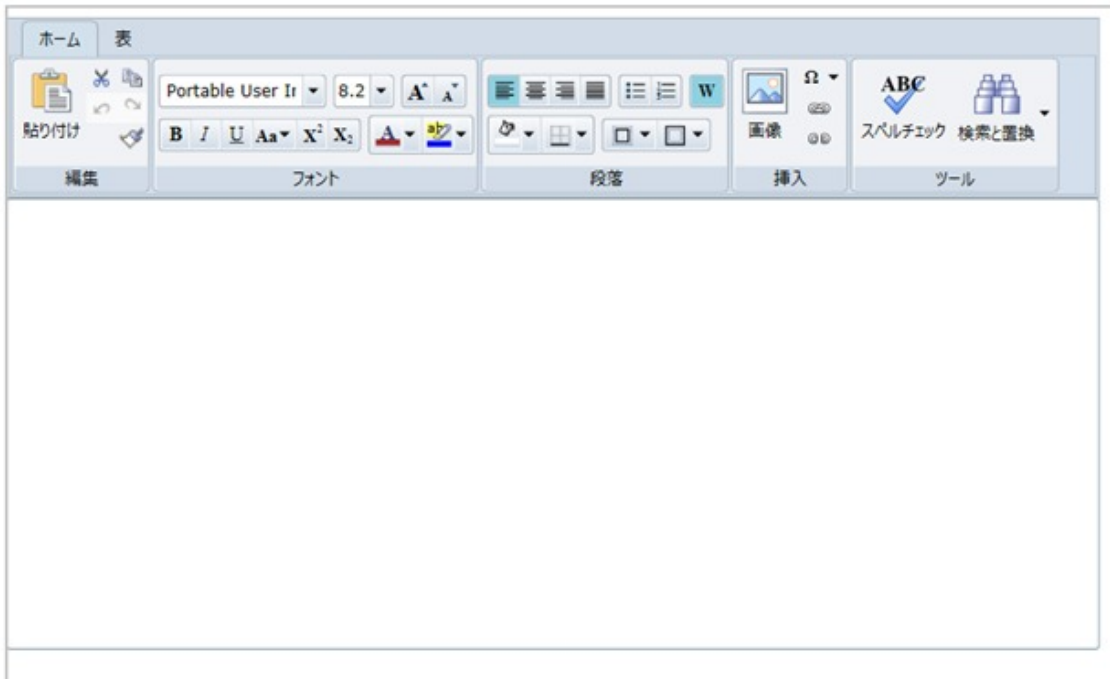
このトピックでは、**C1RichTextBox** および **C1RichTextBoxToolBar** コントロールの作成に使用される XAML の概要を提供します。

開発を開始するには、ルート要素タグに `c1` 名前空間宣言を追加します。

XAML

```
xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
```

これは **C1RichTextBox** および **C1RichTextBoxToolBar** のサンプルです。



このサンプルの XAML は次のようになります。

## WPF

```
<Window x:Class="RichTextBoxWPFXAML.MainWindow"
xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
Title="MainWindow"Height="370"Width="697"xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml">
<GridHeight="310">
<StackPanelHorizontalAlignment="Left"Name="SP"VerticalAlignment="Top"Height="327"Width="663">
<c1:C1RichTextBoxToolBarx:Name="richToolBar"RichTextBox="{BindingElementName=C1RTB}" />
<c1:C1RichTextBoxx:Name="C1RTB"Grid.Row="1"BorderThickness="0"Height="165" /> </StackPanel> </Grid>
</Window>
```

## Silverlight

```
<UserControl xmlns:c1=http://schemas.componentone.com/winfx/2006/xaml
x:Class="SilverlightApplication2.MainPage"
xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
xmlns:d=http://schemas.microsoft.com/expression/blend/2008
```

## RichTextBox for WPF

```
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006" mc:Ignorable="d"
d:DesignHeight="489" d:DesignWidth="668"> <Grid x:Name="LayoutRoot" Background="White">
<c1:C1RichTextBox Name="C1RTB" Margin="0,133,12,28" /> <c1:C1RichTextBoxToolbar
Name="richToolbar" RichTextBox="{Binding ElementName=C1RTB}" Margin="2,12,12,0"
Height="123" VerticalAlignment="Top" /> </Grid> </UserControl>
```



## RichTextBox for WPF の使い方

**RichTextBox for WPF** は、WPF で使用できる最も完成度が高いリッチテキストエディタです。書式設定されたテキストをロードして編集し、HTML ドキュメントまたは RTF ドキュメントとして保存できます。**C1RichTextBox** コントロールは、さまざまな書式設定、行の自動折り返し、HTML と RTF のインポート/エクスポート、テーブルのサポート、画像、注釈などを提供します。

**C1.WPF.RichTextBox.4** または **C1.Silverlight.RichTextBox.5** アセンブリには、**C1RichTextBox** コントロールと **C1Document** オブジェクトという 2 つの主要なオブジェクトがあります。

**C1RichTextBox** は、書式設定されたテキストを表示および編集するための強力なテキストエディタです。

**C1RichTextBox** は、フォント、背景色、前景色、リスト、ハイパーリンク、イメージ、境界などの一般的な書式設定オプションをすべてサポートします。**C1RichTextBox** は、HTML 形式のドキュメントのロードと保存もサポートします。

**C1Document** は、**C1RichTextBox** のコンテンツを表すクラスで、WPF の **FlowDocument** クラスに似ています。WPF と同様に、**C1Document** はいくつかの要素 (**C1Block** オブジェクト) 積み重ねて構成され、それぞれの要素はまたいくつかのインライン要素 (**C1Run** オブジェクト) で構成されています。

アプリケーションの多くはこの **C1RichTextBox** コントロールのみを処理し、このコントロールによってドキュメントは単純に線形的に表示されます。ドキュメント構造に完全にアクセスし、ドキュメントを直接作成したり管理するために、**C1Document** クラスが提供するリッチオブジェクトモデルを選択するアプリケーションもあります。

また、**C1RichTextBoxToolBar** コントロールや **C1SpellChecker** コンポーネントなどの関連する要素を使用して、**C1RichTextBox** コントロールの機能を拡張できます。**C1RichTextBoxToolBar** は、**C1.WPF.RichTextBox.Toolbar.4** または **C1.Silverlight.RichTextBox.Toolbar.5** アセンブリに含まれるリボン形式のツールバーです。**C1RichTextBoxToolBar** コントロールを追加して **C1RichTextBox** コントロールにリンクし、十分な機能を持つリッチテキストエディタを簡単に作成できます。**C1SpellChecker** は、**C1.WPF.SpellChecker.4** または **C1.Silverlight.SpellChecker.5** アセンブリにあり、これを使用してエディタにスペルチェック機能を追加できます。

## C1RichTextBox の概念と主要なプロパティ

一見、**C1RichTextBox** コントロールは標準の **TextBox** と同じように見えます。フォント、色、テキスト、および選択を制御するために、このコントロールにも同じプロパティがあります。これはメリットになる場合があります。たとえば、**TextBox** コントロールを使用するアプリケーションがあるとすると、何も変更することなくこれを **C1RichTextBox** コントロールに置き換えられる場合があります。

たとえば、次のコードは簡単な検索置換ルーチンですが、これは **TextBox** コントロールと **C1RichTextBox** コントロールの両方で動作します。

### VisualBasic

```
Private Sub SearchAndReplace(tb As TextBox, find As String, replace As String)
    Dim start As Integer = 0
    While True
        Dim pos As Integer = tb.Text.IndexOf(find, start)
        If pos < 0 Then
```

# RichTextBox for WPF

```
        Exit While
    End If
    tb.[Select](pos, find.Length)
    ' オプションで、変更を確認するダイアログボックスを表示します。
tb.SelectedText = replace
    start = pos + 1
End While
End Sub
```

## C#

```
void SearchAndReplace(TextBox tb, string find, string replace)
{
for (int start = 0; ; )
{
    int pos = tb.Text.IndexOf(find, start);
    if (pos < 0) break;
    tb.Select(pos, find.Length);
    // オプションで、変更を確認するダイアログボックスを表示します。
    tb.SelectedText = replace;
    start = pos + 1;
}
}
```

このコードは、**Text** プロパティ内で一致を検索し、**Select** メソッドを使用して一致をそれぞれ選択し、**SelectedText** プロパティを使用してそのテキストを置き換えます。このメソッドを **C1RichTextBox** コントロールで使用できるように変換するには、通常の **TextBox** の代わりに **C1RichTextBox** を使用できるように、最初の引数のタイプを変更するだけです。

これは、**C1RichTextBox** が通常の **TextBox** と共通しているところです。ただし、それを越える部分も当然あります。置換文字列を黄色の背景色で強調表示しようとしても、通常の **TextBox** では不可能です。**C1RichTextBox** を使用すると、コードを 1 行追加するだけでこれを実現できます。

## VisualBasic

```
Private Sub SearchAndReplace(tb As TextBox, find As String, replace As String)
    Dim start As Integer = 0
    While True
        Dim pos As Integer = tb.Text.IndexOf(find, start)
        If pos < 0 Then
            Exit While
        End If
        tb.[Select](pos, find.Length)
        ' オプションで、変更を確認するダイアログボックスを表示します。
        tb.Selection.InlineBackground = New SolidColorBrush(Colors.Yellow)
        tb.SelectedText = replace
    start = pos + 1
    End While
End Sub
```

## C#

```

Private Sub SearchAndReplace(tb As TextBox, find As String, replace As String)
    Dim start As Integer = 0
    While True
        Dim pos As Integer = tb.Text.IndexOf(find, start)
        If pos < 0 Then
            Exit While
        End If
        tb.[Select](pos, find.Length)
        ' オプションで、変更を確認するダイアログボックスを表示します。
        tb.Selection.InlineBackground = New SolidColorBrush(Colors.Yellow)
        tb.SelectedText = replace
    start = pos + 1
    End While
End Sub

```

**Selection** プロパティには、現在の選択範囲の書式を取得および変更するプロパティがあります。このプロパティや **TextBox** コントロールと共通するプロパティを使用して、簡単にドキュメントを作成してさまざまな書式を追加できます。

前述のテクニックを使用すると、ツールバーを実装したり、ドキュメントに構文の色指定を追加できます。これらのトピックの詳細については、後のセクションを参照してください。

## C1RichTextBox のコンテンツ

**C1RichTextBox** のコンテンツを指定するには2つの方法があります。それには、**Text** プロパティまたは **Html** プロパティを使用します。**Text** プロパティは、コントロールの内容をプレーンテキストとして割り当てたり取得するために使用されます。

### VisualBasic

```
Me.C1RichTextBox1.Text = "Hello World!"
```

## C#

```
this.c1RichTextBox1.Text = "Hello World!";
```

**Html** プロパティは、HTML として書式設定されたテキストを割り当てたり取得するために使用されます。HTML テキストは XAML ファイル内でエンコードする必要があるため、たとえば太字のタグは `<b>` ではなく `&lt;b&gt;` としてエンコードされます。

### VisualBasic

```
Me.C1RichTextBox1.Html = "<b>Hello World!</b>"
```

## C#

```
this.c1RichTextBox1.Html = "<b>Hello World!</b>"
```

**C1RichTextBox** は、コントロール内で長い行を折り返すか、それとも行を折り返さずに水平スクロールバーを表示するかを指定する **TextWrapping** プロパティを公開します。

## VisualBasic

```
Me.C1RichTextBox1.TextWrapping = TextWrapping.NoWrap
```

## C#

```
this.c1RichTextBox1.TextWrapping = TextWrapping.NoWrap;
```

上のコードでは、テキストコンテンツがコントロール内でラップすることなく単一の連続した行として表示されるように **C1RichTextBox** コントロールを設定しています。

## ロードと保存

単純な **TextBox** コントロールのコンテンツは、**Text** プロパティを使用して保存できます。**Text** プロパティを使用して **C1RichTextBox** コントロールのコンテンツを保存することもできますが、豊富な書式設定は失われます。代わりに、**Html** プロパティを使用すると、書式設定を維持したまま **C1RichTextBox** のコンテンツを保存できます。

**Html** プロパティは、**C1RichTextBox** の書式設定されたコンテンツを HTML 文字列として取得または設定します。**C1RichTextBox** に組み込まれた HTML フィルタはかなり機能が豊富で、CSS スタイル、イメージ、ハイパーリンク、リストなどをサポートします。ただし、このフィルタはすべての HTML をサポートするわけではなく、**C1RichTextBox** コントロール自体がサポートする機能に限定されます。たとえば、**C1RichTextBox** の現在のバージョンは、テーブルをサポートしていません。それでも、**Html** プロパティを使用してシンプルな HTML ドキュメントを表示できます。

**C1RichTextBox** に「Hello World」と入力すると、**Html** プロパティは次のマークアップを返します。

### HTML

```
<html>
<head>
  <style type="text/css">
    .c0 { font-family:Portable User Interface;font-size:9pt; }
    .c1 { margin-bottom:7.5pt; }
  </style>
</head>
<body class="c0">
<p class="c1">Hello world.</p>
</body>
```

```
</html>
```

**Html** プロパティは、HTML と内部 **C1Document** クラスの間のフィルタになることに注意してください。

**C1RichTextBox** によってサポートされていないコメント、メタ情報などの情報は HTML ストリームから破棄され、後でこの HTML ドキュメントを保存しても保持されません。

## Load

Once you set the content to the C1RichTextBox through Text or Html property, you can render the content in C1RichTextBox in html, RTF and Text formats. You can refer the below code for loading the content in the C1RichTextBox in various formats.

### HTML

```
//C1RichTextBoxのテキストを設定します
richTextBox.Text = "C1RichTextBoxsupports importing and exporting to HTMLand
RTF.";

//C1RichTextBoxテキストをHTML形式に変換します
richTextBox.Text = new
HtmlFilter().ConvertFromDocument(richTextBox.Document);
```

### RTF

```
//C1RichTextBoxのHTMLコンテンツを設定します
richTextBox.Html = @"<html>
<head>
<style>
td
{
border: solid 1px Red;
padding: 2px;
}
</style>
</head>
<body style=""font-family: Verdana; font-size: medium;"">
<p style='text-align:center; background-color:#FFFFCC; font-size:12pt'>
<b>C1RichTextBox</b> supports importing and exporting to <span style='text-
decoration: underline'>HTML</span>
and <span style='text-decoration: underline'>RTF</span>.</p>
</body>
</html>";

//C1RichTextBoxテキストをRTF形式に変換します
richTextBox.Text = new RtfFilter().ConvertFromDocument(richTextBox.Document);
```

### Text

```
//C1RichTextBoxのテキストを設定します
richTextBox.Text = "C1RichTextBoxsupports importing and exporting to HTMLand
RTF.";
```

## Export To Word

C1RichTextBox enables you to render the richtextbox document to Microsoft Word using the C1Word library. The rendered document contains all the set of properties that define the formatting of the content stored in the textbox.

To implement this feature, you need to add reference of C1.WPF.Word API to the application. To save the C1RichTextbox document as a word document, first convert it into a byte array by using the MemoryStream class. Then you can use the LoadFromRtf and Save methods of the C1WordDocument class for loading and saving the content as a word document. The below code snippet shows how to save the C1RichTextBox content as a word document.

C#

```
//C1RichTextBoxのHTMLコンテンツを設定します
richTextBox.Html = @"<html>
<head>
<style>
td
{
border: solid 1px Red;
padding: 2px;
}
</style>
</head>
<body style=""font-family: Verdana; font-size: medium;"">
<p style='text-align:center; background-color:#FFFFCC; font-size:12pt'>
<b>C1RichTextBox</b> supports importing and exporting to <span style='text-decoration:
underline'>HTML</span>
and <span style='text-decoration: underline'>RTF</span>.</p>
</body>
</html>";
//RTFフィルタを使用してHTMLコンテンツをRTF形式に変換します
var rtfText = new
C1.WPF.RichTextBox.Documents.RtfFilter().ConvertFromDocument(richTextBox.Document);

//C1WordDocumentのオブジェクトを作成します
C1WordDocument doc = new C1.WPF.Word.C1WordDocument();

//MemoryStreamクラスを使用してRTFテキストをバイト単位で取得します
var stream = new MemoryStream(Encoding.UTF8.GetBytes(rtfText));

//C1WordDocumentにRTFテキストをロードします
doc.LoadFromRtf(stream);

//コンテンツをWord文書形式で保存します
doc.Save("Document.docx");
```

## カスタムコマンドバーを作成する

**C1RichTextBox** コントロールに書式設定などの機能を適用するために、独自のツールバー、コンテキストメニュー、またはポップアップコントロールを作成することができます。次のセクションでは、最も基本的な書式設定コマンドを実行するために必要なコードについて説明します。ここでは、ツールバーや **AppBar** を設定するためのコードについては説明していません。

以下のコードスニペットでは、ページの **C1RichTextBox** コントロールの名前が `rtb` であることを前提としています。

## クリップボード機能

次のコードスニペットは、クリップボード機能に使用されるコードを示しています。

### コピー

```
C#
rtb.ClipboardCopy();
Paste
if(!rtb.IsReadOnly)
{
    rtb.ClipboardPaste();
}
```

### 貼り付け

```
Example Title
if(rt.IsReadOnly)
    rt.ClipboardCopy();
else
{
    rt.ClipboardCut();
}
```

## テキスト揃え機能

テキスト揃え機能 The following code snippets demonstrate the code used for aligning text:

### 左揃え

```
C#
rtb.Selection.TextAlignment = C1TextAlignment.Left;
```

### 中央揃え

```
C#
rtb.Selection.TextAlignment = C1TextAlignment.Center;
```

## 右揃え

```
C#
rtb.Selection.TextAlignment = C1TextAlignment.Right;
```

## 両端揃え

```
C#
rtb.Selection.TextAlignment = C1TextAlignment.Justify;
```

## フォント機能

次のコードスニペットは、フォント機能に使用されるコードを示しています。

### フォント ファミリ

```
C#
rtb.Selection.FontFamily = newFontFamily("Arial");
Font Size
rtb.Selection.TrimRuns();
foreach(varrun inrtb.Selection.Runs)
{
    run.FontSize = size;
}
```

## 書式設定機能

次のコードスニペットは、書式設定機能に使用されるコードを示しています。

### 前景色

```
C#
rtb.Selection.Foreground = newSolidColorBrush(Colors.Red);
Highlight (background) color
rtb.Selection.InlineBackground = newSolidColorBrush(Colors.Yellow);
Toggle Bold
if(rtbt.Selection.FontWeight != null&& rtbt.Selection.FontWeight.Value.Weight ==
FontWeights.Bold.Weight)
{
    rtbt.Selection.FontWeight = FontWeights.Normal;
}
else
{
    rtbt.Selection.FontWeight = FontWeights.Bold;
}
```



## 斜体の切り替え

```
C#  
if(rtb.Selection.FontStyle != null&& rtb.Selection.FontStyle == FontStyle.Italic)  
{  
    rtb.Selection.FontStyle = FontStyle.Normal;  
}  
else  
{  
    rtb.Selection.FontStyle = FontStyle.Italic;  
}
```

## 下線の切り替え

```
C#  
var range = rtb.Selection;  
var collection = new C1TextDecorationCollection();  
if (range.TextDecorations == null)  
{  
    collection.Add(C1TextDecorations.Underline[0]);  
}  
else if (!range.TextDecorations.Contains(C1TextDecorations.Underline[0]))  
{  
    foreach (var decoration in range.TextDecorations)  
        collection.Add(decoration);  
  
    collection.Add(C1TextDecorations.Underline[0]);  
}  
else  
{  
    foreach (var decoration in range.TextDecorations)  
        collection.Add(decoration);  
  
    collection.Remove(C1TextDecorations.Underline[0]);  
    if (collection.Count == 0)  
        collection = null;  
}  
range.TextDecorations = collection;
```

## 書式のクリア

```
C#  
rtb.Selection.InlineBackground = null;  
rtb.Selection.Foreground = rtb.Foreground;  
rtb.Selection.FontWeight = FontWeights.Normal;  
rtb.Selection.FontStyle = FontStyle.Normal;  
rtb.Selection.TextDecorations = null;
```

## テキスト選択機能

# RichTextBox for WPF

次のコードスニペットは、テキストの選択に使用されるコードを示しています。

## すべて選択

```
C#
rtb.SelectAll();
```

## ドキュメント履歴機能

次のスニペットは、ドキュメント履歴機能の作成に使用されるコードを示しています。

### 元に戻す

```
C#
if (rtb.DocumentHistory.CanUndo)
{
    rtb.DocumentHistory.Undo();
}
```

### やり直し

```
C#
if (rtb.DocumentHistory.CanRedo)
{
    rtb.DocumentHistory.Redo();
}
```

## ハイパーリンク

**C1RichTextBox** は、ハイパーリンクをサポートします。通常の HTML ドキュメントと同様に、この機能によってドキュメントの特定の部分をアクティブ化できます。ユーザーがハイパーリンクをクリックすると、アプリケーションは通知を受け取り、何らかのアクションを実行します。

次のコードに、ハイパーリンクを作成する方法を示します。

## VisualBasic

```
Public Sub New()
    InitializeComponent()
    ' テキストを設定します
    _rtb.Text = "This is some text with a hyperlink in it."
    ' ハイパーリンクを作成します
    Dim pos As Integer = _rtb.Text.IndexOf("hyperlink")
```

```

_rtb.[Select](pos, 9)
Dim uri = New Uri("https://www.grapecity.co.jp/developer/", UriKind.Absolute)
_rtb.Selection.MakeHyperlink(uri)
' ナビゲーション要求を処理します
_rtb.NavigationMode = NavigationMode.OnControlKey
AddHandler _rtb.RequestNavigate, AddressOf _rtb_RequestNavigate;
End Sub

```

## C#

```

public MainPage()
{
    InitializeComponent();
    // テキストを設定します
    _rtb.Text = "This is some text with a hyperlink in it.";
    // ハイパーリンクを作成します
    int pos = _rtb.Text.IndexOf("hyperlink");
    _rtb.Select(pos, 9);
    var uri = new Uri("https://www.grapecity.co.jp/developer/", UriKind.Absolute);
    _rtb.Selection.MakeHyperlink(uri);
    // ナビゲーション要求を処理します
    _rtb.NavigationMode = NavigationMode.OnControlKey;
    _rtb.RequestNavigate += _rtb_RequestNavigate;
}

```

このコードは、最初に **C1RichTextBox** にテキストを割り当てます。次に、"hyperlink" という単語を選択し、**MakeHyperlink** メソッドを呼び出してハイパーリンクにします。パラメータは、新しいハイパーリンクの **NavigateUri** プロパティに割り当てられる URI です。

その後、**NavigationMode** プロパティを設定して、ハイパーリンクの上にマウスポインタが置かれたときに **C1RichTextBox** がそれを処理する方法を決定します。デフォルトの動作は Microsoft Word と Visual Studio と同じで、[Ctrl] キーを押しながらハイパーリンクの上にマウスを移動すると、カーソルが手のアイコンに変化し、[Ctrl] キーを押しながらクリックすると、**RequestNavigate** イベントが発生します。これにより、ユーザーはハイパーリンクのテキストを通常のテキストと同様に編集できます。

**RequestNavigate** イベントハンドラは、ハイパーリンクナビゲーションを処理します。多くの場合、これには、新しいブラウザウィンドウを開き、別の URL に移動することが必要です。以下に例を示します。

## VisualBasic

```

Private Sub _rtb_RequestNavigate(sender As Object, e As RequestNavigateEventArgs)
    ' リンクを新しいウィンドウで開きます ("self" を指定すると現在のウィンドウが使用されます)
    Dim target As String = "_blank"
    System.Windows.Browser.HtmlPage.Window.Navigate(e.Hyperlink.NavigateUri, target)
End Sub

```

## C#

# RichTextBox for WPF

```
void _rtb_RequestNavigate(object sender, RequestNavigateEventArgs e)
{
    // リンクを新しいウィンドウで開きます ("self" を指定すると現在のウィンドウが使用されます)
    string target = "_blank";
    System.Windows.Browser.HtmlPage.Window.Navigate(e.Hyperlink.NavigateUri, target);
}
```

ハイパーリンクのアクションは、URI ナビゲーションに限定されません。一連のカスタム URI アクションをアプリケーション内でコマンドとして使用するように定義することもできます。**RequestNavigate** ハンドラで、これらのカスタム URI を解析して処理します。たとえば、次のコードは、ハイパーリンクを使用してメッセージボックスを表示します。

## VisualBasic

```
Public Sub New()
    InitializeComponent()
    ' テキストを設定します
    _rtb.Text = "ハイパーリンクに指定する任意テキスト"
    ' ハイパーリンクを作成します
    Dim pos As Integer = _rtb.Text.IndexOf("hyperlink")
    _rtb.[Select](pos, 9)
    Dim uri = New Uri("msgbox:Thanks for clicking!")
    _rtb.Selection.MakeHyperlink(uri)
    ' ナビゲーション要求を処理します
    _rtb.NavigationMode = NavigationMode.OnControlKey
    AddHandler _rtb.RequestNavigate, AddressOf _rtb_RequestNavigate
End Sub
Private Sub _rtb_RequestNavigate(sender As Object, e As RequestNavigateEventArgs)
    Dim uri As Uri = e.Hyperlink.NavigateUri
    If uri.Scheme = "msgbox" Then
        MessageBox.Show(uri.LocalPath)
    End If
End Sub
```

## C#

```
public MainPage()
{
    InitializeComponent();
    // テキストを設定します
    _rtb.Text = "This is some text with a hyperlink in it.";
    // ハイパーリンクを作成します
    int pos = _rtb.Text.IndexOf("hyperlink");
    _rtb.Select(pos, 9);
    var uri = new Uri("msgbox:Thanks for clicking!");
    _rtb.Selection.MakeHyperlink(uri);
    // ナビゲーション要求を処理します
    _rtb.NavigationMode = NavigationMode.OnControlKey;
    _rtb.RequestNavigate += _rtb_RequestNavigate;
}
```

```

}
void _rtb_RequestNavigate(object sender, RequestNavigateEventArgs e)
{
    Uri uri = e.Hyperlink.NavigateUri;
    if (uri.Scheme == "msgbox")
    {
        MessageBox.Show(uri.LocalPath);
    }
}

```

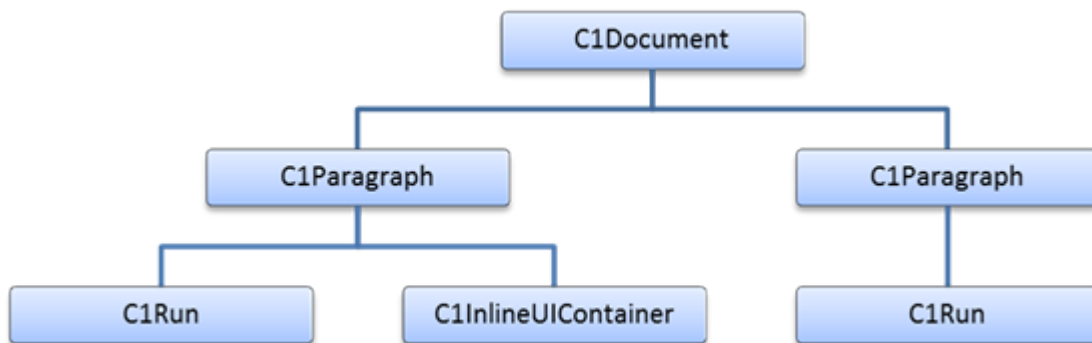
**MakeHyperlink** コード内の変更点は、URI を作成する行だけです。**RequestNavigate** ハンドラは、URI メンバを使用してコマンドと引数を解析します。このテクニックを使用すると、たとえば埋め込みメニューを含むドキュメントを作成できます。

**CreateHyperlink** メソッドは、ドキュメント内の既存の部分をハイパーリンクに簡単迅速に変更する方法にすぎません。**C1Hyperlink** 要素を **C1Document** オブジェクトに追加して、ハイパーリンクを作成することもできます。これについては、後のセクションで説明します。

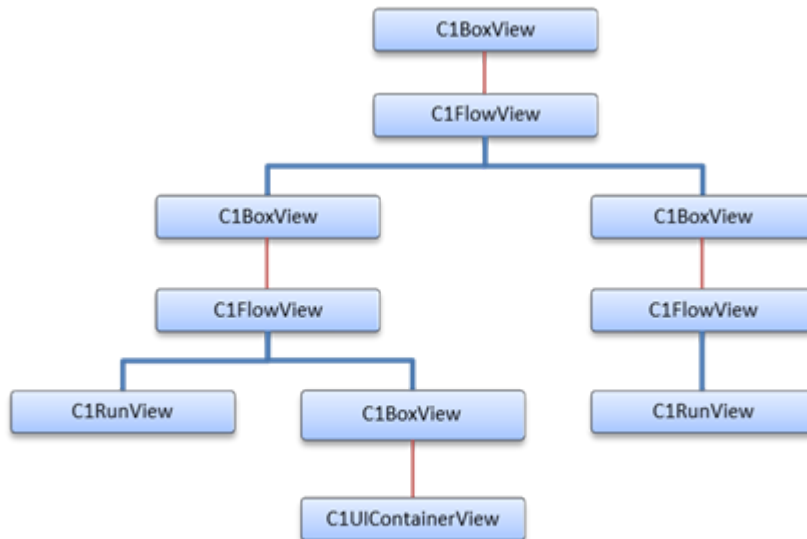
## レイアウト情報へのアクセス

**C1RichTextBox** によって **C1Document** レイアウトを作成すると、**C1TextElementView** オブジェクトで構成されるツリーが平行して作成されます。**C1Document** ツリー内の各 **C1TextElement** には、レイアウトと描画の役割を担う **C1TextElementView** が1つ以上含まれます。

この **C1Document** ツリーを例にします。



対応するビューツリーは次のようになります。



各 **C1TextElementView** は、対応する **C1TextElement** の基本レイアウト情報を提供します。

- **Origin:** ドキュメント座標によるビューの原点です。
- **DesiredSize:** 最後に測定されたときからのビューに必要なサイズです。

レイアウトと描画を処理するために、外側の **C1TextElement** を複数の **C1TextElementView** で構成することもできます。その場合、**Content** プロパティには、構成内の最も内側の **C1TextElementView** が含まれます。コンテンツビューの子は、関連付けられている **C1TextElement** の **Children** コレクションに対応します。

ビューの構成は、**Content** ビューのマージン、パディング、および境界を処理するために **C1BoxView** で使用されます。つまり、各 **C1BoxView** の原点はマージン、パディング、および境界ボックスの外側で、**Content** の原点は内側です。

**C1FlowView** は、いくつかの行としてフローするボックスやテキストを表します。各行は、**C1Line** オブジェクトによって表されます。**C1Line** には、単一行のテキストだけでなく、段落全体が含まれる場合もあります。

各 **C1FlowView** には **C1Line** のリストが含まれ、これらの行は常に垂直方向に積み重ねられます。各 **C1Line** はいくつかの **C1LineFragment** で構成され、これらの行フラグメントは水平方向に積み重ねられます。

**C1LineFragment** には子要素の参照が含まれ、その原点はフラグメントの位置に一致します。

たとえば、次のコードは **C1RichTextBox** 内の行数をカウントします。

## VisualBasic

```
Private Function CountLines(rtb As C1RichTextBox) As Integer
    Dim root = rtb.ViewManager.GetView(rtb.Document)
    Return CountLines(root)
End Function
Private Function CountLines(view As C1TextElementView) As Integer
    Dim count As Integer = 0
    Dim flow = TryCast(view, C1FlowView)
    If flow IsNot Nothing Then
        For Each line As var In flow.Lines
```

```

        If TypeOf line.Fragments.First().Element Is C1Inline Then
            count += 1
        End If
    Next
End If
For Each child As var In view.Children
    count += CountLines(child)
Next
Return count
End Function

```

## C#

```

int CountLines(C1RichTextBox rtb)
{
    var root = rtb.ViewManager.GetView(rtb.Document);
    return CountLines(root);
}
int CountLines(C1TextElementView view)
{
    int count = 0;
    var flow = view as C1FlowView;
    if (flow != null)
    {
        foreach (var line in flow.Lines)
        {
            if (line.Fragments.First().Element is C1Inline)
            {
                ++count;
            }
        }
    }
    foreach (var child in view.Children)
    {
        count += CountLines(child);
    }
    return count;
}

```

まず、ルートビューを取得します。これはルート要素に関連付けられているビューと同じなので、**GetView** を使用して **rtb.Document** のビューを取得します。次に、ビューツリーを走査して、各 **C1FlowView** で見つかった行数がカウントされます。**C1Inline** 要素の行数のみをカウントすることに注意してください。そうしないと、段落などのコンテナブロックもカウントしてしまいます。

## ペインタ

ペインタは、**C1RichTextBox** がテキストを表示するキャンバスと同じキャンバスに **UIElement** を表示することで、**C1RichTextBox** 拡張します。これは、スタイルのオーバーライドより汎用的な拡張を可能にしますが、使い方は多少難しくなります。**C1RichTextBox** では、選択範囲を表示するためにペインタを内部的に使用していま

# RichTextBox for WPF

す。

ペインタは、**IRichTextPainter** インタフェースの実装です。このインタフェースには **Paint** と **PaintInline** の2つのメソッドがあり、それぞれ **C1RichTextBox** の描画パスの異なる段階で呼び出されます。各メソッドは、描画されるビューポートのチェックに使用される **C1PaintingContext** オブジェクトを受け取り、カスタム **UIElement** を描画するためのメソッドを持ちます。

**Paint** は、画面全体が再描画されるたびに呼び出されます。このメソッドが呼び出されるたびに、**Paint** を呼び出すことによってすべての **UIElements** を描画する必要があります。そうしないと、要素が削除されます。毎回同じ **UIElement** を渡す方が、ビジュアルツリーから削除されないのが効率的です。

**PaintInline** は、描画される **C1Line** ごとに呼び出されます。このメソッドを使用すると、カスタム **UIElement** を描画するレイヤを細かく制御できます。たとえば、要素をテキストの背景の上で、かつテキスト自体の下に描画することができます。ルールは、**Paint** と同じです。**PaintInline** を呼び出して、すべての **UIElement** を描画する必要があります。そうしないと、要素が削除されます。

**Annotations** サンプルは、ペインタを使用して付箋を表示します。実装は次のとおりです。

## VisualBasic

```
Class StickyPainter
    Implements IRichTextPainter
    Private _stickies As List(Of StickyNote)
    Public Sub New(stickies As List(Of StickyNote))
        _stickies = stickies
    End Sub
    Public Sub Paint(context As C1PaintingContext)
        For Each sticky As var In _stickies
            Dim rect = context.ViewManager.GetRectFromPosition(sticky.Range.Start)
            context.Paint(Math.Round(rect.X), Math.Round(rect.Bottom), False, sticky)
        Next
    End Sub
    Public Sub PaintInline(context As C1PaintingContext, line As C1Line)
    End Sub
    Public Event PainterChanged As EventHandler(Of RichTextPainterChangeEventArgs)
End Class
```

## C#

```
class StickyPainter : IRichTextPainter
{
    List<StickyNote> _stickies;
    public StickyPainter(List<StickyNote> stickies)
    {
        _stickies = stickies;
    }
    public void Paint(C1PaintingContext context)
    {
        foreach (var sticky in _stickies)
        {
```



```

        var rect = context.ViewManager.GetRectFromPosition(sticky.Range.Start);
        context.Paint(Math.Round(rect.X), Math.Round(rect.Bottom), false, sticky);
    }
}
public void PaintInline(C1PaintingContext context, C1Line line)
{
}
public event EventHandler<RichTextPainterChangeEventArgs> PainterChanged;
}

```

**StickyPainter** は、**Paint** メソッドのみを使用します。このメソッドは、各付箋に対してドキュメント内の座標を取得し、**Paint** を呼び出すだけです。これは、ページング、スクロール、およびズームに左右されないドキュメント座標です。

## スペルチェック

高性能なエディタの多くは、次の2種類のスペルチェックを実装しています。

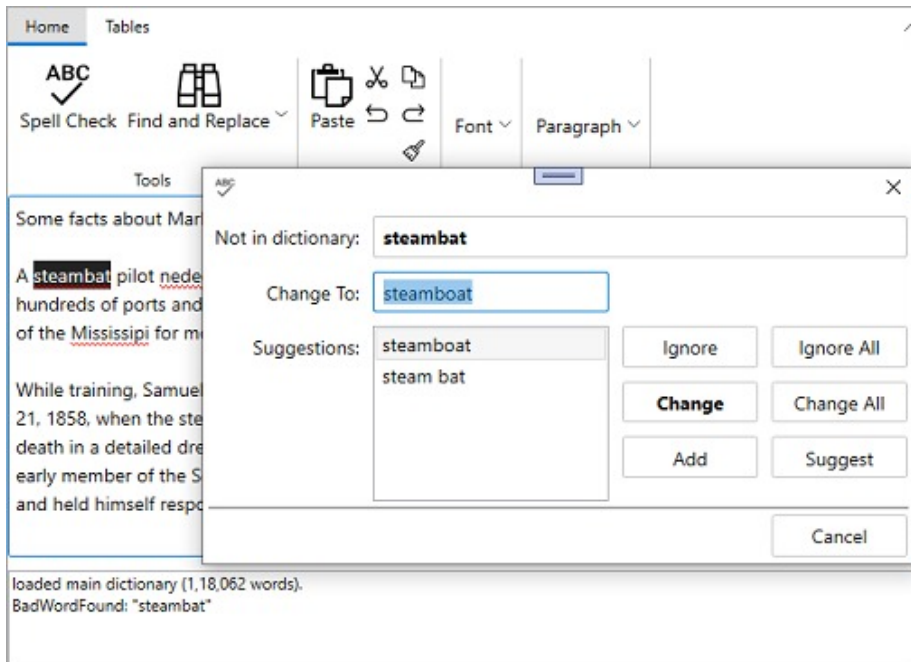
- **モーダルスペルチェック**: **[スペル]** ダイアログボックスを表示して、ドキュメント内のスペルミスを選択します。スペルミスを見逃したり、正しいスペルを入力するか候補リストから選択してミスを変更したり、その単語を辞書に追加することができます。
- **入力中スペルチェック**: 入力中にスペルミスを通常は赤い波線の下線で強調表示します。ドキュメント内のスペルミスを右クリックすると、見逃す、辞書に追加する、スペル候補から選択してスペルミスを自動的に修正するなどのオプションメニューが表示されます。

**C1RichTextBox** は、**C1SpellChecker** コンポーネントを使用した2種類のスペルチェックをサポートします。このコンポーネントも、ComponentOne for WPF に含まれています。**C1SpellChecker** は、他のコントロールもスペルチェックできるため、別のアセンブリとしてリリースされています。

## Modal Spell Checking

To implement modal spell checking, you need to add a reference to the **C1.WPF.SpellChecker** assembly to your project. Then, add the following code to your project. This code creates a new **C1SpellChecker** object to be shared by all controls on the page that require spell-checking. Later, the page constructor invokes the **Load** method to load the main spelling dictionary from a stream containing the compressed Dictionary data. **C1SpellChecker** includes over 20 other dictionaries which can be downloaded from our site. In this case, we are loading **C1Spell\_en-US.dct**, the American English dictionary. This file must be present on the application folder. When the modal checking is complete, the **\_c1SpellChecker\_CheckControlCompleted** event fires and shows a dialog box to indicate that the spell-checking operation is complete.

# RichTextBox for WPF



C#

```
public partial class SpellCheckerRichTextBoxDemo : UserControl
{
    // C1SpellCheckerを宣言します
    C1SpellChecker _c1SpellChecker = new C1SpellChecker();

    public SpellCheckerRichTextBoxDemo()
    {
        InitializeComponent();
        this.Tag = Properties.Resources.SpellCheckerRtbDemoDescription;
        Loaded += Page_Loaded;
        Unloaded += Page_Unloaded;
    }

    void Page_Loaded(object sender, RoutedEventArgs e)
    {
        // C1RichTextBoxにツールバーを接続します
        _rtbToolBar.RichTextBox = _richTextBox;
        _richTextBox.SpellChecker = _c1SpellChecker;

        // サンプルテキストをテキストボックスにロードします
        using (var stream =
Assembly.GetExecutingAssembly().GetManifestResourceStream("SpellCheckerExplorer.Resources.test.txt"))
        using (var sr = new StreamReader(stream))
        {
            var text = sr.ReadToEnd();
            _richTextBox.Text = text;
        }

        // 無視リストを設定します
        WordList il = _c1SpellChecker.IgnoreList;
        il.Add("ComponentOne");
        il.Add("Silverlight");

        // イベントを監視します
        _c1SpellChecker.BadWordFound += _c1SpellChecker_BadWordFound;
    }
}
```

```

        _c1SpellChecker.CheckControlCompleted += _c1SpellChecker_CheckControlCompleted;

        // メイン辞書をロードします
        if (_c1SpellChecker.MainDictionary.State != DictionaryState.Loaded)
            _c1SpellChecker.MainDictionary.Load(Application.GetResourceStream(new Uri("/") + new
AssemblyName(Assembly.GetExecutingAssembly().FullName).Name + ";component/Resources/C1Spell_en-
US.dct", UriKind.Relative)).Stream);
        if (_c1SpellChecker.MainDictionary.State == DictionaryState.Loaded)
        {
            WriteLine("loaded main dictionary ({0:n0} words).",
_c1SpellChecker.MainDictionary.WordCount);
        }
        else
        {
            WriteLine("failed to load dictionary: {0}", _c1SpellChecker.MainDictionary.State);
        }

        // アプリの終了時にユーザー辞書を保存します
        App.Current.Exit += App_Exit;
    }

    void Page_Unloaded(object sender, RoutedEventArgs e)
    {
        _c1SpellChecker.BadWordFound -= _c1SpellChecker_BadWordFound;
        _c1SpellChecker.CheckControlCompleted -= _c1SpellChecker_CheckControlCompleted;
    }

    // スペルチェッカーイベントを監視します
    void _c1SpellChecker_CheckControlCompleted(object sender, CheckControlCompletedEventArgs e)
    {
        if (!e.Cancelled)
        {
            var msg = string.Format("Spell-check complete, {0} errors found.", e.ErrorCount);
            MessageBox.Show(msg, "Spelling");
        }
        WriteLine("CheckControlCompleted: {0} errors found", e.ErrorCount);
        if (e.Cancelled)
        {
            WriteLine("\t(cancelled...)");
        }
    }
    void _c1SpellChecker_BadWordFound(object sender, BadWordEventArgs e)
    {
        WriteLine("BadWordFound: \"{0}\" {1}", e.BadWord.Text, e.BadWord.Duplicate ? "
(duplicate)" : string.Empty);
    }
}

```

## 構文の色指定

「[C1TextPointer の理解](#)」では、**Selection** プロパティを使用して現在の選択範囲に対応する **C1TextRange** オブジェクトを取得する方法について説明し、そのオブジェクトを使用してカスタム書式を検証し、ドキュメントの各部に適用する方法についても説明します。

ただし、範囲を選択することなく書式を検証したり範囲に適用したい場合もあります。**Selection** プロパティを使

# RichTextBox for WPF

用してこれを実行するには、現在の選択範囲を保存し、すべての書式を適用してから、元の選択範囲を復元する必要があります。また、選択範囲を変更すると、新しい選択範囲を表示するために、ドキュメントがスクロールする可能性があります。

このような状況に対処するために、**C1RichTextBox** は **GetTextRange** メソッドを公開しています。

**GetTextRange** メソッドは、現在の選択範囲に影響を与えずに **C1TextRange** を返します。

たとえば、**GetTextRange** メソッドを使用して、**C1RichTextBox** に HTML 構文の色指定を追加することもできます。最初の手順は、ドキュメントに対する変更を検出することです。この変更により、実際に構文の色指定を行うメソッドがトリガされます。

## VisualBasic

```
' タイマーによって構文の色指定を更新します
Private _updating As Boolean
Private _syntax As Storyboard
' ドキュメントが変更されるたびにタイマーをオンにします
Private Sub tb_TextChanged(sender As Object, e As C1TextChangedEventArgs)
    If Not _updating Then
        ' ストーリーボードが null の場合は、それを作成します
        If _syntax Is Nothing Then
            _syntax = New Storyboard()
            AddHandler _syntax.Completed, AddressOf _syntax_Completed
            _syntax.Duration = New Duration(TimeSpan.FromMilliseconds(1000))
        End If
        ' ストーリーボードを再開します
        _syntax.[Stop]()
        _syntax.Seek(TimeSpan.Zero)
        _syntax.Begin()
    End If
End Sub
' タイマーの時間が経過し、構文の色指定を更新します
Private Sub _syntax_Completed(sender As Object, e As EventArgs)
    _updating = True
    UpdateSyntaxColoring(_rtb)
    _updating = False
End Sub
```

## C#

```
// タイマーによって構文の色指定を更新します
bool _updating;
Storyboard _syntax;
// ドキュメントが変更されるたびにタイマーをオンにします
void tb_TextChanged(object sender, C1TextChangedEventArgs e)
{
    if (!_updating)
    {
        // ストーリーボードが null の場合は、それを作成します
        if (_syntax == null)
```

```

{
    _syntax = new Storyboard();
    _syntax.Completed += _syntax_Completed;
    _syntax.Duration = new Duration(TimeSpan.FromMilliseconds(1000));
}
// ストーリーボードを再開します
_syntax.Stop();
_syntax.Seek(TimeSpan.Zero);
_syntax.Begin();
}
}
// タイマーの時間が経過し、構文の色指定を更新します
void _syntax_Completed(object sender, EventArgs e)
{
    _updating = true;
    UpdateSyntaxColoring(_rtb);
    _updating = false;
}

```

このコードは、ユーザーがドキュメントを変更するたびにオンになるタイマーを作成します。タイマーがオンになっているときにユーザーがドキュメントを変更すると、タイマーはリセットされます。これで、ユーザーがすばやく入力している間にコードが頻繁に構文の色指定を更新することがなくなります。

タイマーの時間が経過すると、構文の色指定の更新中に行われた変更によってタイマーがオンにならないようにするフラグを設定し、**UpdateSyntaxColoring** メソッドを呼び出します。

## VisualBasic

```

' 構文の色指定します
Private Sub UpdateSyntaxColoring(rtb As C1RichTextBox)
    ' HTML の解析に使用される正規表現を初期化します
    Dim pattern As String = "</?(?<tagName>[a-zA-Z0-9_:\-]+)" & "(\s+(?<attName>[a-zA-Z0-9_:\-]+)(?<attValue>(=""[^"]*"")?))*\s*/?>"
    ' ドキュメントの色指定に使用されるブラシを初期化します
    Dim brDarkBlue As Brush = New SolidColorBrush(Color.FromArgb(255, 0, 0, 180))
    Dim brDarkRed As Brush = New SolidColorBrush(Color.FromArgb(255, 180, 0, 0))
    Dim brLightRed As Brush = New SolidColorBrush(Colors.Red)
    ' 以前の色指定を削除します
    Dim input = rtb.Text
    Dim range = rtb.GetTextRange(0, input.Length)
    range.Foreground = rtb.Foreground
    ' 一致を強調表示します
    For Each m As Match In Regex.Matches(input, pattern)
        ' タグ全体を選択して濃い青色にします
        range = rtb.GetTextRange(m.Index, m.Length)
        range.Foreground = brDarkBlue
        ' タグ名を選択して濃い赤色にします
        Dim tagName = m.Groups("tagName")
        range = rtb.GetTextRange(tagName.Index, tagName.Length)
        range.Foreground = brDarkRed
    ' 属性名を選択して明るい赤色にします

```

# RichTextBox for WPF

```
Dim attGroup = m.Groups("attName")
If attGroup IsNot Nothing Then
    Dim atts = attGroup.Captures
    For i As Integer = 0 To atts.Count - 1
        Dim att = atts(i)
        range = rtb.GetTextRange(att.Index, att.Length)
        range.Foreground = brLightRed
    Next
End If
Next
End Sub
```

## C#

```
// 構文を色指定します
void UpdateSyntaxColoring(C1RichTextBox rtb)
{
    // HTML の解析に使用される正規表現を初期化します
    string pattern =
        @"</?(?<tagName>[a-zA-Z0-9_:\-]+)" +
        @"(\s+(?<attName>[a-zA-Z0-9_:\-]+)(?<attValue>(=""[^""]+"")?)*)\s*/?>";
    // ドキュメントの色指定に使用されるブラシを初期化します
    Brush brDarkBlue = new SolidColorBrush(Color.FromArgb(255, 0, 0, 180));
    Brush brDarkRed = new SolidColorBrush(Color.FromArgb(255, 180, 0, 0));
    Brush brLightRed = new SolidColorBrush(Colors.Red);
    // 以前の色指定を削除します
    var input = rtb.Text;
    var range = rtb.GetTextRange(0, input.Length);
    range.Foreground = rtb.Foreground;
    // 一致を強調表示します
    foreach (Match m in Regex.Matches(input, pattern))
    {
        // タグ全体を選択して濃い青色にします
        range = rtb.GetTextRange(m.Index, m.Length);
        range.Foreground = brDarkBlue;
        // タグ名を選択して濃い赤色にします
        var tagName = m.Groups["tagName"];
        range = rtb.GetTextRange(tagName.Index, tagName.Length);
        range.Foreground = brDarkRed;
        // 属性名を選択して明るい赤色にします
        var attGroup = m.Groups["attName"];
        if (attGroup != null)
        {
            var atts = attGroup.Captures;
            for (int i = 0; i < atts.Count; i++)
            {
                var att = atts[i];
                range = rtb.GetTextRange(att.Index, att.Length);
                range.Foreground = brLightRed;
            }
        }
    }
}
```

```
}
}
```

最初にコードは、HTML を解析するための正規表現パターンを定義します。これは、HTML を解析するための最も効率的な方法ではなく、表現も格別に読みやすく保守しやすいというわけではありません。サンプルコード以外で HTML の解析に正規表現を使用することはお勧めしません。サンプルコードでは、正規表現によってコードがコンパクトになり、理解しやすくなります。

次の手順では、残っている色指定を削除します。それには、ドキュメント全体をカバーする範囲を作成し、その **Foreground** プロパティを **C1RichTextBox** コントロールの **Foreground** に一致するように設定します。

次に、正規表現を使用してドキュメントを解析します。このコードは、各一致候補をスキャンし、**C1TextRange** オブジェクトを作成し、**Foreground** プロパティを目的の値に設定します。HTML タグには濃い青色、タグ名には濃い赤色、および属性名には明るい赤色を使用します。

必要なコードはこれだけです。次の画像は、今作成した構文の色指定を **C1RichTextBox** に適用し、その中に HTML ドキュメントを表示したところです。

```
<html xmlns="http://www.w3.org/1999/xhtml" style="height:100%;">
<head id="Head1" runat="server">
  <title>Silverlight QuickStart</title>
</head>
<body style="height:100%;margin:0;overflow:hidden">
  <form id="form1" runat="server" style="height:100%;">
    <asp:ScriptManager ID="ScriptManager1" runat="server"></asp:ScriptManager>
    <div style="height:100%;">
      <asp:Silverlight
        ID="Xaml1"
        Windowless="True"
        runat="server"
        Source="~/ClientBin/QuickStart.xap"
        MinimumVersion="2.0.30523"
        Width="100%" Height="100%" />
    </div>
  </form>
</body>
</html>
```

任意の HTML テキストをコントロールに入力するか、貼り付けて、アプリケーションをテストしてください。入力を止めるとすぐに、入力したテキストに自動的に色が付けられます。

実際のアプリケーションでは、テキスト変更の種類を検出し、ドキュメントの小さな部分の色指定を更新することで、構文の色指定プロセスを最適化できます。さらにスタイルシート、コメントなどの要素を検出し、通常は正規表現の代わりに専用のパーサーを使用します。

ただし、基本的なメカニズムは同じです。ドキュメント内の範囲を検出し、**C1TextRange** オブジェクトを取得し、書式設定を適用します。

## スタイルのオーバーライド

「**構文の色指定**」では、**C1TextRange** オブジェクトを使用し、選択範囲を移動することなくドキュメントの一部のスタイルを変更する方法について説明しました。ただし、ドキュメント自体ではなく、ビューのみを変更する必要がある場合もあります。

# RichTextBox for WPF

たとえば、現在の選択範囲は、異なる前景色と背景色で強調表示されます。これは、ドキュメント自体ではなくビューに属するスタイルの変更です。他の例としては、構文の色指定や入力中のスペルチェックがあります。

**C1RichTextBox** コントロールは、**StyleOverrides** プロパティを使用してこのようなシナリオをサポートします。このプロパティには、ビューにのみ適用される範囲とスタイルの変更を指定するためのオブジェクトのコレクションが含まれます。この方法には、前のセクションで行った **C1TextRange** オブジェクトにスタイルの変更を適用する方法に比べて、次の2つのメリットがあります。

- スタイルのオーバーライドはドキュメントに適用されず、ドキュメントを HTML として保存する際も適用されません（通常、現在の選択範囲やスペルミスのインジケータをファイルに保存する必要はありません）。
- この方法は、ドキュメントに変更を加えず、また現在可視の部分にのみ影響するので、**C1TextRange** オブジェクトを直接変更する方法よりはるかに効率的です。

この方法には、ドキュメントフローに影響するスタイル要素をスタイルの変更に入れることができないという制限があります。スタイルのオーバーライドを使用して、背景や前景を変更したり、ドキュメントの一部に下線を引くことができます。ただし、フォントのサイズやスタイルの変更は、ドキュメントフローに影響する可能性があるので実行できません。

ここでは、前の構文の色指定の例をスタイルのオーバーライドの使用例に変えて示します。

まず、**C1RangeStyleCollection** オブジェクトを宣言し、これをコントロールの **StyleOverrides** コレクションに追加する必要があります。これで、コレクションに追加されたすべてのオーバーライドがコントロールに適用されるようになります。後で、ドキュメントの構文の色指定部分をコレクションに挿入します。

## VisualBasic

```
Private _rangeStyles As New C1RangeStyleCollection()
Public Sub New()
    InitializeComponent()
    _rtb = New C1RichTextBox()
    LayoutRoot.Children.Add(_rtb)
    AddHandler _rtb.TextChanged, AddressOf tb_TextChanged
    _rtb.FontFamily = New FontFamily("Courier New")
    _rtb.FontSize = 16
    _rtb.Text = GetStringResource("w3c.htm")
    ' C1RangeStyleCollection をコントロールの
    ' StyleOverrides コレクションに追加します
    _rtb.StyleOverrides.Add(_rangeStyles)
End Sub
```

## C#

```
C1RangeStyleCollection _rangeStyles = new C1RangeStyleCollection();
public MainPage()
{
    InitializeComponent();
    _rtb = new C1RichTextBox();
```



```

LayoutRoot.Children.Add(_rtb);
_rtb.TextChanged += tb_TextChanged;
_rtb.FontFamily = new FontFamily("Courier New");
_rtb.FontSize = 16;
_rtb.Text = GetStringResource("w3c.htm");
// C1RangeStyleCollection をコントロールの
// StyleOverrides コレクションに追加します
_rtb.StyleOverrides.Add(_rangeStyles);
}

```

ここで必要なことは、以前に示した **UpdateSyntaxColoring** メソッドを変更して、(以前のようにドキュメントに色指定を適用する代わりに) そこでコレクションに範囲スタイルを挿入します。

## VisualBasic

```

' StyleOverrides コレクションを使用して構文を色指定します
' (デフォルトドキュメントの強調表示には数分の1秒かかります)
PrivateSub UpdateSyntaxColoring(ByVal rtb As C1RichTextBox)
' HTML の解析に使用される正規表現を初期化します
String pattern =
    "</?(?<tagName>[a-zA-Z0-9_:\-]+)" +
    "(\\s+(?<attName>[a-zA-Z0-9_:\-]+)" +
    "(?<attValue>(\\s*=\\s*\"\"(^[^\"]+\"\")))*\\s*/?>"
' ドキュメントの色指定に使用されるスタイルを初期化します
Dim key As var =C1TextElement.ForegroundProperty
Dim brDarkBlue As var =New C1TextElementStyle()
brDarkBlue(key) = New SolidColorBrush(Color.FromArgb(255, 0, 0, 180))
Dim brDarkRed As var =New C1TextElementStyle()
brDarkRed(key)= New SolidColorBrush(Color.FromArgb(255, 180, 0, 0))
Dim brLightRed As var =New C1TextElementStyle()
brLightRed(key) = New SolidColorBrush(Colors.Red)
' 以前の色指定を削除します
_rangeStyles.Clear()
' 一致を強調表示します
Dim input As var =rtb.Text
Dim m As Match
For Each m In Regex.Matches(input,pattern)
' タグ全体を選択して濃い青色にします
Dim range As var =rtb.GetTextRange(m.Index,m.Length)
_rangeStyles.Add(New C1RangeStyle(range,brDarkBlue))
' タグ名を選択して濃い赤色にします
Dim tagName As var =m.Groups("tagName")
range = rtb.GetTextRange(tagName.Index, tagName.Length)
_rangeStyles.Add(New C1RangeStyle(range,brDarkRed))
' 属性名を選択して明るい赤色にします
Dim attGroup As var =m.Groups("attName")
If Not attGroup Is Nothing Then
Dim att As Capture
For Each att In attGroup.Captures
range = rtb.GetTextRange(att.Index, att.Length)
_rangeStyles.Add(New C1RangeStyle(range,brLightRed))

```

# RichTextBox for WPF

```
Next
End If
Next
End Sub
```

## C#

```
// StyleOverrides コレクションを使用して構文を色指定します
// (デフォルトドキュメントの強調表示には数分の1秒かかります)
void UpdateSyntaxColoring(C1RichTextBox rtb)
{
    // HTML の解析に使用される正規表現を初期化します
    string pattern =
        @"</?(?<tagName>[a-zA-Z0-9_:\-]+)" +
        @"(\s+(?<attName>[a-zA-Z0-9_:\-]+)" +
        "(?<attValue>(\s*=\s*\"[^\"]+\"|\"?\"?))*\s*/?>";
    // ドキュメントの色指定に使用されるスタイルを初期化します
    var key = C1TextElement.ForegroundProperty;
    var brDarkBlue= new C1TextElementStyle();
    brDarkBlue[key] = new SolidColorBrush(Color.FromArgb(255, 0, 0, 180));
    var brDarkRed = new C1TextElementStyle();
    brDarkRed[key]= new SolidColorBrush(Color.FromArgb(255, 180, 0, 0));
    var brLightRed= new C1TextElementStyle();
    brLightRed[key] = new SolidColorBrush(Colors.Red);
    // 以前の色指定を削除します
    _rangeStyles.Clear();
    // 一致を強調表示します
    var input = rtb.Text;
    foreach (Match m in Regex.Matches(input, pattern))
    {
        // タグ全体を選択して濃い青色にします
        var range = rtb.GetTextRange(m.Index, m.Length);
        _rangeStyles.Add(new C1RangeStyle(range, brDarkBlue));
        // タグ名を選択して濃い赤色にします
        var tagName = m.Groups["tagName"];
        range = rtb.GetTextRange(tagName.Index, tagName.Length);
        _rangeStyles.Add(new C1RangeStyle(range, brDarkRed));
        // 属性名を選択して明るい赤色にします
        var attGroup = m.Groups["attName"];
        if (attGroup != null)
        {
            foreach (Capture att in attGroup.Captures)
            {
                range = rtb.GetTextRange(att.Index, att.Length);
                _rangeStyles.Add(new C1RangeStyle(range, brLightRed));
            }
        }
    }
}
```

変更されたコードは、元のコードとほとんど同じです。ドキュメントの色指定のためのブラシを作成する代わりに

に、前景のプロパティをオーバーライドした **C1TextElementStyle** オブジェクトを作成します。このコードは、最初にオーバーライドコレクションをクリアし、次に正規表現を使用してドキュメント内の各 HTML タグの場所を特定し、最後に範囲を **C1TextElementStyle** オブジェクトに関連付ける **C1RangeStyle** オブジェクトをオーバーライドコレクションに挿入します。

この新しいコードを実行すると、パフォーマンスが格段に向上していることがわかります。新しいコードは、元のコードより数千倍高速です。

## ヒットテスト

**C1RichTextBox** は、ユーザーの対話式操作を実装するための標準メカニズムとして[ハイパーリンク](#)をサポートします。さらに、カスタムのマウス対話機能を追加して提供することもできます。たとえば、ユーザーが要素をクリックしたときに、カスタム書式を適用したり、コンテキストメニューを表示することができます。

このようなシナリオを可能にするために、**C1RichTextBox** には、**ElementMouse\*** イベントと **GetPositionFromPoint** メソッドが公開されています。

マウスイベントをトリガした要素を確認するだけでよい場合は、イベントハンドラの **source** パラメータから取得できます。より詳細な情報（要素内でクリックされた単語など）が必要な場合は、**GetPositionFromPoint** メソッドが必要です。**GetPositionFromPoint** はクライアント座標でポイントを受け取り、その位置をドキュメント座標で表す **C1TextPosition** オブジェクトを返します。

**C1TextPosition** オブジェクトには、**Element** と **Offset** という2つの主要なプロパティがあります。**Element** プロパティはドキュメント内の要素を表し、**Offset** プロパティは文字インデックス（要素が **C1Run** の場合）または指定されたポイントにある子要素のインデックスです。

たとえば、次のコードは、**C1RichTextBox** を作成し、ハンドラを **ElementMouseLeftButtonDown** イベントに関連付けます。

## VisualBasic

```
Public Sub New()
    ' デフォルトの初期化
    InitializeComponent()
    ' C1RichTextBox を作成してページに追加します
    _rtb = New C1RichTextBox()
    LayoutRoot.Children.Add(_rtb)
    ' イベントハンドラを関連付けます
    Add Handler _rtb.ElementMouseLeftButtonDown AddressOf rtb_ElementMouseLeftButtonDown
End Sub
```

## C#

```
public MainPage()
{
    // デフォルトの初期化
    InitializeComponent();
    // C1RichTextBox を作成してページに追加します
    _rtb = new C1RichTextBox();
```

# RichTextBox for WPF

```
LayoutRoot.Children.Add(_rtb);  
// イベントハンドラを関連付けます  
_rtb.ElementMouseDown += rtb_ElementMouseDown;  
}
```

次のイベントハンドラは、クリックされた要素全体に対して **FontWeight** プロパティを切り替えます。この要素は、単語、文、または段落全体の場合があります。

## VisualBasic

```
Private Sub _rtb_ElementMouseDown(sender As Object, e As MouseButtonEventArgs)  
    If Keyboard.Modifiers <> 0 Then  
        Dim run = TryCast(sender, C1Run)  
        If run IsNot Nothing Then  
            run.FontWeight = If(run.FontWeight = FontWeights.Bold, FontWeights.Normal,  
FontWeights.Bold)  
        End If  
    End If  
End Sub
```

## C#

```
void _rtb_ElementMouseDown(object sender, MouseButtonEventArgs e)  
{  
    if (Keyboard.Modifiers != 0)  
    {  
        var run = sender as C1Run;  
        if (run != null)  
        {  
            run.FontWeight = run.FontWeight == FontWeights.Bold  
                ? FontWeights.Normal  
                : FontWeights.Bold;  
        }  
    }  
}
```

このコードは、**sender** パラメータを **C1Run** オブジェクトにキャストして、クリックされた要素を取得します。

単語の **FontWeight** value を切り替える場合は、クリックされた文字を特定してから選択範囲を単語全体に拡張する必要があります。ここで **GetPositionFromPoint** メソッドが必要になります。次に、これを行うイベントハンドラの改訂バージョンを示します。

## VisualBasic

```
Private Sub _rtb_ElementMouseDown(sender As Object, e As MouseButtonEventArgs)  
    If Keyboard.Modifiers <> 0 Then  
        ' コントロール座標で位置を取得します  
        Dim pt = e.GetPosition(_rtb)
```

```

' その位置のテキストポインタを取得します
Dim pointer = _rtb.GetPositionFromPoint(pt)
' ポインタが C1Run をポイントしていることを確認します
Dim run = TryCast(pointer.Element, C1Run)
If run IsNot Nothing Then
    ' C1Run 内の単語を取得します
    Dim text = run.Text
    Dim start = pointer.Offset
    Dim [end] = pointer.Offset
    While start > 0 AndAlso Char.IsLetterOrDigit(text, start - 1)
        start -= 1
    End While
    While [end] < text.Length - 1 AndAlso Char.IsLetterOrDigit(text, [end] + 1)
        [end] += 1
    End While
    ' クリックされたランの太字プロパティを切り替えます
    Dim word = New C1TextRange(pointer.Element, start, [end] - start + 1)
    word.FontWeight = If(word.FontWeight.HasValue AndAlso word.FontWeight.Value =
FontWeights.Bold, FontWeights.Normal, FontWeights.Bold)
    End If
End If
End Sub

```

## C#

```

void _rtb_ElementMouseDown(object sender, MouseButtonEventArgs e)
{
if (Keyboard.Modifiers != 0)
{
// コントロール座標で位置を取得します
var pt = e.GetPosition(_rtb);
// その位置のテキストポインタを取得します
var pointer = _rtb.GetPositionFromPoint(pt);
// ポインタが C1Run をポイントしていることを確認します
var run = pointer.Element as C1Run;
if (run != null)
{
// C1Run 内の単語を取得します
var text = run.Text;
var start = pointer.Offset;
var end = pointer.Offset;
while (start > 0 && char.IsLetterOrDigit(text, start - 1))
start--;
while (end < text.Length - 1 && char.IsLetterOrDigit(text, end + 1))
end++;
// クリックされたランの太字プロパティを切り替えます
var word = new C1TextRange(pointer.Element, start, end - start + 1);
word.FontWeight =
word.FontWeight.HasValue && word.FontWeight.Value == FontWeights.Bold
? FontWeights.Normal
: FontWeights.Bold;
}
}
}

```

```
}  
}  
}
```

**FontWeight** プロパティは、null 可能な値を返します。この範囲内で、この属性に複数の値が混ざって含まれる場合、このプロパティは null を返します。**FontWeight** プロパティを切り替えるために使用するコードは、以前に書式設定ツールバーを実装したときに使用したコードと同じです。

**GetPositionFromPoint** を使用すると、画面上のポイントから **C1TextPosition** オブジェクトを取得できます。**GetRectFromPosition** メソッドは逆の操作を実行します。すなわち、**C1TextPosition** オブジェクトの画面上の位置を表す Rect を返します。これは、ドキュメントの特定部分の近くに UI 要素を表示するような場合に役立ちます。

## HtmlFilter のカスタマイズ

**HtmlFilter** は、HTML 文字列と **C1Document** を相互に変換するための **C1RichTextBox** のコンポーネントです。また、**C1HtmlDocument** という HTML ドキュメントの中間表現と相互に変換することもできます。

**C1HtmlDocument** と **C1Document** の間で変換する際は、いくつかのイベントが発生するので、変換される各ノードをカスタマイズできます。以下のイベントがあります。

- **ConvertingHtmlNode**: このイベントは、HTML ノードが変換される直前に発生します。イベントハンドラで処理済みのマークが付けられていると、**HtmlFilter** はそのノードが変換されたと見なし、処理をスキップします。
- **ConvertedHtmlNode**: このイベントは、ノードが変換された後に発生します。このイベントは、変換結果に多少の変更を加える場合に使用できます。
- **ConvertingTextElement**: このイベントは、**C1TextElement** が変換される直前に発生します。イベントハンドラで処理済みのマークが付けられていると、**HtmlFilter** はその要素が変換されたと見なし、処理をスキップします。
- **ConvertedTextElement**: このイベントは、**C1TextElement** が変換された後に発生します。このイベントは、変換結果に多少の変更を加える場合に使用できます。

例として、C1.WPF.Imaging.4 または C1.Silverlight.Imaging.5 を使用して GIF イメージのサポートを追加している **HtmlFilterCustomization** サンプルを見てみます。このサンプルは、**ConvertingHtmlNode** イベントと **ConvertingTextElement** イベントの両方を使用します。これは **ConvertingHtmlNode** イベントハンドラです。

## VisualBasic

```
Private Sub HtmlFilter_ConvertingHtmlNode(sender As Object, e As  
ConvertingHtmlNodeEventArgs)  
    Dim htmlElement = TryCast(e.HtmlNode, C1HtmlElement)  
    If htmlElement IsNot Nothing AndAlso htmlElement.Name = "img" Then  
        Dim src As String  
        If htmlElement.Attributes.TryGetValue("src", src) Then  
            Dim uri = New Uri("/HtmlFilterCustomization;component/" & src, UriKind.Relative)
```

```

Dim resource = Application.GetResourceStream(uri)
If resource IsNot Nothing Then
    Dim imageSource = New C1Bitmap(resource.Stream).ImageSource
    Dim image = New Image() With { _
        Key .Source = imageSource _
    }
    SetImageSource(image, src)
    e.Parent.Children.Add(New C1InlineUIContainer() With { _
        Key .Child = image _
    })
    e.Handled = True
End If
End If
End If
End Sub

```

## C#

```

void HtmlFilter_ConvertingHtmlNode(object sender, ConvertingHtmlNodeEventArgs e)
{
    var htmlElement = e.HtmlNode as C1HtmlElement;
    if (htmlElement != null && htmlElement.Name == "img")
    {
        string src;
        if (htmlElement.Attributes.TryGetValue("src", out src))
        {
            var uri = new Uri("/HtmlFilterCustomization;component/" + src, UriKind.Relative);
            var resource = Application.GetResourceStream(uri);
            if(resource != null)
            {
                var imageSource = new C1Bitmap(resource.Stream).ImageSource;
                var image = new Image { Source = imageSource };
                SetImageSource(image, src);
                e.Parent.Children.Add(new C1InlineUIContainer { Child = image });
                e.Handled = true;
            }
        }
    }
}

```

このイベントハンドラは最初に、**e.HtmlNode** を **C1HtmlElement** にキャストします。**C1HtmlNode** を継承する型は2つあります。**C1HtmlElement** はなどの HTML 要素を表し、**C1HtmlText** はテキストノードを表します。

**C1HtmlNode** オブジェクトが **C1HtmlElement** にキャストされると、タグ名を調べて属性にアクセスできるようになります。そこで、要素が実際に IMG タグかどうかを確認し、SRC 属性を取得します。コードの残りの部分では、適切な要素を作成して e.Parent に追加しています。SRC 値は、エクスポートする際にアクセスできるように添付プロパティとして保存されます。

変換が完了すると、ハンドラは、**HtmlFilter** がこの **C1HtmlNode** を変換できないようにするために、e.Handled を True に設定します。

# RichTextBox for WPF

**ConvertingTextElement** イベントハンドラは、次のようになります。

## VisualBasic

```
Private Sub HtmlFilter_ConvertingTextElement(sender As Object, e As
ConvertingTextElementEventArgs)
    Dim inlineContainer = TryCast(e.TextElement, C1InlineUIContainer)
    If inlineContainer IsNot Nothing Then
        Dim src = GetImageSource(inlineContainer.Child)
        If src IsNot Nothing Then
            Dim element = New C1HtmlElement("img")
            element.Attributes("src") = src
            e.Parent.Add(element)
            e.Handled = True
        End If
    End If
End Sub
```

## C#

```
void HtmlFilter_ConvertingTextElement(object sender, ConvertingTextElementEventArgs e)
{
    var inlineContainer = e.TextElement as C1InlineUIContainer;
    if (inlineContainer != null)
    {
        var src = GetImageSource(inlineContainer.Child);
        if (src != null)
        {
            var element = new C1HtmlElement("img");
            element.Attributes["src"] = src;
            e.Parent.Add(element);
            e.Handled = true;
        }
    }
}
```

これは、**C1TextElement** を **C1HtmlElement** に変換すること以外は、もう一方のハンドラとほとんど同じです。添付プロパティから SRC 値が復元され、その属性を使用して **C1HtmlElement** が作成されます。前と同様に、新しい要素が **e.Parent** に追加され、イベントは **Handled** としてマークされます。



## C1Document オブジェクトの使い方

ここまでは、**C1RichTextBox** コントロールのオブジェクトモデルに注目してきました。ただし、このコントロールは **C1Document** オブジェクトの編集可能なビューということにすぎません。このオブジェクトは、基底のドキュメントを作成および編集するための機能豊富なオブジェクトモデルを公開しています。このアーキテクチャは、**FlowDocument** オブジェクトのビューを提供する Microsoft WPF **RichTextBox** コントロールのアーキテクチャに似ています。

レポートの生成、インポートフィルタやエクスポートフィルタの実装などの多くのタスクの実行には、**C1Document** オブジェクトの直接的なプログラミングが最も適しています。たとえば、Html プロパティは、**C1Document** オブジェクトと HTML 文字列を相互に変換するメソッドを **HTML** フィルタとして公開しています。RTF、PDF などの他の一般的な形式をインポートおよびエクスポートするための同様のフィルタクラスを実装することもできます。

**C1RichTextBox** は主にテキストを処理します。このコントロールは、コントロールコンテンツのフラットで線形的なビューを提供します。一方、**C1Document** は、ドキュメントの構造を公開します。このドキュメントモデルにより、各段落、各リスト内の項目などに含まれるランを簡単に列挙できます。これについては、後のセクションで説明します。

## ドキュメントとレポートの作成

**C1Document** を作成するプロセスの例を示すために、簡単なアセンブリドキュメントユーティリティを実装するために必要な手順について説明します。

まず、新しいプロジェクトを作成し、**C1.WPF.4** または **C1.Silverlight.4** アセンブリと **C1.WPF.RichTextBox.4** または **C1.Silverlight.RichTextBox.5** アセンブリへの参照を追加します。次に、ページのコンストラクタを次のように編集します。

## VisualBasic

```
Imports C1.WPF
Imports C1.WPF.RichTextBox
Imports C1.WPF.RichTextBox.Documents
Public Partial Class MainPage
    Inherits UserControl
    ' C1Document ドキュメントを表示する C1RichTextBox
    Private _rtb As C1RichTextBox
    Public Sub New()
        ' デフォルトの初期化
        InitializeComponent()
        ' C1RichTextBox を作成してページに追加します
        _rtb = New C1RichTextBox()
        LayoutRoot.Children.Add(_rtb)
        ' ドキュメントを作成して C1RichTextBox に表示します
        _rtb.Document = DocumentAssembly(GetType(C1RichTextBox).Assembly)
        _rtb.IsReadOnly = True
    End Sub
End Class
```

## C#

```
using C1.WPF;
using C1.WPF.RichTextBox;
using C1.WPF.RichTextBox.Documents;
public partial class MainPage : UserControl
{
    // C1Document ドキュメントを表示する C1RichTextBox
    C1RichTextBox _rtb;
    public MainPage()
    {
        // デフォルトの初期化
        InitializeComponent();
        // C1RichTextBox を作成してページに追加します
        _rtb = new C1RichTextBox();
        LayoutRoot.Children.Add(_rtb);
        // ドキュメントを作成して C1RichTextBox に表示します
        _rtb.Document = DocumentAssembly(typeof(C1RichTextBox).Assembly);
        _rtb.IsReadOnly = true;
    }
}
```

このコードは、**C1RichTextBox** を作成し、その **Document** プロパティに **DocumentAssembly** メソッドの呼び出しの結果を割り当てます。次に、ユーザーがレポートを変更できないようにコントロールを読み取り専用にします。

**DocumentAssembly** メソッドは、**Assembly** を引数として受け取り、アセンブリドキュメントを含む **C1Document** を構築します。実装は次のとおりです。

## VisualBasic

```
Private Function DocumentAssembly(asm As Assembly) As C1Document
    ' ドキュメントを作成します
    Dim doc As New C1Document()
    doc.FontFamily = New FontFamily("Tahoma")
    ' アセンブリ
    doc.Blocks.Add(New Heading1("Assembly" & vbCrLf & vbCrLf + asm.FullName.Split(",")(0)))
    ' タイプ
    For Each t As Type In asm.GetTypes()
        DocumentType(doc, t)
    Next
    ' Done
    Return doc
End Function
```

## C#

```
C1Document DocumentAssembly(Assembly asm)
```

```

{
    // ドキュメントを作成します
    C1Document doc = new C1Document();
    doc.FontFamily = new FontFamily("Tahoma");
    // アセンブリ
    doc.Blocks.Add(new Heading1("Assembly\r\n" + asm.FullName.Split(',')[0]));
    // タイプ
    foreach (Type t in asm.GetTypes())
        DocumentType(doc, t);
    // Done
    return doc;
}

```

このメソッドは、最初に新しい **C1Document** オブジェクトを作成し、その **FontFamily** プロパティを設定します。これは、ドキュメントに追加されるすべてのテキスト要素のデフォルト値になります。

次に、このメソッドは、アセンブリ名を含む **Heading1** 段落を新しいドキュメントの **Blocks** コレクションに追加します。ブロックは、ドキュメントに縦方向に並べられる段落、リスト項目などの要素から成ります。ブロックは、HTML の "div" 要素に似ています。ドキュメント要素には、代わりに **Inlines** コレクションが含まれる場合もあります。これらのコレクションには、HTML の "span" 要素のように、横方向に並べられる要素が含まれます。

**Heading1** クラスは **C1Paragraph** を継承し、書式設定を追加します。ここでは、通常の段落と見出し 1～4 に対応するこのようなクラスをいくつかプロジェクトに追加します。

**Normal** 段落は、コンストラクタでコンテンツ文字列を受け取る **C1Paragraph** です。

## VisualBasic

```

Class Normal
    Inherits C1Paragraph
    Public Sub New(text As String)
        Me.Inlines.Add(New C1Run() With { _
            Key .Text = text _
        })
        Me.Padding = New Thickness(30, 0, 0, 0)
        Me.Margin = New Thickness(0)
    End Sub
End Class

```

## C#

```

class Normal : C1Paragraph
{
    public Normal(string text)
    {
        this.Inlines.Add(new C1Run() { Text = text });
        this.Padding = new Thickness(30, 0, 0, 0);
        this.Margin = new Thickness(0);
    }
}

```

# RichTextBox for WPF

Heading 段落は、Normal を拡張してテキストを太字にします。

## VisualBasic

```
Class Heading
    Inherits Normal
    Public Sub New(text As String)
        MyBase.New(text)
        Me.FontWeight = FontWeights.Bold
    End Sub
End Class
```

## C#

```
class Heading : Normal
{
    public Heading(string text) : base(text)
    {
        this.FontWeight = FontWeights.Bold;
    }
}
```

Heading1 から Heading4 は、Heading を拡張してフォントサイズ、パディング、境界、および色を指定します。

## VisualBasic

```
Class Heading1
    Inherits Heading
    Public Sub New(text As String)
        MyBase.New(text)
        Me.Background = New SolidColorBrush(Colors.Yellow)
        Me.FontSize = 24
        Me.Padding = New Thickness(0, 10, 0, 10)
        Me.BorderBrush = New SolidColorBrush(Colors.Black)
        Me.BorderThickness = New Thickness(3, 1, 1, 0)
    End Sub
End Class

Class Heading2
    Inherits Heading
    Public Sub New(text As String)
        MyBase.New(text)
        Me.FontSize = 18
        Me.FontStyle = FontStyles.Italic
        Me.Background = New SolidColorBrush(Colors.Yellow)
        Me.Padding = New Thickness(10, 5, 0, 5)
        Me.BorderBrush = New SolidColorBrush(Colors.Black)
        Me.BorderThickness = New Thickness(3, 1, 1, 1)
    End Sub
End Class
```

```

    End Sub
End Class
Class Heading3
    Inherits Heading
    Public Sub New(text As String)
        MyBase.New(text)
        Me.FontSize = 14
        Me.Background = New SolidColorBrush(Colors.LightGray)
        Me.Padding = New Thickness(20, 3, 0, 0)
    End Sub
End Class
Class Heading4
    Inherits Heading
    Public Sub New(text As String)
        MyBase.New(text)
        Me.FontSize = 14
        Me.Padding = New Thickness(30, 0, 0, 0)
    End Sub
End Class

```

## C#

```

class Heading1 : Heading
{
    public Heading1(string text) : base(text)
    {
        this.Background = new SolidColorBrush(Colors.Yellow);
        this.FontSize = 24;
        this.Padding = new Thickness(0, 10, 0, 10);
        this.BorderBrush = new SolidColorBrush(Colors.Black);
        this.BorderThickness = new Thickness(3, 1, 1, 0);
    }
}
class Heading2 : Heading
{
    public Heading2(string text): base(text)
    {
        this.FontSize = 18;
        this.FontStyle = FontStyles.Italic;
        this.Background = new SolidColorBrush(Colors.Yellow);
        this.Padding = new Thickness(10, 5, 0, 5);
        this.BorderBrush = new SolidColorBrush(Colors.Black);
        this.BorderThickness = new Thickness(3, 1, 1, 1);
    }
}
class Heading3 : Heading
{
    public Heading3(string text) : base(text)
    {
        this.FontSize = 14;
        this.Background = new SolidColorBrush(Colors.LightGray);
    }
}

```

# RichTextBox for WPF

```
        this.Padding = new Thickness(20, 3, 0, 0);
    }
}
class Heading4 : Heading
{
    public Heading4(string text): base(text)
    {
        this.FontSize = 14;
        this.Padding = new Thickness(30, 0, 0, 0);
    }
}
```

これでドキュメント内の各段落タイプに対応するクラスを設定したので、次にコンテンツを追加します。最初のコードブロックで **DocumentType** メソッドを使用したことを思い出してください。次にそのメソッドの実装を示します。

## VisualBasic

```
Private Sub DocumentType(doc As C1Document, t As Type)
    ' 非パブリック/ジェネリックはスキップします
    If Not t.IsPublic OrElse t.ContainsGenericParameters Then
        Return
    End If
    ' タイプ
    doc.Blocks.Add(New Heading2("Class " & Convert.ToString(t.Name)))
    ' プロパティ
    doc.Blocks.Add(New Heading3("Properties"))
    For Each pi As PropertyInfo In t.GetProperties()
        If pi.DeclaringType = t Then
            DocumentProperty(doc, pi)
        End If
    Next
    ' メソッド
    doc.Blocks.Add(New Heading3("Methods"))
    For Each mi As MethodInfo In t.GetMethods()
        If mi.DeclaringType = t Then
            DocumentMethod(doc, mi)
        End If
    Next
    ' イベント
    doc.Blocks.Add(New Heading3("Events"))
    For Each ei As EventInfo In t.GetEvents()
        If ei.DeclaringType = t Then
            DocumentEvent(doc, ei)
        End If
    Next
End Sub
```

## C#

```

void DocumentType(C1Document doc, Type t)
{
    // 非パブリック/ジェネリックはスキップします
    if (!t.IsPublic || t.ContainsGenericParameters)
        return;
    // タイプ
    doc.Blocks.Add(new Heading2("Class " + t.Name));
    // プロパティ
    doc.Blocks.Add(new Heading3("Properties"));
    foreach (PropertyInfo pi in t.GetProperties())
    {
        if (pi.DeclaringType == t)
            DocumentProperty(doc, pi);
    }
    // メソッド
    doc.Blocks.Add(new Heading3("Methods"));
    foreach (MethodInfo mi in t.GetMethods())
    {
        if (mi.DeclaringType == t)
            DocumentMethod(doc, mi);
    }
    // イベント
    doc.Blocks.Add(new Heading3("Events"));
    foreach (EventInfo ei in t.GetEvents())
    {
        if (ei.DeclaringType == t)
            DocumentEvent(doc, ei);
    }
}

```

このメソッドは、クラス名を含む **Heading2** 段落を追加し、次にリフレクションを使用してすべてのパブリックプロパティ、イベント、およびメソッドを列挙します。これらのメソッドのコードは簡単です。

## VisualBasic

```

Private Sub DocumentProperty(doc As C1Document, pi As PropertyInfo)
    If pi.PropertyType.ContainsGenericParameters Then
        Return
    End If
    doc.Blocks.Add(New Heading4(pi.Name))
    Dim text = String.Format("public {0} {1} {{ {2}{3} }}", pi.PropertyType.Name, pi.Name,
    If(pi.CanRead, "get; ", String.Empty), If(pi.CanWrite, "set; ", String.Empty))
    doc.Blocks.Add(New Normal(text))
End Sub

```

## C#

```

void DocumentProperty(C1Document doc, PropertyInfo pi)
{
    if (pi.PropertyType.ContainsGenericParameters)

```

# RichTextBox for WPF

```
return;
doc.Blocks.Add(new Heading4(pi.Name));
var text = string.Format("public {0} {1} {{ {2}{3} }}",
    pi.PropertyType.Name,
    pi.Name,
    pi.CanRead ? "get; " : string.Empty,
    pi.CanWrite ? "set; " : string.Empty);
doc.Blocks.Add(new Normal(text));
}
```

このメソッドは、プロパティ名を含む **Heading4** 段落を追加し、次にプロパティタイプ、名前、およびアクセサを含む **Normal** テキストを追加します。

イベントとプロパティを記述するために使用するメソッドは似ています。

## VisualBasic

```
Private Sub DocumentMethod(doc As C1Document, mi As MethodInfo)
    If mi.IsSpecialName Then
        Return
    End If
    doc.Blocks.Add(New Heading4(mi.Name))
    Dim parms = New StringBuilder()
    For Each parm As var In mi.GetParameters()
        If parms.Length > 0 Then
            parms.Append(", ")
        End If
        parms.AppendFormat("{0} {1}", parm.ParameterType.Name, parm.Name)
    Next
    Dim text = String.Format("public {0} {1}({2})", mi.ReturnType.Name, mi.Name,
parms.ToString())
    doc.Blocks.Add(New Normal(text))
End Sub
Private Sub DocumentEvent(doc As C1Document, ei As EventInfo)
    doc.Blocks.Add(New Heading4(ei.Name))
    Dim text = String.Format("public {0} {1}", ei.EventHandlerType.Name, ei.Name)
    doc.Blocks.Add(New Normal(text))
End Sub
```

## C#

```
void DocumentMethod(C1Document doc, MethodInfo mi)
{
    if (mi.IsSpecialName)
        return;
    doc.Blocks.Add(new Heading4(mi.Name));
    var parms = new StringBuilder();
    foreach (var parm in mi.GetParameters())
    {
        if (parms.Length > 0)
```

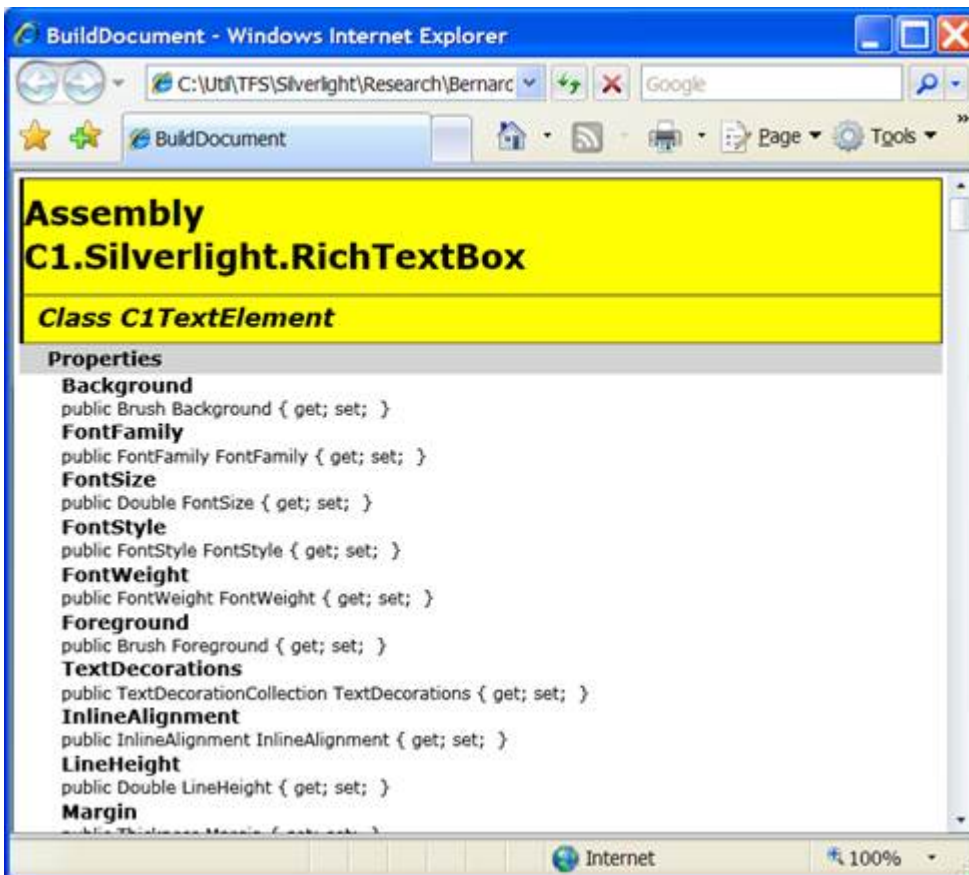


```

        parms.Append(", ");
        parms.AppendFormat("{0} {1}", parm.ParameterType.Name, parm.Name);
    }
    var text = string.Format("public {0} {1}({2})",
        mi.ReturnType.Name,
        mi.Name,
        parms.ToString());
    doc.Blocks.Add(new Normal(text));
}
void DocumentEvent(C1Document doc, EventInfo ei)
{
    doc.Blocks.Add(new Heading4(ei.Name));
    var text = string.Format("public {0} {1}",
        ei.EventHandlerType.Name,
        ei.Name);
    doc.Blocks.Add(new Normal(text));
}
}

```

ここでプロジェクトを実行すると、次のようなウィンドウが表示されます。



結果のドキュメントは、他と同様に **C1RichTextBox** で表示および編集できます。**C1RichTextBox** の **Html** プロパティを使用して、これを HTML にエクスポートしたり、クリップボードを使用して Microsoft Word、Excel などのアプリケーションにコピーすることもできます。

同じテクニックを使用して、データベースから取得したデータに基づくレポートを作成することもできます。書式設定されたテキストに加えて、**C1Document** オブジェクトモデルは以下の機能をサポートします。

- リスト

# RichTextBox for WPF

リストは、**C1List** のオブジェクトをドキュメントに追加して作成します。**C1List** オブジェクトには、**C1ListItem** のオブジェクトを含む **ListItems** プロパティがあります。このオブジェクトもブロックです。

- **ハイパーリンク**

ハイパーリンクは、**C1Hyperlink** のオブジェクトをドキュメントに追加して作成します。**C1Hyperlink** オブジェクトは、ラン（通常はテキストを含む **C1Run** 要素）のコレクションを含む **Inlines** プロパティと、このハイパーリンクがクリックされたときに実行するアクションを決定する **NavigateUrl** を持ちます。

- **イメージ**

イメージなどの **FrameworkElement** のオブジェクトは、ドキュメントに **C1BlockUIContainer** オブジェクトを追加することによって作成します。**C1BlockUIContainer** オブジェクトには **Child** プロパティがあり、これは任意の **FrameworkElement** オブジェクトに設定できます。すべてのオブジェクトを HTML にエクスポートできるわけではありません。イメージは、その処理方法が HTML フィルタで定義されている特別なケースです。

## 分割表示の実装

多くのエディタにはドキュメントの分割表示機能があり、ドキュメントの一部を表示しながら、別の部分を作業することができます。

これは、2つ以上の **C1RichTextBox** コントロールを同じ基底の **C1Document** に連結することによって簡単に実現できます。各コントロールは独立したビューとして動作するので、通常どおりにドキュメントをスクロール、選択、および編集できます。1つのビューへの変更は、他のすべてのビューに反映されます。

この動作を示すために、ページのコンストラクタに数行のコードを追加して、前の例を拡張します。

## VisualBasic

```
Public Sub New()  
    ' デフォルトの初期化  
    InitializeComponent()  
    ' C1RichTextBox を作成してページに追加します  
    _rtb = New C1RichTextBox()  
    LayoutRoot.Children.Add(_rtb)  
    ' ドキュメントを作成して C1RichTextBox に表示します  
    _rtb.Document = DocumentAssembly(GetType(C1RichTextBox).Assembly)  
    _rtb.IsReadOnly = True  
    ' イベントハンドラを関連付けます  
    AddHandler _rtb.ElementMouseLeftButtonDown, AddressOf _rtb_ElementMouseLeftButtonDown  
    ' ページに2番目の C1RichTextBox を追加します  
    LayoutRoot.RowDefinitions.Add(New RowDefinition())  
    LayoutRoot.RowDefinitions.Add(New RowDefinition())  
    Dim rtb2 = New C1RichTextBox()  
    rtb2.SetValue(Grid.RowProperty, 1)  
    LayoutRoot.Children.Add(rtb2)  
    ' 2番目の C1RichTextBox を同じドキュメントに連結します  
    rtb2.Document = _rtb.Document  
End Sub
```

## C#

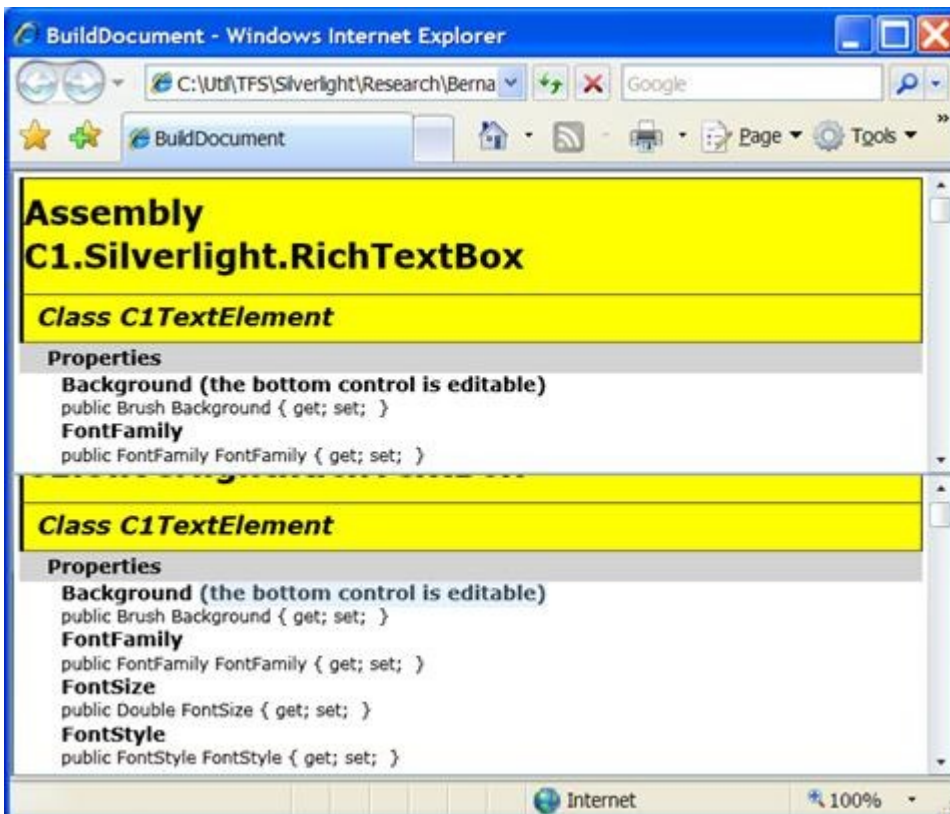
```

public MainPage()
{
    // デフォルトの初期化
    InitializeComponent();
    // C1RichTextBox を作成してページに追加します
    _rtb = new C1RichTextBox();
    LayoutRoot.Children.Add(_rtb);
    // ドキュメントを作成して C1RichTextBox に表示します
    _rtb.Document = DocumentAssembly(typeof(C1RichTextBox).Assembly);
    _rtb.IsReadOnly = true;
    // イベントハンドラを関連付けます
    _rtb.ElementMouseDownLeftButton += _rtb_ElementMouseDownLeftButton;
    // ページに2番目の C1RichTextBox を追加します
    LayoutRoot.RowDefinitions.Add(new RowDefinition());
    LayoutRoot.RowDefinitions.Add(new RowDefinition());
    var rtb2 = new C1RichTextBox();
    rtb2.SetValue(Grid.RowProperty, 1);
    LayoutRoot.Children.Add(rtb2);
    // 2番目の C1RichTextBox を同じドキュメントに連結します
    rtb2.Document = _rtb.Document;
}

```

この新しいコードは、新しい **C1RichTextBox** をページに追加し、次に元の **C1RichTextBox** によって表示されるドキュメントを **Document** プロパティに設定します。

プロジェクトを再度実行すると、次のようなウィンドウが表示されます。



# RichTextBox for WPF

下部のコントロールは編集可能です (`IsReadOnly` プロパティを **False** に設定していません)。そこに入力すると、両方のコントロールに変更が同時に表示されます。

このメカニズムは一般的です。さらに多くのビューを同じドキュメントに簡単に連結できます。さらに、基底のドキュメントに対する変更は、すべてのビューに即座に反映されます。

## C1Document クラスの使用

前述のように、**C1RichTextBox** がコントロールコンテンツの線形的でフラットなビューを提供する一方、**C1Document** クラスはドキュメント構造を公開します。

ドキュメントのオブジェクトを直接操作するメリットを示すために、ユーザーが [Ctrl] キーを押したときに、**Heading2** タイプのすべての段落のテキストを大文字にする機能を前の例に追加します。

**C1RichTextBox** が公開するオブジェクトモデルには、これを確実に実行できる機能はありません。書式設定に基づいてスパンの場所を特定する必要がありますが、それは非効率で信頼性も高くありません。ユーザーが **Heading2** で使用されている書式設定と同じ書式をプレーンテキストに設定していたらどうなるでしょう。

**C1Document** オブジェクトモデルを使用すると、このタスクを容易に実行できます。

## VisualBasic

```
Public Sub New()  
    ' デフォルトの初期化  
    InitializeComponent()  
    ' ここは変更しません...  
    ' 2番目の C1RichTextBox を同じドキュメントに連結します  
    rtb2.Document = _rtb.Document  
    AddHandler rtb2.KeyDown, AddressOf rtb2_KeyDown  
End Sub  
Private Sub rtb2_KeyDown(sender As Object, e As KeyEventArgs)  
    If e.Key = Key.Ctrl Then  
        For Each heading2 As var In _rtb.Document.Blocks.OfType(Of Heading2)()  
            Dim text = heading2.ContentRange.Text  
            heading2.ContentRange.Text = text.ToUpper()  
        Next  
    End If  
End Sub
```

## C#

```
public MainPage()  
{  
    // デフォルトの初期化  
    InitializeComponent();  
    // ここは変更しません...  
    // 2番目の C1RichTextBox を同じドキュメントに連結します  
    rtb2.Document = _rtb.Document;  
    rtb2.KeyDown += rtb2_KeyDown;  
}
```

```

void rtb2_KeyDown(object sender, KeyEventArgs e)
{
    if (e.Key == Key.Ctrl)
    {
        foreach (var heading2 in _rtb.Document.Blocks.OfType<Heading2>())
        {
            var text = heading2.ContentRange.Text;
            heading2.ContentRange.Text = text.ToUpper();
        }
    }
}

```

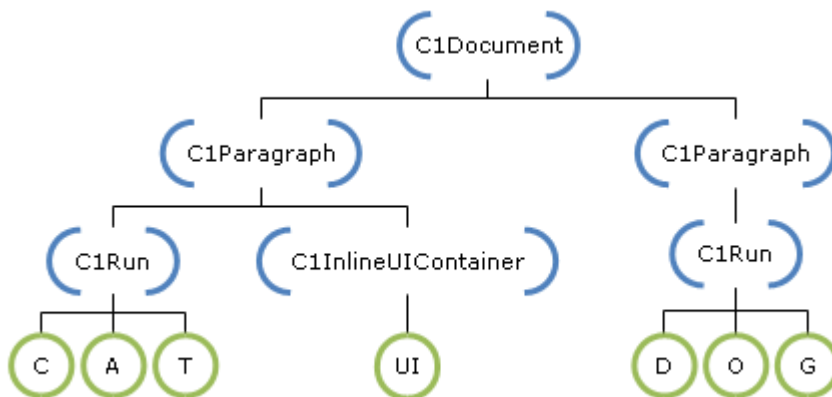
このコードはキーボード入力を監視します。ユーザーが [Ctrl] キーを押すと、ドキュメント内のすべての **Heading2** 要素が列挙され、そのコンテンツが大文字に変換されます。

**C1RichTextBox** オブジェクトモデルを使用して同じ作業を行うと、さらに多くのコードが必要になり、結果もそれほど信頼できません。

## C1TextPointer の理解

**C1TextPointer** クラスは、**C1Document** 内の位置を表します。その目的は、**C1Document** の走査と操作を容易にすることです。この機能は WPF の **TextPointer** クラスに似ていますが、オブジェクトモデルには多くの違いがあります。

**C1TextPointer** は、**C1TextElement** およびその内部のオフセットによって定義されます。ここでは、このドキュメントを例にして説明します。



上の青色の角かっこで囲まれたノードは **C1TextElement** で、**C1TextPointer** のオフセットは、その位置がどの子の間にあるかを示します。たとえば、上の **C1Document** をオフセット 0 でポイントする位置は最初の **C1Paragraph** の直前、オフセット 1 は 2 つの段落の間、オフセット 2 は 2 番目の段落の後を示します。

**C1TextPointer** が **C1Run** をポイントする場合は、そのテキスト内の各文字が **C1Run** の子と見なされ、オフセットはテキスト内の位置を示します。**C1InlineUIContainer** は子を 1 つだけ持つ（それが表示する **UIElement**）と見なされ、その子の前と後という 2 つの位置があります。

ドキュメントを一連のシンボルとして可視化する方法もあります。ここで、シンボルは要素タグまたは何らかのタイプのコンテンツになります。要素タグは、要素の開始または終了を示します。XML では、上のドキュメントは次のように記述されます。

XAML

# RichTextBox for WPF

```
<C1Document>
  <C1Paragraph>
    <C1Run>CAT</C1Run>
    <C1InlineUIContainer><UI/></C1InlineUIContainer>
  </C1Paragraph>
  <C1Paragraph>
    <C1Run>DOG</C1Run>
  </C1Paragraph>
</C1Document>
```

このようにドキュメントを表示する場合、**C1TextPointer** はタグやコンテンツの間の位置をポイントします。このビューは、**C1TextPointer** に明確な順序も提供します。実際、**C1TextPointer** は **IComparable** を実装し、便宜のために比較演算子もオーバーロードします。

**C1TextPointer** の後にあるシンボルは、**Symbol** プロパティを使用して取得できます。このプロパティは、**StartTag**、**EndTag**、**char**、**UIElement** のいずれかのタイプのオブジェクトを返します。

ドキュメント内の位置を反復処理する場合は、**GetPositionAtOffset** と **Enumerate** という2つのメソッドを使用できます。**GetPositionAtOffset** は低レベルのメソッドで、単に指定された整数オフセットの位置を返します。**Enumerate** は、位置を反復処理する場合にお勧めする方法です。このメソッドは、指定された方向にすべての位置に対して反復処理を行う **IEnumerable** を返します。たとえば、次のコードはドキュメントのすべての位置を返します。

```
document.ContentStart.Enumerate()
```

**ContentStart** は **C1TextElement** の最初の **C1TextPointer** を返します。最後の位置を返す **ContentEnd** プロパティもあります。

**Enumerate** の興味深い点は、必要に応じて列挙を返すことです。つまり、**IEnumerable** が反復処理される場合にのみ **C1TextPointer** オブジェクトが作成されます。これにより、フィルタ処理、検索、選択などのための LINQ 拡張メソッドを効率的に使用できます。たとえば、**C1TextPointer** の下に含まれる単語に対応する **C1TextRange** を取得するとします。次の手順を実行します。

## VisualBasic

```
Private Function ExpandToWord(pos As C1TextPointer) As C1TextRange
  ' 単語の先頭を探します
  Dim wordStart = If(pos.IsWordStart, pos,
pos.Enumerate(LogicalDirection.Backward).First(Function(p) p.IsWordStart))
  ' 単語の末尾を探します
  Dim wordEnd = If(pos.IsWordEnd, pos,
pos.Enumerate(LogicalDirection.Forward).First(Function(p) p.IsWordEnd))
  ' 単語の先頭から末尾までの新しい範囲を返します
  Return New C1TextRange(wordStart, wordEnd)
End Function
```

## C#

```
C1TextRange ExpandToWord(C1TextPointer pos)
```

```

{
    // 単語の先頭を探します
    var wordStart = pos.IsWordStart
        ? pos
        : pos.Enumerate(LogicalDirection.Backward).First(p => p.IsWordStart);
    // 単語の末尾を探します
    var wordEnd = pos.IsWordEnd
        ? pos
        : pos.Enumerate(LogicalDirection.Forward).First(p => p.IsWordEnd);
    // 単語の先頭から末尾までの新しい範囲を返します
    return new C1TextRange(wordStart, wordEnd);
}

```

**Enumerate** メソッドは、指定された方向で位置を検索して返しますが、現在の位置は含まれません。したがって、コードはパラメータの位置が単語の先頭かどうかを最初にチェックし、そうでない場合は逆方向に単語の先頭を検索します。単語の末尾についても同様に、パラメータの位置をチェックしてから順方向に検索します。パラメータの位置を含む単語を探しているため、順方向に移動して最初の単語の末尾を求め、逆方向に移動して最初の単語の先頭を求めます。**C1TextPointer** には、周囲のシンボルに基づいてその位置が単語の先頭または末尾かどうかを判別する **IsWordStart** プロパティと **IsWordEnd** プロパティが既に用意されています。ここでは、First LINQ 拡張メソッドを使用して、必要な述語を満たす最初の位置を探しています。最後に、2つの位置から **C1TextRange** を作成します。

LINQ 拡張メソッドは、位置を操作する際に大いに役立ちます。別の例として、次のようにするとドキュメント内の単語をカウントできます。

## VisualBasic

```
document.ContentStart.Enumerate().Count(Function(p) p.IsWordStart AndAlso TypeOf p.Symbol Is Char)
```

## C#

```
document.ContentStart.Enumerate().Count(p => p.IsWordStart && p.Symbol is char)
```

**IsWordStart** は、正確には単語の先頭ではない位置に対しても **True** を返すため、単語の先頭に続くシンボルが char かどうかをチェックする必要があることに注意してください。たとえば、**C1Run** の最初の位置が単語の先頭の場合、**C1Run** の開始タグの直前の位置は単語の先頭と見なされます。

もう1つの例として、**Find** メソッドを実装してみます。

## VisualBasic

```

Private Function FindWordFromPosition(position As C1TextPointer, word As String) As C1TextRange
    ' テキストの長さが word.Length に等しいすべての範囲を取得します
    Dim ranges = position.Enumerate().[Select](Function(pos)
        ' word.Length オフセットにある位置を取得します
        ' ただし、テキストフローを変更しないタグは無視します

```

# RichTextBox for WPF

```
Dim [end] = pos.GetPositionAtOffset(word.Length, C1TextRange.TextTagFilter)
Return New C1TextRange(pos, [end])
End Function)
' 単語が見つからない場合の戻り値は null です
Return ranges.FirstOrDefault(Function(range) range.Text = word)
End Function
```

## C#

```
C1TextRange FindWordFromPosition(C1TextPointer position, string word)
{
    // テキストの長さが word.Length に等しいすべての範囲を取得します
    var ranges = position.Enumerate().Select(pos =>
    {
        // word.Length オフセットにある位置を取得します
        // ただし、テキストフローを変更しないタグは無視します
        var end = pos.GetPositionAtOffset(word.Length, C1TextRange.TextTagFilter);
        return new C1TextRange(pos, end);
    });
    // 単語が見つからない場合の戻り値は null です
    return ranges.FirstOrDefault(range => range.Text == word);
}
```

指定された位置から単語を見つけるために、順方向にすべての位置を列挙し、テキストの長さが **word.Length** のすべての範囲を選択します。それぞれの位置に対して、**word.Length** の距離にある位置を探します。そのために、**GetPositionAtOffset** メソッドを使用します。このメソッドは、指定されたオフセットの位置を返しますが、すべてのインラインタグも有効な位置として返します。ここでは、単語が2つの **C1Run** 要素の間で分割されている場合を考慮するために、これを無視する必要があります。**C1TextRange.TextTagFilter** を使用するのはそのためです。これは、内部ロジックでドキュメントツリーをテキストに変換する際に使用されるフィルタメソッドと同じです。最後の手順として、テキストが検索対象の単語に一致する範囲を検索します。

最後の例として、最初に見つかった単語を置換します。

## VisualBasic

```
Dim wordRange = FindWordFromPosition(document.ContentStart, "cat")
If wordRange IsNot Nothing Then
    wordRange.Text = "dog"
End If
```

## C#

```
var wordRange = FindWordFromPosition(document.ContentStart, "cat");
if (wordRange != null)
{
    wordRange.Text = "dog";
}
```



この例では、まず単語を検索し、次に **Text** プロパティを割り当ててテキストを置換します。

## C1RichTextBoxToolbar の使い方

**C1RichTextBoxToolbar** コントロールは、**C1RichTextBox** コントロールを簡単に完全なテキストエディタに変えるだけの十分な機能を備えたリボン形式のツールバーです。**C1RichTextBoxToolbar** コントロールは、完全なカスタマイズを可能にする **C1Toolbar** コントロールに基づいています。

**C1RichTextBoxToolbar** コントロールを **C1RichTextBox** コントロールに接続するには、**RichTextBox** プロパティをこのツールバーにリンクするコントロールの名前に設定します。例については、「**C1RichTextBoxToolbar** を **C1RichTextBox** に接続する」を参照してください。

アプリケーションに **C1RichTextBoxToolbar** を追加すると、次のようになります。



ツールバー内のボタンは、Microsoft Word などのエディタにあるオプションとよく似ています。したがって、エンドユーザーにとっては使い慣れたツールです。[ホーム] と [テーブル] の2つのタブがあります。

### [ホーム] タブ

[ホーム] タブには、[編集]、[フォント]、[段落]、[挿入]、および [ツール] の5つの要素グループが定義されています。



### [テーブル] タブ

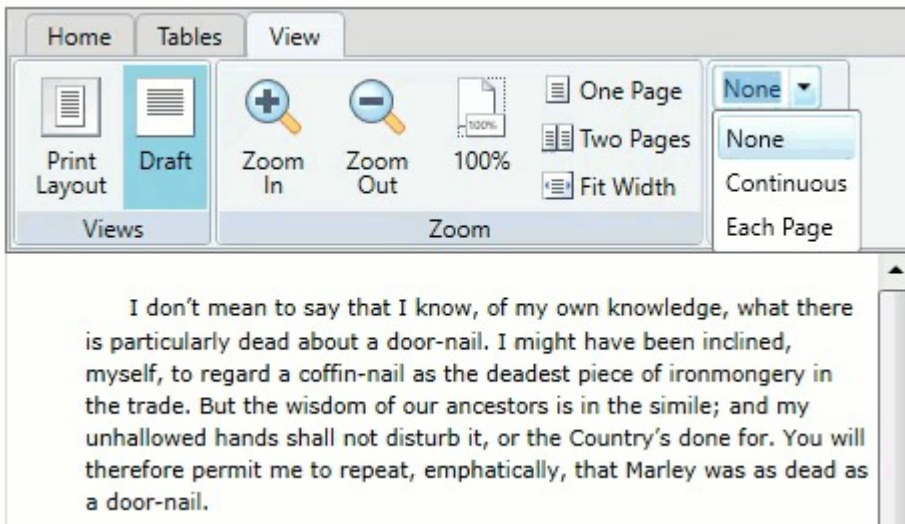
[テーブル] タブには、[テーブル]、[行と列]、[マージ]、[セル] の4つの要素グループが定義されています。



### View Tab

The **View** Tab which includes three defined group of elements; **Views**, **Zoom**, and **Line Number**.

- The Views group allows you to view the document in Print Layout and Draft mode.
- The Zoom group allows you to Zoom In and Zoom out the document for specific purposes. It also allows you to view the document pages in multiple modes like One Page, Two Pages, and Fit Width.
- The Line Number group allows you provide line numbers to the text in C1RichTextBox. You can select continuous numbering for all pages, separate line numbering sequence for each page, or no line numbering.



## [編集] グループ

[編集] グループは、コピー、貼り付け、元に戻すなどの標準的な編集ツールを提供します。次の画像のようになります。



[編集] グループには、次のオプションがあります。

- 貼り付け**： [貼り付け] ボタンをクリックすると、選択したテキストが **C1RichTextBox** に貼り付けられます。テキストを選択した状態で [Ctrl] + [V] キーを押しても、テキストを貼り付けることができます。
- 切り取り**： [切り取り] ボタンをクリックすると、選択したテキストが切り取られ、クリップボードに貼り付けられます。テキストを選択した状態で [Ctrl] + [X] キーを押しても、テキストを切り取ることができます。
- コピー**： [コピー] ボタンをクリックすると、選択したテキストをコピーして、**C1RichTextBox** または他のアプリケーションに貼り付けることができます。テキストを選択した状態で [Ctrl] + [C] キーを押しても、テキストをコピーできます。
- 元に戻す**： [元に戻す] ボタンをクリックすると、前に **C1RichTextBox** のコンテンツに加えられた変更が元に戻ります。[元に戻す] ボタンは、変更が行われるまではアクティブになりません。[Ctrl] + [Z] キーを押しても、変更を元に戻すことができます。
- やり直し**： [元に戻す] ボタンをクリックした後に [やり直し] ボタンをクリックすると、前に元に戻した変更をやり直します。[やり直し] ボタンは、[元に戻す] ボタンがクリックされるまでアクティブになりません。[Ctrl] + [Y] キーを押しても、変更をやり直すことができます。
- [書式のコピー/貼り付け]**： [書式のコピー/貼り付け] ボタンをクリックすると、選択したテキストの書式設定をコピーして、他のテキストに適用できます。

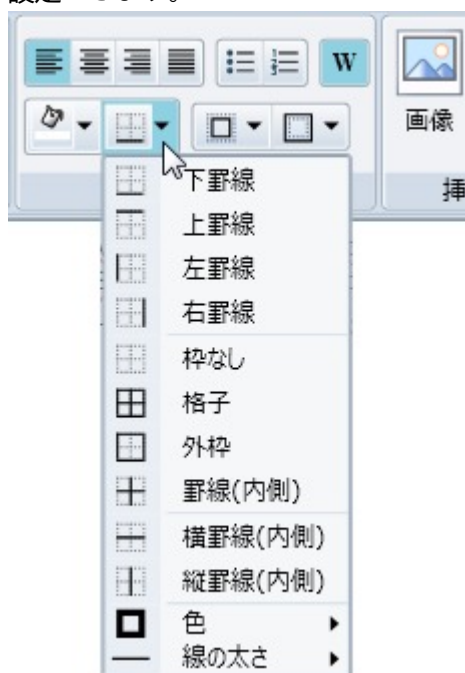
## [段落] グループ

[段落] グループには、**C1RichTextBox** コントロールで使用される段落設定をカスタマイズするためのオプションがあります。[段落] グループの外観は次の画像のようになります。



[段落] グループには、次のオプションがあります。

- **テキスト左揃え**：[テキスト左揃え] オプションは、選択したテキストを左揃えで配置します。これはデフォルトの配置オプションです。
- **テキスト中央揃え**：[テキスト中央揃え] オプションは、選択したテキストを RichTextBox コントロールの中央に配置します。
- **テキスト右揃え**：[テキスト右揃え] オプションは、選択したテキストを右揃えで配置します。
- **均等揃え**：[均等揃え] オプションは、選択されたテキストに空間を入れて均等に配置します。
- **箇条書き**：[箇条書き] オプションは、選択されたテキストに箇条書きを追加または削除するボタンです。
- **番号付き**：[番号付き] オプションは、選択されたテキストに番号を追加または削除するボタンです。
- **テキスト折り返し**：[テキスト折り返し] オプションは、テキストの行を RichTextBox 内に収まるように折り返すかどうかを判断します。デフォルトでは、テキストは折り返されません。
- **境界線**：[境界線] ドロップダウンボックスを使用すると、境界線の色、太さ、境界線を表示する場所を設定できます。



- **段落の色:** **[段落の色]** オプションは、段落全体の背景の色を変更するための標準的な色をリストするドロップダウンカラーピッカーです。デフォルトでは、このオプションは **[白]** に設定されます。



- **マージン:** **[マージン]** ドロップダウンボックスを使用すると、テキストの周囲の**左、右、上**、および下のマージンの大きさを設定できます。デフォルトでは、**左、右、上**マージンの値は**0**、**下**マージンの値**10**に設定されます。
- **パディング:** **[パディング]** ドロップダウンボックスを使用すると、選択されたテキストの周囲の**左、右、上**、および下のパディングの大きさを設定できます。デフォルトでは、これらの値は**0**に設定され、パディングはありません。

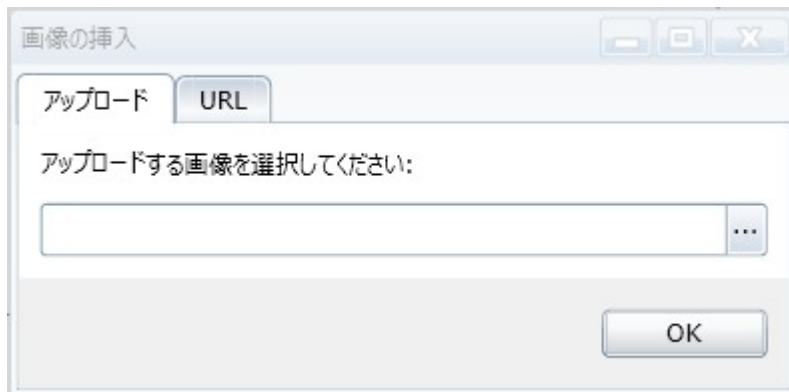
## [挿入] グループ

**[挿入]** グループを使用すると、**C1RichTextBox** コントロール内のコンテンツを挿入できます。**[挿入]** グループの外観は次の画像のようになります。



**[挿入]** グループには、次のオプションがあります。

- **画像の挿入:** **[画像の挿入]** ボタンをクリックすると、**[画像の挿入]** ダイアログボックスが表示され、挿入する画像を参照して選択したり、画像の URL を入力することができます。



- **記号の挿入:** **[記号の挿入]** オプションをクリックすると、RichTextBox に挿入する記号を選択するためのドロップダウンダイアログボックスが表示されます。



- **ハイパーリンクの挿入:** [ハイパーリンクの挿入] ボタンをクリックすると、[ハイパーリンクの挿入] ダイアログボックスが表示され、挿入するハイパーリンクテキストと URL を入力できます。



- **ハイパーリンクの削除:** [ハイパーリンクの削除] ボタンをクリックすると、選択したテキスト内のハイパーリンクが削除されます。RichTextBox 内のハイパーリンクを強調表示し、[ハイパーリンクの削除] ボタンをクリックしてリンクを削除します。

## [ツール] グループ

[ツール] グループを使用すると、**C1RichTextBox** コントロール内のコンテンツを操作および編集できます。

[ツール] グループの外観は次の画像のようになります。



[ツール] グループには、次のオプションがあります。

- **スペルチェック:** [スペルチェック] ボタンをクリックすると、[スペル] ダイアログボックスが表示されます。このボタンをアクティブ化するには、スペルチェックを設定する必要があります。詳細については、「[スペルチェック](#)」を参照してください。[スペル] ダイアログボックスでは、提案されたスペルを確認して、エラーを無視したり、同じ単語のエラーをすべて無視したり、現在のスペルを提案されているスペルに変更したり、現在のスペルの出現箇所をすべて変更したり、現在のスペルをスペルチェック辞書に追加したり、提案されたスペルに基づいてその他のスペルを表示したり、ダイアログボックスを閉じてスペルチェック操作をキャンセルすることができます。



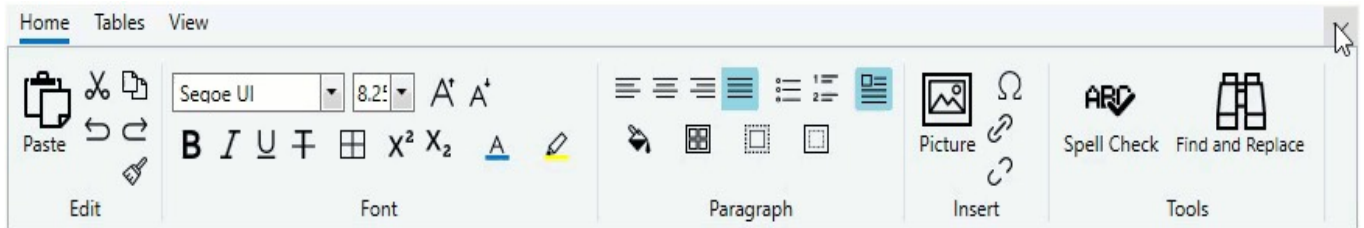
- 検索と置換:** **[検索と置換]** オプションをクリックすると、ドロップダウンボックスが表示され、RichTextBox のコンテンツを検索したり、検索して置換するためのテキストを入力できます。1 つずつ**置換**したり、**すべてを置換**したり、**次の一致を検索**することができます。



## C1SimplifiedRichTextBoxToolbarの使い方

The new **C1SimplifiedRichTextBoxToolbar** offers WPF Simplified Ribbon which provides you an option to work with a single line collapsed view or an expanded view containing three line appearances. This allows you to allocate more screen space to the working area of your application. In addition, it automatically combines items in a group into a dropdown when it doesn't have enough space to display all group items across the window width.

The control is represented by the **C1SimplifiedRichTextBoxToolbar** class. It has improvised look and feel in comparison to **C1RichTextBoxToolbar** as shown in GIF below.



### Quick Start

In this step, you create a new WPF application and add the **RichTextBox** and **SimplifiedRichTextBoxToolbar** controls in XAML. After completing this step, you have a mostly functional text-rich editor.

### In Design View

To add the RichTextBox control to your WPF application in Design view, perform the following steps

1. Create a new WPF application in Visual Studio.
2. Navigate to the Toolbox and locate the **C1RichTextBox** and **C1SimplifiedRichTextBoxToolbar** controls.
3. Double-click the C1RichTextBox and C1SimplifiedRichTextBoxToolbar icons to add the controls to the MainWindow.
4. In the XAML view, add the relevant namespaces to the page by editing the MainWindow tags.

```
XAML copyCode  
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"  
x:Class="RTBQuickStart.MainWindow"  
Title="MainWindow" Height="350" Width="525">
```

5. Place the cursor between the <Grid> and </Grid> tags, click once, and add the following markup within the <Grid> tags to add a **StackPanel** panel.

```
XAML copyCode  
<StackPanel HorizontalAlignment="Left" Margin="0,10,0,0" x:Name="SP"  
VerticalAlignment="Top" Height="418" Width="645" Grid.ColumnSpan="2"  
Grid.Column="1"/>
```

6. Click between the StackPanel tags and add the following markup to edit the appearance and positioning of RichTextBox and SimplifiedRichTextBoxToolbar controls in the XAML view.

```
XAML  
<c1:C1RichTextBox Name="c1RichTextBox1" Margin="0,127,0,10"/>  
<c1:C1SimplifiedRichTextBoxToolbar RichTextBox="{Binding ElementName=  
c1RichTextBox1}" Name="C1RTBTB" Margin="3,0,-3,197" />
```

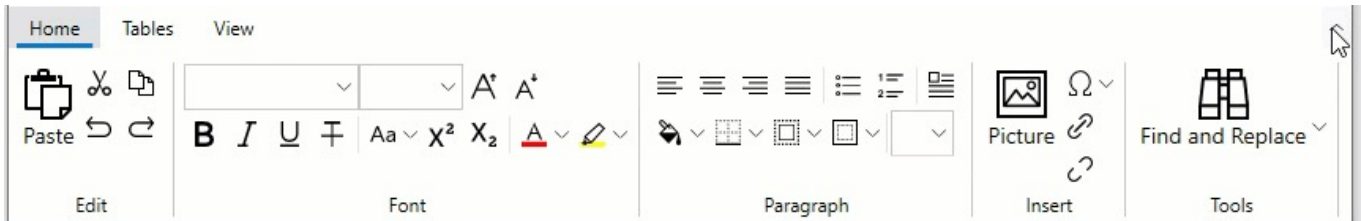


You can enter text in the RichTextBox control and edit, format, and position the text using latest options in the SimplifiedRichTextBoxToolbar. To set up spell-checking and customize the application further, refer to [RichTextBox for WPF Quick Start](#).

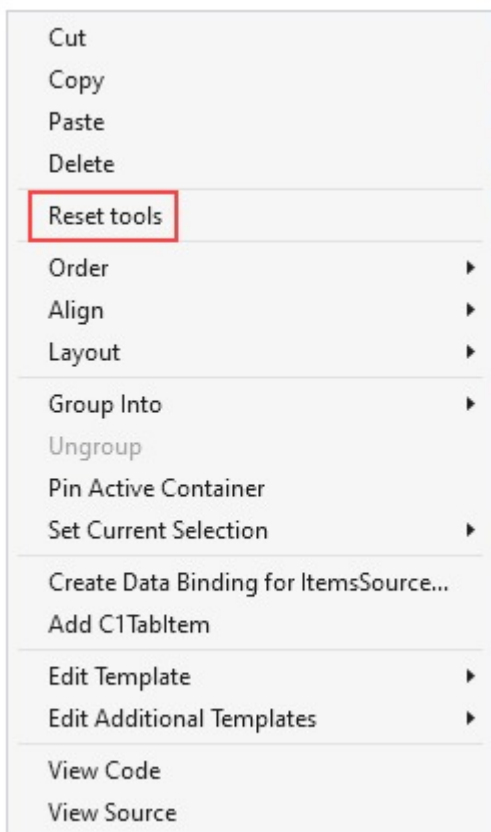
## RichTextBox Ribbon の使い方

**Note:** The **RichTextBoxToolStrip** is available only for .NET 5 framework. For .NET 4.5.2 framework, RichTextBox provides the **C1SimplifiedRichTextBox** and **C1RichTextBoxToolbar**.

RichTextBoxRibbon is a ribbon control that has been designed to work with the commands of RichTextBox. Once you add the **C1.WPF.RichTextBox.Ribbon** nuget package, C1RichTextBoxRibbon gets automatically added in the Visual Studio Toolbox automatically. This control is represented by the **C1RichTextBoxRibbon** class.



The RichTextBox Ribbon UI has predefined elements such as Ribbon tab, Ribbon Group, and other Ribbon Items. The Ribbon group and tab controls are represented by **RibbonTabItem** and **RibbonGroup** classes. Moreover, the tabs, groups and all ribbon items in the Ribbon UI is easily editable in the designer. Also, you can use the Reset tools option in the context menu of RichTextBox.Ribbon to get the original layout of the tools in the Ribbon.



When a RichTextBoxRibbon control is dragged onto the form, the following XAML lines are generated:

### XAML

```
<c1:C1RichTextBoxRibbon VerticalAlignment="Top" SelectedIndex="2">
  <c1:RibbonTabItem Header="Home">
    <c1:RibbonGroup Header="Edit">
      <c1:C1PasteTool/>
      <c1:C1CutTool/>
      <c1:C1UndoTool/>
```

```

    <c1:C1ToolSeparator/>
    <c1:C1CopyTool/>
    <c1:C1RedoTool/>
</c1:RibbonGroup>
<c1:RibbonGroup Header="Font">
    <c1:C1ToolStrip>
        <c1:C1FontFamilyTool/>
        <c1:C1FontSizeTool/>
        <c1:C1IncreaseFontSizeTool/>
        <c1:C1DecreaseFontSizeTool/>
    </c1:C1ToolStrip>
    <c1:C1ToolStrip>
        <c1:C1BoldTool/>
        <c1:C1ItalicTool/>
        <c1:C1UnderlineTool/>
        <c1:C1StrikethroughTool/>
        <c1:C1ToolSeparator/>
        <c1:C1ChangeCaseTool/>
        <c1:C1SuperscriptTool/>
        <c1:C1SubscriptTool/>
        <c1:C1ToolSeparator/>
        <c1:C1FontColorTool/>
        <c1:C1TextHighlightTool/>
    </c1:C1ToolStrip>
</c1:RibbonGroup>
<c1:RibbonGroup Header="Paragraph">
    <c1:C1ToolStrip>
        <c1:C1LeftAlignTool/>
        <c1:C1CenterAlignTool/>
        <c1:C1RightAlignTool/>
        <c1:C1JustifyTool/>
        <c1:C1ToolSeparator/>
        <c1:C1BulletsTool/>
        <c1:C1NumberingTool/>
        <c1:C1ToolSeparator/>
        <c1:C1TextWrappingTool/>
    </c1:C1ToolStrip>
    <c1:C1ToolStrip>
        <c1:C1ParagraphColorTool/>
        <c1:C1BorderMenuTool/>
        <c1:C1MarginTool/>
        <c1:C1PaddingTool/>
        <c1:C1ToolSeparator/>
        <c1:C1LineSpacingTool/>
    </c1:C1ToolStrip>
</c1:RibbonGroup>
<c1:RibbonGroup Header="Insert">
    <c1:C1InsertImageTool/>
    <c1:C1InsertSymbolTool/>
    <c1:C1InsertHyperlinkTool/>
    <c1:C1RemoveHyperlinkTool/>
</c1:RibbonGroup>

```


```
<c1:RibbonGroup Header="Tools">
  <c1:RibbonGroup.GroupSizeDefinitions>
    <c1:RibbonGroupSizeDefinition>
      <c1:RibbonToolSizeDefinition Size="Large"/>
    </c1:RibbonGroupSizeDefinition>
  </c1:RibbonGroup.GroupSizeDefinitions>
  <c1:C1FindAndReplaceTool/>
</c1:RibbonGroup>
</c1:RibbonTabItem>
<c1:RibbonTabItem Header="Tables">
  <c1:RibbonGroup Header="Table">
    <c1:RibbonGroup.GroupSizeDefinitions>
      <c1:RibbonGroupSizeDefinition>
        <c1:RibbonToolSizeDefinition Size="Large"/>
      </c1:RibbonGroupSizeDefinition>
    </c1:RibbonGroup.GroupSizeDefinitions>
    <c1:C1InsertTableTool/>
    <c1:C1SelectTableMenuTool/>
    <c1:C1ShowGridlinesTool/>
    <c1:C1TableWidthTool/>
  </c1:RibbonGroup>
  <c1:RibbonGroup Header="Rows And Columns">
    <c1:RibbonGroup.GroupSizeDefinitions>
      <c1:RibbonGroupSizeDefinition>
        <c1:RibbonToolSizeDefinition Size="Large"/>
      </c1:RibbonGroupSizeDefinition>
    </c1:RibbonGroup.GroupSizeDefinitions>
    <c1:C1DeleteTableMenuTool/>
    <c1:C1InsertColumnsLeftTool/>
    <c1:C1InsertColumnsRightTool/>
    <c1:C1InsertRowsAboveTool/>
    <c1:C1InsertRowsBelowTool/>
  </c1:RibbonGroup>
  <c1:RibbonGroup Header="Merge">
    <c1:RibbonGroup.GroupSizeDefinitions>
      <c1:RibbonGroupSizeDefinition>
        <c1:RibbonToolSizeDefinition Size="Large"/>
      </c1:RibbonGroupSizeDefinition>
    </c1:RibbonGroup.GroupSizeDefinitions>
    <c1:C1MergeCellsTool/>
    <c1:C1UnmergeCellTool/>
  </c1:RibbonGroup>
  <c1:RibbonGroup Header="Cell">
    <c1:RibbonGroup.GroupSizeDefinitions>
      <c1:RibbonGroupSizeDefinition>
        <c1:RibbonToolSizeDefinition Size="Small"/>
      </c1:RibbonGroupSizeDefinition>
    </c1:RibbonGroup.GroupSizeDefinitions>
    <c1:C1ColumnWidthTool/>
    <c1:C1RowHeightTool/>
    <c1:C1ToolSeparator/>
    <c1:C1TopAlignTool/>
    <c1:C1VerticalCenterAlignTool/>
  </c1:RibbonGroup>
</c1:RibbonTabItem>
</c1:RibbonTab>
</c1:Ribbon>
```

```

        <c1:C1BottomAlignTool/>
    </c1:RibbonGroup>
</c1:RibbonTabItem>
<c1:RibbonTabItem Header="View">
    <c1:RibbonGroup Header="Document">
        <c1:RibbonGroup.GroupSizeDefinitions>
            <c1:RibbonGroupSizeDefinition>
                <c1:RibbonToolSizeDefinition Size="Large"/>
            </c1:RibbonGroupSizeDefinition>
        </c1:RibbonGroup.GroupSizeDefinitions>
        <c1:C1PrintViewTool/>
        <c1:C1DraftViewTool/>
    </c1:RibbonGroup>
    <c1:RibbonGroup Header="Zoom">
        <c1:RibbonGroup.GroupSizeDefinitions>
            <c1:RibbonGroupSizeDefinition>
                <c1:RibbonToolSizeDefinition Size="Large"/>
                <c1:RibbonToolSizeDefinition Size="Large"/>
                <c1:RibbonToolSizeDefinition Size="Large"/>
                <c1:RibbonToolSizeDefinition Size="Small"/>
                <c1:RibbonToolSizeDefinition Size="Small"/>
                <c1:RibbonToolSizeDefinition Size="Small"/>
            </c1:RibbonGroupSizeDefinition>
        </c1:RibbonGroup.GroupSizeDefinitions>
        <c1:C1ZoomInTool/>
        <c1:C1ZoomOutTool/>
        <c1:C1Zoom100PercentTool/>
        <c1:C1OnePageTool/>
        <c1:C1TwoPagesTool/>
        <c1:C1FitWidthTool/>
    </c1:RibbonGroup>
    <c1:RibbonGroup Header="Line Number">
        <c1:RibbonGroup.GroupSizeDefinitions>
            <c1:RibbonGroupSizeDefinition>
                <c1:RibbonToolSizeDefinition Size="Small"/>
            </c1:RibbonGroupSizeDefinition>
        </c1:RibbonGroup.GroupSizeDefinitions>
        <c1:C1LineNumberTool/>
    </c1:RibbonGroup>
</c1:RibbonTabItem>
</c1:C1RichTextBoxRibbon>

```

## RichTextBox ToolStrip の使い方

 **Note:** The **RichTextBoxToolStrip** is available only for .NET 5 framework. For .NET 4.5.2 framework, RichTextBox provides the [C1SimplifiedRichTextBox](#) and [C1RichTextBoxToolBar](#).

The RichTextBoxToolStrip control lays out a series of basic tools, which will be automatically bound to the assigned RichTextBox. Once you add the **C1.WPF.RichTextBox.Ribbon** nuget package, **C1RichTextBoxToolStrip** gets automatically added in the Visual Studio Toolbox automatically. This control is represented by the **C1RichTextBoxToolStrip** class.



When you drop a RichTextBox ToolStrip control onto the window, the following XAML code is generated:

### XAML

```
<c1:C1RichTextBoxToolStrip VerticalAlignment="Top">
    <c1:C1UndoTool/>
    <c1:C1RedoTool/>
    <c1:C1FontFamilyTool/>
    <c1:C1FontSizeTool/>
    <c1:C1PasteTool/>
    <c1:C1CutTool/>
    <c1:C1CopyTool/>
    <c1:C1ToolSeparator/>
    <c1:C1BoldTool/>
    <c1:C1ItalicTool/>
    <c1:C1UnderlineTool/>
    <c1:C1StrikethroughTool/>
    <c1:C1ToolSeparator/>
    <c1:C1NumberingTool/>
    <c1:C1BulletsTool/>
    <c1:C1ToolSeparator/>
    <c1:C1LeftAlignTool/>
    <c1:C1CenterAlignTool/>
    <c1:C1RightAlignTool/>
    <c1:C1JustifyTool/>
</c1:C1RichTextBoxToolStrip>
```

## RichTextBox でサポートされる要素

以下のセクションでは、**RichTextBox for WPF** でサポートされている HTML 4 要素、HTML 属性、CSS2 プロパティ、および CSS セレクタについて詳細に説明します。

### HTML 要素

**RichTextBox for WPF** は、多数の HTML 4 要素をサポートしています。次の表は、HTML 要素の名前と、それが **RichTextBox for WPF** でサポートされるかどうかを示します。

名前	サポート
A	○
ABBR	○
ACRONYM	○
ADDRESS	○
APPLET	×
AREA	×
B	○
BASE	×
BASEFONT	○
BDO	×
BIG	○
BLOCKQUOTE	○
BODY	○
BR	○
BUTTON	×
CAPTION	×
CENTER	○
CITE	○
CODE	○
COL	○
COLGROUP	○
DD	○
Del	○
DFN	○
DIR	○
DIV	○

# RichTextBox for WPF

DL	○
DT	○
EM	○
FIELDSET	×
FONT	○
FORM	×
FRAME	×
FRAMESET	×
H1	○
H2	○
H3	○
H4	○
H5	○
H6	○
HEAD	○
HR	○
HTML	○
I	○
IFRAME	×
IMG	○
INPUT	×
INS	○
ISINDEX	×
KBD	○
LABEL	○
LEGEND	×
LI	○
LINK	×
MAP	×
MENU	○
META	×
NOFRAMES	×
NOSCRIPT	×
OBJECT	×
OL	○



OPTGROUP	×
OPTION	×
P	○
PARAM	×
Pre	○
Q	×
S	○
SAMP	○
SCRIPT	×
SELECT	×
SMALL	○
SPAN	○
STRIKE	○
STRONG	○
STYLE	○
SUB	○
SUP	○
	○
TABLE	○
TBODY	○
TD	○
TEXTAREA	×
TFOOT	○
TH	○
THEAD	○
TITLE	×
TR	○
TT	○
U	○
UL	○
VAR	○

## HTML の属性

**RichTextBox for WPF** は、多数の HTML 4 属性をサポートしています。次の表は、HTML 属性の名前および要素と、それが **RichTextBox for WPF** でサポートされるかどうかを示します。

# RichTextBox for WPF

名前	要素	サポート
abbr	TD、TH	×
accept-charset	FORM	×
accept	FORM、INPUT	×
accesskey	A、AREA、BUTTON、INPUT、LABEL、LEGEND、TEXTAREA	×
action	FORM	×
align	CAPTION	×
align	APPLET、IFRAME、IMG、INPUT、OBJECT	×
align	LEGEND	×
align	TABLE	×
align	HR	×
align	DIV、H1、H2、H3、H4、H5、H6、P	○
align	COL、COLGROUP、TBODY、TD、TFOOT、TH、THEAD、TR	×
alink	BODY	×
alt	APPLET	×
alt	AREA、IMG	×
alt	INPUT	×
archive	APPLET	×
archive	OBJECT	×
axis	TD、TH	×
background	BODY	×
bgcolor	TABLE	○
bgcolor	TR	○
bgcolor	TD、TH	○
bgcolor	BODY	○
border	TABLE	○
border	IMG、OBJECT	○
cellpadding	TABLE	×
cellspacing	TABLE	○
char	COL、COLGROUP、TBODY、TD、TFOOT、TH、THEAD、TR	×
charoff	COL、COLGROUP、TBODY、TD、TFOOT、TH、THEAD、TR	×
charset	A、LINK、SCRIPT	×
checked	INPUT	×
cite	BLOCKQUOTE、Q	×
cite	DEL、INS	×

class	BASE、BASEFONT、HEAD、HTML、META、PARAM、SCRIPT、STYLE、TITLE 以外のすべての要素	○
classid	OBJECT	×
clear	BR	×
code	APPLET	×
codebase	OBJECT	×
codebase	APPLET	×
codetype	OBJECT	×
color	BASEFONT、FONT	×
cols	FRAMESET	×
cols	TEXTAREA	×
colspan	TD、TH	○
compact	DIR、DL、MENU、OL、UL	×
content	META	×
coords	AREA	×
coords	A	×
data	OBJECT	×
datetime	DEL、INS	×
declare	OBJECT	×
defer	SCRIPT	×
dir	APPLETBASE、BASEFONT、BDO、BR、FRAME、FRAMESET、IFRAME、 PARAM、SCRIPT 以外のすべての要素	×
dir	BDO	×
disabled	BUTTON、INPUT、OPTGROUP、OPTION、SELECT、TEXTAREA	×
enctype	FORM	×
face	BASEFONT、FONT	○
for	LABEL	×
frame	TABLE	○
frameborder	FRAME、IFRAME	×
headers	TD、TH	×
height	IFRAME	×
height	TD、TH	×
height	IMG、OBJECT	○
height	APPLET	×
href	A、AREA、LINK	○
href hspace	BASE	×

# RichTextBox for WPF

APPLET、 IMG、OBJECT ○		
hreflang	A、LINK	×
http-equiv	META	×
id	BASE、HEAD、HTML、META、SCRIPT、STYLE、TITLE 以外のすべての要素	○
ismap	IMG、INPUT	×
label	OPTION	×
label	OPTGROUP	×
lang	APPLETBASE、BASEFONT、BR、FRAME、FRAMESET、IFRAME、PARAM、 SCRIPT 以外のすべての要素	×
language	SCRIPT	×
link	BODY	×
longdesc	IMG	×
longdesc	FRAME、IFRAME	×
marginheight	FRAME、IFRAME	×
marginwidth	FRAME、IFRAME	×
maxlength	INPUT	×
media	STYLE	×
media	LINK	×
method	FORM	×
multiple	SELECT	×
name	BUTTON、TEXTAREA	×
name	APPLET	×
name	SELECT	×
name	FORM	×
name	FRAME、IFRAME	×
name	IMG	×
name	A	×
name	INPUT、OBJECT	×
name	MAP	×
name	PARAM	×
name	META	×
nohref	AREA	×
noresize	FRAME	×
noshade	HR	×

nowrap	TD、TH	×
object	APPLET	×
onblur	A、AREA、BUTTON、INPUT、LABEL、SELECT、TEXTAREA	×
onchange	INPUT、SELECT、TEXTAREA	×
onclick	APPLET、BASE、BASEFONT、BDO、BR、FONT、FRAME、FRAMESET、HEAD、HTML、IFRAME、ISINDEX、META、PARAM、SCRIPT、STYLE、TITLE 以外のすべての要素	×
ondblclick	APPLET、BASE、BASEFONT、BDO、BR、FONT、FRAME、FRAMESET、HEAD、HTML、IFRAME、ISINDEX、META、PARAM、SCRIPT、STYLE、TITLE 以外のすべての要素	×
onfocus	A、AREA、BUTTON、INPUT、LABEL、SELECT、TEXTAREA	×
onkeydown	APPLET、BASE、BASEFONT、BDO、BR、FONT、FRAME、FRAMESET、HEAD、HTML、IFRAME、ISINDEX、META、PARAM、SCRIPT、STYLE、TITLE 以外のすべての要素	×
onkeypress	APPLET、BASE、BASEFONT、BDO、BR、FONT、FRAME、FRAMESET、HEAD、HTML、IFRAME、ISINDEX、META、PARAM、SCRIPT、STYLE、TITLE 以外のすべての要素	×
onkeyup	APPLET、BASE、BASEFONT、BDO、BR、FONT、FRAME、FRAMESET、HEAD、HTML、IFRAME、ISINDEX、META、PARAM、SCRIPT、STYLE、TITLE 以外のすべての要素	×
onload	FRAMESET	×
onload	BODY	×
onmousedown	APPLET、BASE、BASEFONT、BDO、BR、FONT、FRAME、FRAMESET、HEAD、HTML、IFRAME、ISINDEX、META、PARAM、SCRIPT、STYLE、TITLE 以外のすべての要素	×
onmousemove	APPLET、BASE、BASEFONT、BDO、BR、FONT、FRAME、FRAMESET、HEAD、HTML、IFRAME、ISINDEX、META、PARAM、SCRIPT、STYLE、TITLE 以外のすべての要素	×
onmouseout	APPLET、BASE、BASEFONT、BDO、BR、FONT、FRAME、FRAMESET、HEAD、HTML、IFRAME、ISINDEX、META、PARAM、SCRIPT、STYLE、TITLE 以外のすべての要素	×
onmouseover	APPLET、BASE、BASEFONT、BDO、BR、FONT、FRAME、FRAMESET、HEAD、HTML、IFRAME、ISINDEX、META、PARAM、SCRIPT、STYLE、TITLE 以外のすべての要素	×
onmouseup	APPLET、BASE、BASEFONT、BDO、BR、FONT、FRAME、FRAMESET、HEAD、HTML、IFRAME、ISINDEX、META、PARAM、SCRIPT、STYLE、TITLE 以外のすべての要素	×
onreset	FORM	×
onselect	INPUT、TEXTAREA	×
onsubmit	FORM	×
onunload	FRAMESET	×
onunload	BODY	×

# RichTextBox for WPF

profile	HEAD	×
prompt	ISINDEX	×
readonly	TEXTAREA	×
readonly	INPUT	×
rel	A、LINK	×
rev	A、LINK	×
rows	FRAMESET	×
rows	TEXTAREA	×
rowspan	TD、TH	○
rules	TABLE	○
scheme	META	×
scope	TD、TH	×
scrolling	FRAME、IFRAME	×
selected	OPTION	×
shape	AREA	×
shape	A	×
size	HR	×
size	FONT	×
size	INPUT	×
size	BASEFONT	×
size	SELECT	×
span	COL	×
span	COLGROUP	×
src	SCRIPT	×
src	INPUT	×
src	FRAME、IFRAME	×
src	IMG	×
standby	OBJECT	×
start	OL	○
style	BASE、BASEFONT、HEAD、HTML、META、PARAM、SCRIPT、STYLE、TITLE 以外のすべての要素	○
summary	TABLE	×
tabindex	A、AREA、BUTTON、INPUT、OBJECT、SELECT、TEXTAREA	×
target	A、AREA、BASE、FORM、LINK	×
text	BODY	×

title	BASE、BASEFONT、HEAD、HTML、META、PARAM、SCRIPT、TITLE 以外のすべての要素	○
type	A、LINK	×
type	OBJECT	×
type	PARAM	×
type	SCRIPT	×
type	STYLE	×
type	INPUT	×
type	LI	×
type	OL	×
type	UL	×
type	BUTTON	×
usemap	IMG、INPUT、OBJECT	×
valign	COL、COLGROUP、TBODY、TD、TFOOT、TH、THEAD、TR	○
value	INPUT	×
value	OPTION	×
value	PARAM	×
value	BUTTON	×
value	LI	×
valuetype	PARAM	×
version	HTML	×
vlink	BODY	×
vspace	APPLET、IMG、OBJECT	○
width	HR	×
width	IFRAME	×
width	IMG、OBJECT	○
width	TABLE	×
width	TD、TH	×
width	APPLET	×
width	COL	○
width	COLGROUP	×
width	PRE	×

## CSS2 プロパティ

RichTextBox for WPF は、多数の CSS2 プロパティをサポートしています。次の表は、CSS2 プロパティの名前お

# RichTextBox for WPF

よびメディアグループと、それが **RichTextBox for WPF** でサポートされるかどうかを示します。

名前	メディアグループ	サポート	コメント
azimuth	aural	×	
background-attachment	visual	×	
background-color	visual	○	
background-image	visual	○	画像は繰り返されません。
background-position	visual	×	
background-repeat	visual	×	
background	visual	○	色と画像のみをサポートします。
border-collapse	visual	○	
border-color	visual	○	
border-spacing	visual	○	
border-style	visual	○	none、hidden、solid をサポートします。その他の値は solid として扱われます。
border-top border-right border-bottom border-left	visual	○	
border-top-color border-right-color border-bottom-color border-left-color	visual	○	
border-top-style border-right-style border-bottom-style border-left-style	visual	○	
border-top-width border-right-width border-bottom-width border-left-	visual	○	



width			
border-width	visual	○	
border	visual	○	
bottom	visual	×	
caption-side	visual	×	
clear	visual	×	
clip	visual	×	
color	visual	○	
content	all	×	
counter-increment	all	×	
counter-reset	all	×	
cue-after	aural	×	
cue-before	aural	×	
cue	aural	×	
cursor	visual、 interactive	○	crosshair、move、progress、help、<uri> 以外のすべての値。
direction	visual	×	
display	all	○	run-in、inline-block、inline-table、table-caption を除くすべての値。
elevation	aural	×	
empty-cells	visual	×	
float	visual	×	
font-family	visual	○	
font-size	visual	○	
font-style	visual	○	
font-variant	visual	×	
font-weight	visual	○	
font	visual	○	
height	visual	○	img 要素内のみ。
left	visual	×	
letter-spacing	visual	×	
line-height	visual	○	
list-style-image	visual	○	
list-style-position	visual	×	

# RichTextBox for WPF

list-style-type	visual	○	georgian、armenian、lower-greek 以外のすべての値。
list-style	visual	○	
margin-right margin-left	visual	○	
margin-top margin-bottom	visual	○	
margin	visual	○	
max-height	visual	×	
max-width	visual	×	
min-height	visual	×	
min-width	visual	×	
orphans	visual、 paged	×	
outline-color	visual、 interactive	×	
outline-style	visual、 interactive	×	
outline-width	visual、 interactive	×	
outline	visual、 interactive	×	
overflow	visual	×	
padding-top padding-right padding- bottom padding-left	visual	○	
padding	visual	○	
page-break- after	visual、 paged	×	
page-break- before	visual、 paged	×	
page-break- inside	visual、 paged	×	
pause-after	aural	×	
pause-before	aural	×	
pause	aural	×	
pitch-range	aural	×	
pitch	aural	×	
play-during	aural	×	
position	visual	×	

quotes	visual	×	
richness	aural	×	
right	visual	×	
speak-header	aural	×	
speak-numeral	aural	×	
speak-punctuation	aural	×	
speak	aural	×	
speech-rate	aural	×	
stress	aural	×	
table-layout	visual	×	
text-align	visual	○	
text-decoration	visual	○	
text-indent	visual	○	
text-transform	visual	×	
top	visual	×	
unicode-bidi	visual	×	
vertical-align	visual	○	<percentage> と <length> 以外のすべての値。
visibility	visual	○	
voice-family	aural	×	
volume	aural	×	
white-space	visual	○	nowrap と pre は、normal と pre-wrap として扱われます。
widows	visual、paged	×	
width	visual	○	img 要素内のみ。
word-spacing	visual	×	
z-index	visual	×	

## CSS2 セレクタ

**RichTextBox for WPF** は、多数の CSS2 セレクタをサポートしています。次の表は、CSS2 セレクタのパターンおよび CSS レベルと、それが **RichTextBox for WPF** でサポートされるかどうかを示します。

パターン	CSS レベル	サポート
*	2	○
E	1	○
E[foo]	2	○
E[foo="bar"]	2	○

# RichTextBox for WPF

E[foo~="bar"]	2	○
E[foo^="bar"]	3	○
E[foo\$="bar"]	3	○
E[foo*="bar"]	3	○
E[foo]="en"]	2	○
E:root	3	×
E:nth-child(n)	3	×
E:nth-last-child(n)	3	×
E:nth-of-type(n)	3	×
E:nth-last-of-type(n)	3	×
E:first-child	2	×
E:last-child	3	×
E:first-of-type	3	×
E:last-of-type	3	×
E:only-child	3	×
E:only-of-type	3	×
E:empty	3	×
E:link	1	×
E:visited	1	×
E:active	2	×
E:hover	2	×
E:focus	2	×
E:target	3	×
E:lang(fr)	2	×
E:enabled	3	×
E:disabled	3	×
E:checked	3	×
E::first-line	1	×
E::first-letter	1	×
E::before	2	×
E::after	2	×
E.warning	1	×
E#myid	1	○
E:not(s)	3	×
E F	1	○

$E > F$	2	<input type="radio"/>
$E + F$	2	<input type="radio"/>
$E \sim F$	3	<input type="radio"/>

## RichTextBox for WPF の外観

以下のトピックでは、**C1RichTextBox** コントロールのレイアウトと外観をカスタマイズする方法について詳しく説明します。ClearStyle 技術を使用して、グリッドの一部にスタイルを設定できます。テーマを使用することで、グリッドの外観をカスタマイズしたり、WPF の XAML ベースのスタイル設定を活用することができます。

## ComponentOne ClearStyle 技術

**ClearStyle** は、WPF コントロールと Silverlight コントロールのスタイル設定をすばやく簡単に実行できる新技術です。ClearStyle を使用すると、面倒な XAML テンプレートやスタイルリソースを操作しなくても、コントロールのカスタムスタイルを作成できます。

現在のところ、すべての標準 WPF コントロールにテーマを追加するには、スタイルリソーステンプレートを作成する必要があります。Microsoft Visual Studio ではこの処理は困難であるため、Microsoft は、このタスクを簡単に実行できるように Expression Blend を導入しました。ただし、Blend に不慣れであったり、十分な学習時間を取れない開発者にとっては、この2つの環境を行き来することはかなり困難な作業です。デザイナーに作業を任せることも考えられますが、デザイナーと開発者が XAML ファイルを共有すると、かえって煩雑になる可能性があります。

このような場合に、**ClearStyle** を使用します。**ClearStyle** は、Visual Studio を使用して直感的な方法でスタイル設定を実行できるようにします。ほとんどの場合は、アプリケーション内のコントロールに対して単純なスタイル変更を行うだけなので、この処理は簡単に行えるべきです。たとえば、データグリッドの行の色を変更するだけであれば、1つのプロパティを設定するだけで簡単に行えるようにする必要があります。一部の色を変更するためだけに、完全に複雑なテンプレートを作成する必要はありません。

## ClearStyle の仕組み

コントロールのスタイルの主な要素は、それぞれ単純な色プロパティとして表されます。これが集まって、コントロール固有のスタイルプロパティセットを形成します。たとえば、**Gauge** には **PointerFill** プロパティや **PointerStroke** プロパティがあり、**DataGrid** の行には **SelectedBrush** や **MouseOverBrush** があります。

たとえば、フォーム内に ClearStyle をサポートしていないコントロールがあるとします。その場合は、ClearStyle によって作成された XAML リソースを使用して、フォーム内の他のコントロールを調整して合わせるすることができます（正確な色合わせなど）。また、スタイルセットの一部を ClearStyle（カスタムスクロールバーなど）で上書きしたいとします。ClearStyle は拡張可能なのでこれも可能です。必要な場所でスタイルを上書きできます。

ClearStyle は、すばやく簡単にスタイルを変更することを意図したソリューションですが、ComponentOne のコントロールには引き続き従来の方法を使用して、必要なスタイルを細かく指定して作成できます。完全なカスタム設計が必要になる特別な状況で ClearStyle が邪魔になることはありません。

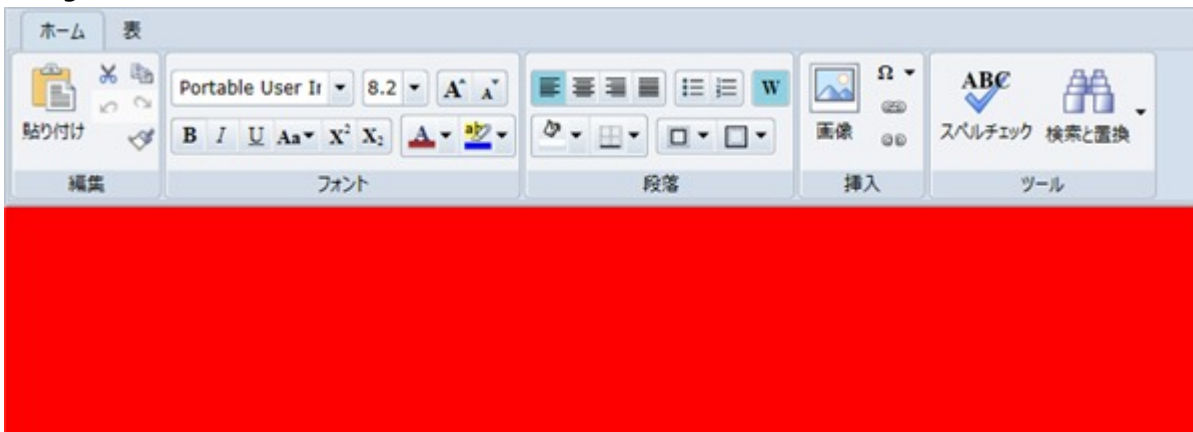
## C1RichTextBox の ClearStyle プロパティ

**RichTextBox for WPF** は、コントロールのテンプレートを変更することなくコントロールの色を簡単に変更できる ComponentOne の新しい ClearStyle 技術をサポートします。色のプロパティをいくつか設定するだけで、コントロールのスタイルを簡単に設定できます。

次の表に、**C1RichTextBox** コントロールのブラシのプロパティの概要を示します。

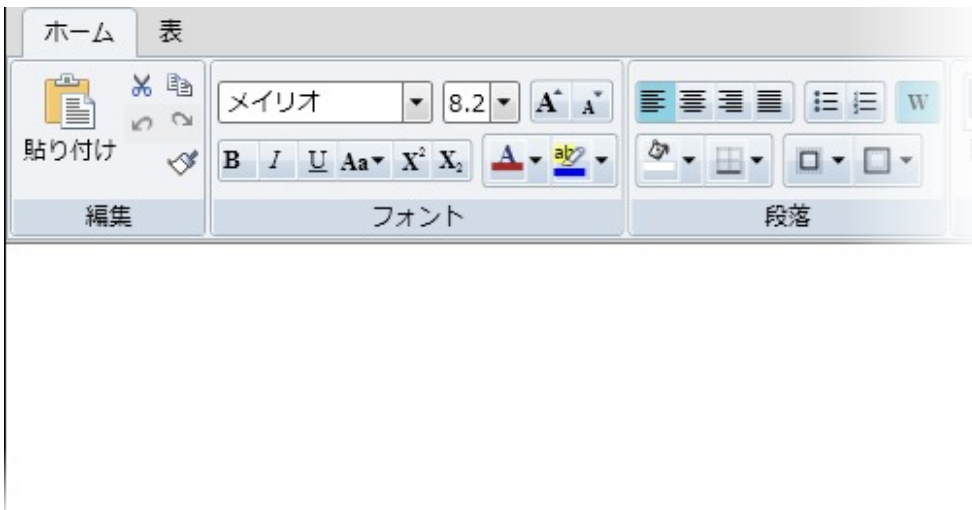
ブラシ	説明
Background	コントロールの背景のブラシを取得または設定します。
BorderBrush	コントロールの境界線のブラシを取得または設定します。
SelectionBackground	選択したテキストの背景の塗りつぶしに使用されるブラシを取得または設定します。
SelectionForeground	選択したテキストの前景の塗りつぶしに使用されるブラシを取得または設定します。

いくつかのプロパティを設定して、**C1RichTextBox** コントロールの外観を完全に変更できます。たとえば、**Background** プロパティを「#FFE20C0C」に設定すると、**C1RichTextBox** コントロールは次のようになります。



## C1RichTextBox のテーマ




**RichTextBox for WPF** には、グリッドの外観をカスタマイズできるいくつかのテーマが組み込まれています。**C1RichTextBox** コントロールを初めてページに追加すると、次の図のように表示されます。



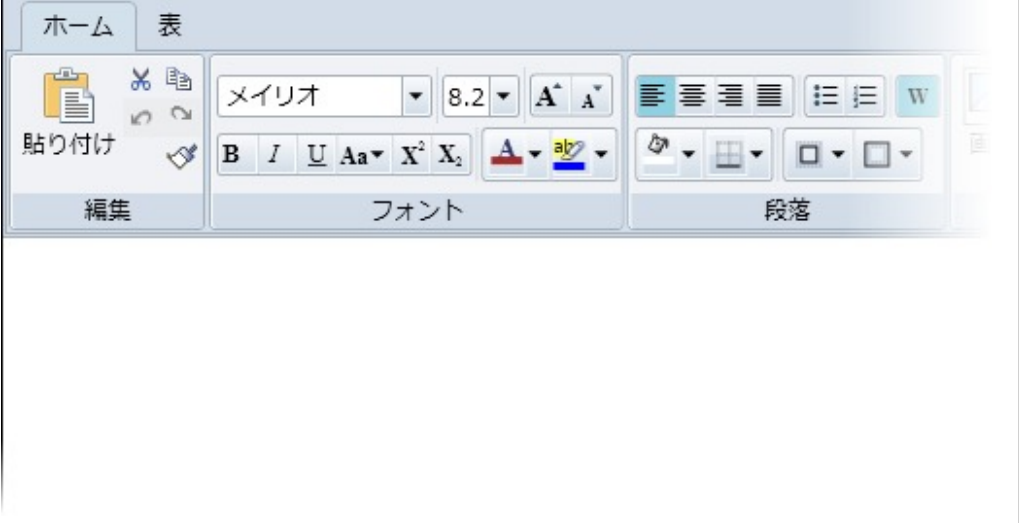
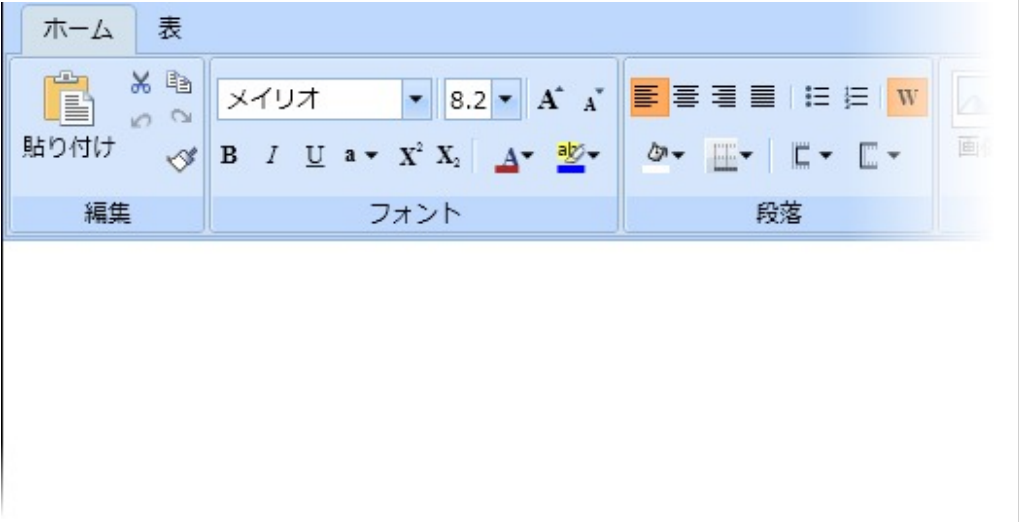
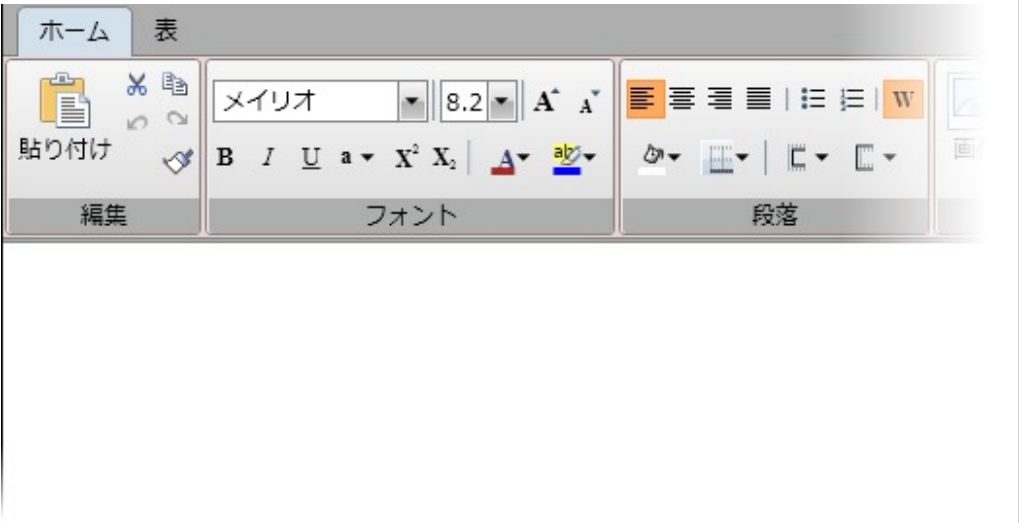
これは、このコントロールのデフォルトの外観です。この外観は、組み込みテーマの1つを使用したり、独自のカスタムテーマを作成することで変更できます。すべての組み込みテーマは、WPF Toolkit テーマに基づいています。以下に、組み込みテーマの説明と図を示します。以下の図では、選択状態のスタイルを示すために1つの行が選択されています。

テーマ名	テーマのプレビュー
C1ThemeBureauBlack	

# RichTextBox for WPF

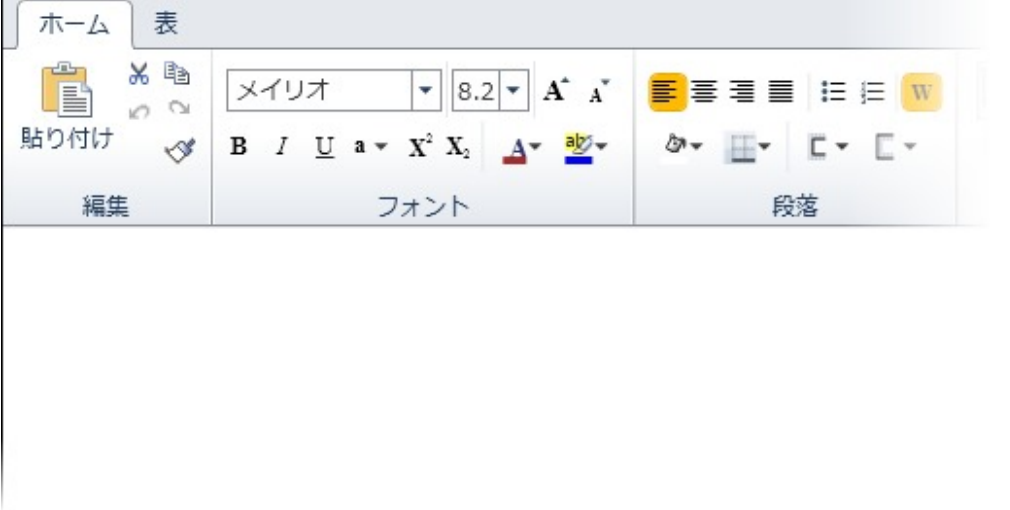
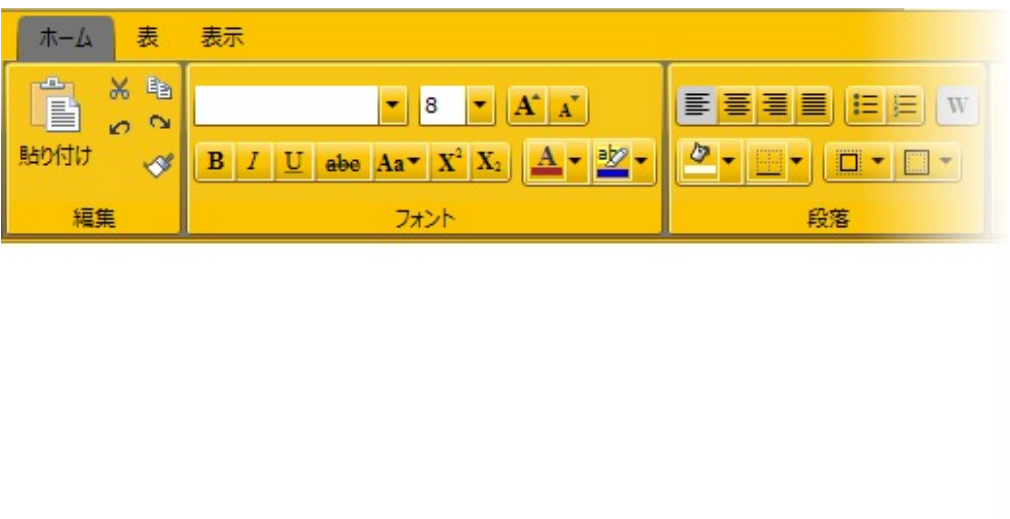

	 <p>The image shows the default ribbon for the RichTextBox control. It features three tabs: 'ホーム' (Home) and '表' (Table). The 'ホーム' tab is active and contains three groups: '貼り付け' (Paste) with icons for paste, copy, and paste as plain text; 'フォント' (Font) with options for font face (メイリオ), size (8.2), bold (B), italic (I), underline (U), text color (A), and background color (ab); and '段落' (Paragraph) with options for bullet points, numbered list, decrease indent, increase indent, and wrap (W).</p>
C1ThemeExpressionDark	 <p>This image shows the ribbon for the C1ThemeExpressionDark theme. The layout and content are identical to the default ribbon, but the color scheme is dark gray, with icons and text rendered in lighter colors for contrast.</p>
C1ThemeExpressionLight	 <p>This image shows the ribbon for the C1ThemeExpressionLight theme. The layout and content are identical to the default ribbon, but the color scheme is light gray, with icons and text rendered in darker colors for contrast.</p>
C1Blue (WPF のみ)	

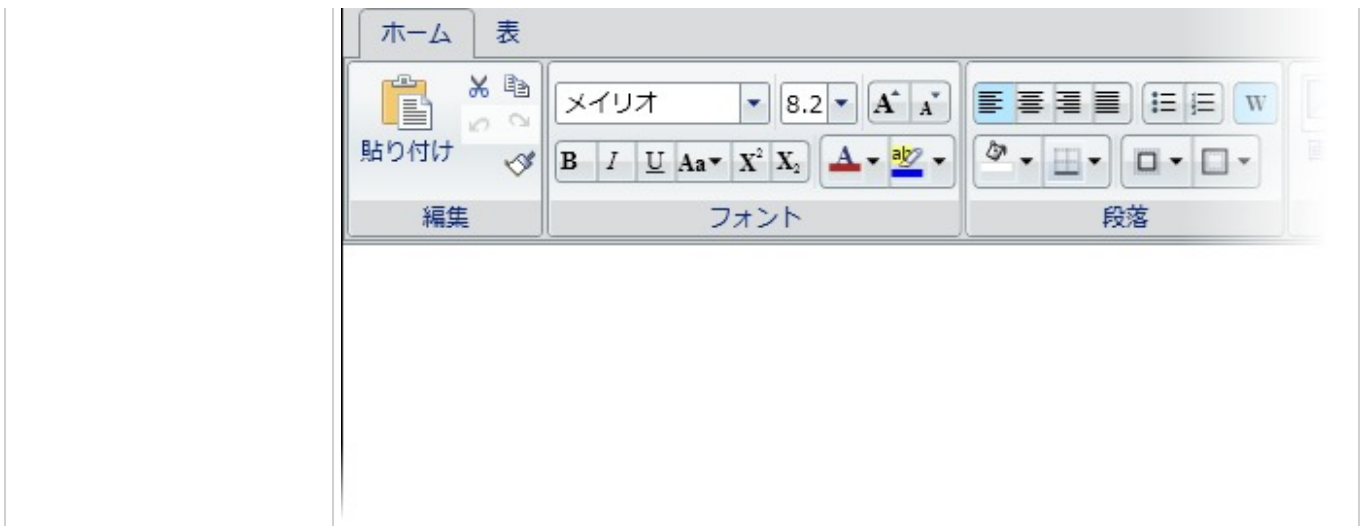


	 A screenshot of the Microsoft Office 2007 ribbon interface. The ribbon is titled 'ホーム' (Home) and '表' (Table). It is divided into three main sections: '貼り付け' (Paste), 'フォント' (Font), and '段落' (Paragraph). The '貼り付け' section includes icons for Paste, Copy, and Paste as Plain Text. The 'フォント' section includes a font name dropdown (currently 'メイリオ'), font size (8.2), bold (B), italic (I), underline (U), text color (A), and background color (ab) options. The '段落' section includes bullet point, numbered list, and paragraph alignment (left, center, right, justified) options.
C1ThemeOffice2007Blue	 A screenshot of the Microsoft Office 2007 ribbon interface with a blue theme. The ribbon is titled 'ホーム' (Home) and '表' (Table). It is divided into three main sections: '貼り付け' (Paste), 'フォント' (Font), and '段落' (Paragraph). The '貼り付け' section includes icons for Paste, Copy, and Paste as Plain Text. The 'フォント' section includes a font name dropdown (currently 'メイリオ'), font size (8.2), bold (B), italic (I), underline (U), text color (A), and background color (a) options. The '段落' section includes bullet point, numbered list, and paragraph alignment (left, center, right, justified) options.
C1ThemeOffice2007Black	 A screenshot of the Microsoft Office 2007 ribbon interface with a black theme. The ribbon is titled 'ホーム' (Home) and '表' (Table). It is divided into three main sections: '貼り付け' (Paste), 'フォント' (Font), and '段落' (Paragraph). The '貼り付け' section includes icons for Paste, Copy, and Paste as Plain Text. The 'フォント' section includes a font name dropdown (currently 'メイリオ'), font size (8.2), bold (B), italic (I), underline (U), text color (a), and background color (ab) options. The '段落' section includes bullet point, numbered list, and paragraph alignment (left, center, right, justified) options.
C1ThemeOffice2007Silver	

# RichTextBox for WPF

	 <p>The image shows a RichTextBox control with a light gray background. The ribbon is divided into three main sections: 'ホーム' (Home), '表' (Table), and '段落' (Paragraph). The 'ホーム' section contains '貼り付け' (Paste) and '編集' (Edit) buttons. The '表' section contains a font name dropdown (メイリオ), font size (8.2), and font style buttons (A<sup>+</sup>, A<sup>-</sup>, B, I, U, a, X<sup>2</sup>, X<sub>2</sub>). The '段落' section contains paragraph alignment and style buttons.</p>
C1ThemeOffice2010Blue	 <p>The image shows the RichTextBox control with the C1ThemeOffice2010Blue theme. The ribbon is light blue. The '貼り付け' button is highlighted in yellow, and the 'W' button in the paragraph section is also highlighted in yellow.</p>
C1ThemeOffice2010Black	 <p>The image shows the RichTextBox control with the C1ThemeOffice2010Black theme. The ribbon is dark gray. The '貼り付け' button is highlighted in light gray, and the 'W' button in the paragraph section is also highlighted in light gray.</p>
C1ThemeOffice2010Silver	

	 <p>The screenshot shows the ribbon interface of a RichTextBox in its default theme. It features three tabs: 'ホーム' (Home), '表' (Table), and '表示' (Display). The 'ホーム' tab is active and divided into three groups: '編集' (Edit), 'フォント' (Font), and '段落' (Paragraph). The '編集' group contains icons for paste, undo, redo, and delete. The 'フォント' group includes a font name dropdown (currently 'メイリオ'), a font size dropdown (8.2), and buttons for bold, italic, underline, text color, and background color. The '段落' group contains icons for bulleted list, numbered list, decrease indent, and increase indent.</p>
<p>C1ThemeRainierOrange (Silverlight のみ)</p>	 <p>This screenshot shows the ribbon interface with the C1ThemeRainierOrange theme. The ribbon has a yellow background. The tabs are 'ホーム', '表', and '表示'. The 'ホーム' tab is active and contains the same functional groups as the default theme, with icons and text labels in a yellow color scheme.</p>
<p>C1ThemeShinyBlue</p>	 <p>This screenshot shows the ribbon interface with the C1ThemeShinyBlue theme. The ribbon has a blue background. The tabs are 'ホーム', '表', and '表示'. The 'ホーム' tab is active and contains the same functional groups as the default theme, with icons and text labels in a blue color scheme.</p>
<p>C1ThemeWhistlerBlue</p>	



要素のテーマを設定するには、**ApplyTheme** メソッドを使用します。最初に、テーマアセンブリへの参照をプロジェクトに追加し、次のようにコードでテーマを設定します。

## VisualBasic

```
Private Sub Window_Loaded(sender As System.Object, e As System.Windows.RoutedEventArgs)
    Handles MyBase.Loaded
    Dim theme As New C1ThemeExpressionDark
    ' ApplyTheme の使用
    C1Theme.ApplyTheme(LayoutRoot, theme)
```

## C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    C1ThemeExpressionDark theme = new C1ThemeExpressionDark();
    //ApplyTheme の使用
    C1Theme.ApplyTheme(LayoutRoot, theme);
}
```

アプリケーション全体にテーマを適用するには、**System.Windows.ResourceDictionary.MergedDictionaries** プロパティを使用します。最初に、テーマアセンブリへの参照をプロジェクトに追加し、次のようにコードでテーマを設定します。

## VisualBasic

```
Private Sub Window_Loaded(sender As System.Object, e As System.Windows.RoutedEventArgs)
    Handles MyBase.Loaded
    Dim theme As New C1ThemeExpressionDark
    ' MergedDictionaries の使用
    Application.Current.Resources.MergedDictionaries.Add(C1Theme.GetCurrentThemeResources(theme))
End Sub
```

## C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    C1ThemeExpressionDark
    theme = new C1ThemeExpressionDark(); //MergedDictionaries の使用
    Application.Current.Resources.MergedDictionaries.Add(C1Theme.GetCurrentThemeResources(theme));
}
```

この方法は、初めてテーマを適用する場合にのみ使用できることに注意してください。別の ComponentOne テーマに切り替える場合は、最初に、**Application.Current.Resources.MergedDictionaries** から前のテーマを削除します。

## スペルチェック

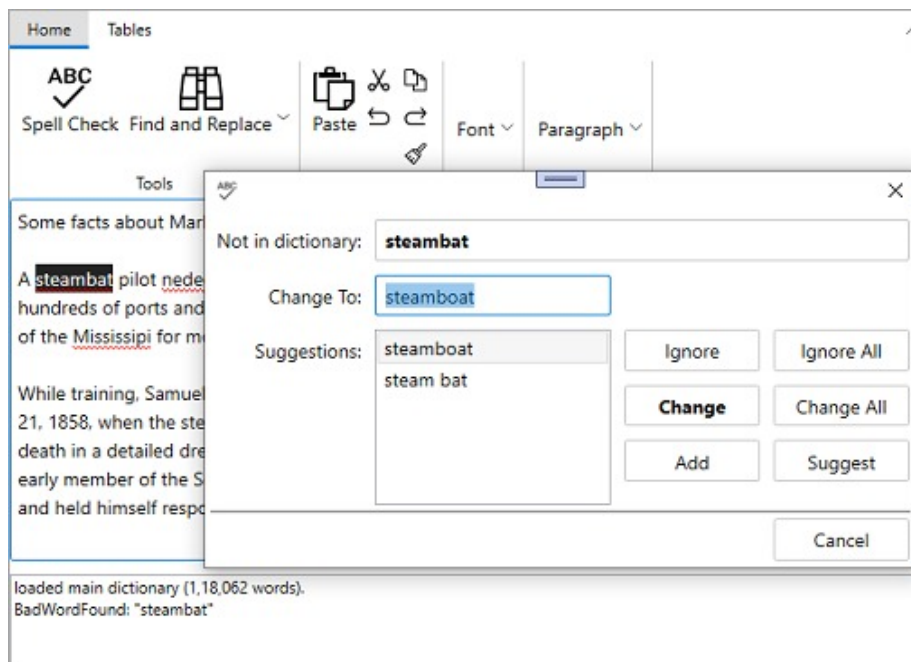
高性能なエディタの多くは、次の2種類のスペルチェックを実装しています。

- **モーダルスペルチェック**: [スペル] ダイアログボックスを表示して、ドキュメント内のスペルミスを一別々に選択します。スペルミスを見無視したり、正しいスペルを入力するか候補リストから選択してミスを見修正したり、その単語を辞書に追加することができます。
- **入力中スペルチェック**: 入力中にスペルミスを見通常は赤い波線の下線で強調表示します。ドキュメント内のスペルミスを見右クリックすると、見無視する、辞書に追加する、スペル候補から選択してスペルミスを見自動的に修正するなどのオプションメニューが表示されます。

**C1RichTextBox** は、**C1SpellChecker** コンポーネントを使用した2種類のスペルチェックを見サポートします。このコンポーネントも、ComponentOne for WPF に見含まれています。**C1SpellChecker** は、他のコントロールも見スペルチェックができるため、別のアセンブリとしてリリースされています。

## Modal Spell Checking

To implement modal spell checking, you need to add a reference to the **C1.WPF.SpellChecker** assembly to your project. Then, add the following code to your project. This code creates a new **C1SpellChecker** object to be shared by all controls on the page that require spell-checking. Later, the page constructor invokes the **Load** method to load the main spelling dictionary from a stream containing the compressed Dictionary data. **C1SpellChecker** includes over 20 other dictionaries which can be downloaded from our site. In this case, we are loading **C1Spell\_en-US.dct**, the American English dictionary. This file must be present on the application folder. When the modal checking is complete, the **\_c1SpellChecker\_CheckControlCompleted** event fires and shows a dialog box to indicate that the spell-checking operation is complete.



C#

```
public partial class SpellCheckerRichTextBoxDemo : UserControl
{
    // C1SpellCheckerを宣言します
    C1SpellChecker _c1SpellChecker = new C1SpellChecker();

    public SpellCheckerRichTextBoxDemo()
    {
        InitializeComponent();
        this.Tag = Properties.Resources.SpellCheckerRtbDemoDescription;
        Loaded += Page_Loaded;
    }
}
```

```

        Unloaded += Page_Unloaded;
    }

    void Page_Loaded(object sender, RoutedEventArgs e)
    {
        // C1RichTextBoxにツールバーを接続します
        _rtbToolBar.RichTextBox = _richTextBox;
        _richTextBox.SpellChecker = _c1SpellChecker;

        // サンプルテキストをテキストボックスにロードします
        using (var stream =
Assembly.GetExecutingAssembly().GetManifestResourceStream("SpellCheckerExplorer.Resources.test.txt"))
        using (var sr = new StreamReader(stream))
        {
            var text = sr.ReadToEnd();
            _richTextBox.Text = text;
        }

        // 無視リストを設定します
        WordList il = _c1SpellChecker.IgnoreList;
        il.Add("ComponentOne");
        il.Add("Silverlight");

        // イベントを監視します
        _c1SpellChecker.BadWordFound += _c1SpellChecker_BadWordFound;
        _c1SpellChecker.CheckControlCompleted += _c1SpellChecker_CheckControlCompleted;

        // メイン辞書をロードします
        if (_c1SpellChecker.MainDictionary.State != DictionaryState.Loaded)
            _c1SpellChecker.MainDictionary.Load(Application.GetResourceStream(new Uri("/") + new
AssemblyName(Assembly.GetExecutingAssembly().FullName).Name + ";component/Resources/C1Spell_en-
US.dct", UriKind.Relative)).Stream);
        if (_c1SpellChecker.MainDictionary.State == DictionaryState.Loaded)
        {
            WriteLine("loaded main dictionary ({0:n0} words).",
_c1SpellChecker.MainDictionary.WordCount);
        }
        else
        {
            WriteLine("failed to load dictionary: {0}", _c1SpellChecker.MainDictionary.State);
        }

        // アプリの終了時にユーザー辞書を保存します
        App.Current.Exit += App_Exit;
    }

    void Page_Unloaded(object sender, RoutedEventArgs e)
    {
        _c1SpellChecker.BadWordFound -= _c1SpellChecker_BadWordFound;
        _c1SpellChecker.CheckControlCompleted -= _c1SpellChecker_CheckControlCompleted;
    }

    // スペルチェッカーイベントを監視します
    void _c1SpellChecker_CheckControlCompleted(object sender, CheckControlCompletedEventArgs e)
    {
        if (!e.Cancelled)

```

## RichTextBox for WPF

```
    {
        var msg = string.Format("Spell-check complete, {0} errors found.", e.ErrorCount);
        MessageBox.Show(msg, "Spelling");
    }
    WriteLine("CheckControlCompleted: {0} errors found", e.ErrorCount);
    if (e.Cancelled)
    {
        WriteLine("\t(cancelled...)");
    }
}
void _c1SpellChecker_BadWordFound(object sender, BadWordEventArgs e)
{
    WriteLine("BadWordFound: \"{0}\" {1}", e.BadWord.Text, e.BadWord.Duplicate ? "
(duplicate)" : string.Empty);
}
}
```



## タスク別ヘルプ

タスク別ヘルプは、ユーザーの皆様が Visual Studio .NET でのプログラミングに精通しており、**C1RichTextBox** コントロールの一般的な使用方法を理解していることを前提としています。**RichTextBox for WPF** 製品を初めて使用される場合は、まず「[クイックスタート](#)」を参照してください。

このセクションの各トピックは、**RichTextBox for WPF** 製品を使用して特定のタスクを実行するためのソリューションを提供します。

また、タスク別ヘルプトピックは、新しい WPF プロジェクトが既に作成されていることを前提としています。

## テキストコンテンツの設定

**Text** プロパティは、**C1RichTextBox** コントロールのテキストのコンテンツを決定します。デフォルトでは、**C1RichTextBox** コントロールは最初は空白で、コンテンツがありませんが、設計時、XAML、またはコードでこの値をカスタマイズできます。HTML マークアップをコントロールのコンテンツとして設定することもできます。詳細については、「[HTML コンテンツの設定](#)」を参照してください。

### 設計時

**Text** プロパティを設定するには、次の手順に従います。

1. **C1RichTextBox** コントロールをクリックして選択します。
2. [プロパティ] ウィンドウに移動し、**Text** プロパティの横にあるテキストボックスにテキスト（たとえば、Hello World!）を入力します。  
これで、**Text** プロパティは指定された値に設定されます。

### XAML の場合

たとえば、**Text** プロパティを設定するには、次に示すように `Text="Hello World!"` を `<c1rtb:C1RichTextBox>` タグに追加します。

XAML

```
<c1rtb:C1RichTextBox HorizontalAlignment="Left" Margin="10,10,0,0" Name="C1RichTextBox1" VerticalAlignment="Top" Height="83" Width="208" Text="Hello World!" />
```

### コードの場合

たとえば、**Text** プロパティを設定するには、プロジェクトに次のコードを追加します。

## VisualBasic

```
C1RichTextBox1.Text = "Hello World!"
```


## C#

# RichTextBox for WPF

```
c1RichTextBox1.Text = "Hello World!";
```

## ここまでの成果

**C1RichTextBox** コントロールのテキストコンテンツを設定しました。アプリケーションを実行すると、最初に「Hello World!」（または選択したテキスト）がコントロールに表示されます。



HTML マークアップをコントロールのコンテンツとして設定することもできます。詳細については、「[HTML コンテンツの設定](#)」を参照してください。

## HTML コンテンツの設定

**Html** プロパティは、**C1RichTextBox** コントロールの HTML マークアップのコンテンツを決定します。デフォルトでは、**C1RichTextBox** コントロールは最初は空白で、コンテンツがありませんが、設計時、XAML、またはコードでこの値をカスタマイズできます。テキストをコントロールのコンテンツとして設定することもできます。詳細については、「[テキストコンテンツの設定](#)」を参照してください。

### 設計時

**Html** プロパティを設定するには、次の手順に従います。

1. **C1RichTextBox** コントロールをクリックして選択します。
2. [プロパティ] ウィンドウに移動し、**Html** プロパティの隣にあるテキストボックスに "&lt;h1&gt;Hello World!&lt;/h1&gt;" などのテキストを入力します。  
これで、**Html** プロパティは指定された値に設定されます。

### XAML の場合

たとえば、**Html** プロパティを設定するには、次に示すように `Html="Hello World!"` を `<c1rtb:C1RichTextBox>` タグに追加します。

XAML

```
<c1rtb:C1RichTextBox HorizontalAlignment="Left" Margin="10,10,0,0" Name="C1RichTextBox1" VerticalAlignment="Top" c1:C1NagScreen.Nag="True" Height="83" Width="208" Html="<h1>Hello World!</h1>" />
```

### コードの場合

たとえば、**Html** プロパティを設定するには、プロジェクトに次のコードを追加します。

## VisualBasic

```
Me.C1RichTextBox1.Html = "<h1>Hello World!</h1>"
```

## C#

```
this.c1RichTextBox1.Html = "<h1>Hello World!</h1>";
```

### ここまでの成果

**C1RichTextBox** コントロールのテキストコンテンツを設定しました。不等号かっこ (< >) は "<" や ">" のように記述する必要があります。アプリケーションを実行すると、最初に「Hello World!」（または選択したテキスト）が大きなテキストでコントロールに表示されます。



テキストをコントロールのコンテンツとして設定することもできます。詳細については、「[テキストコンテンツの設定](#)」を参照してください。**C1RichTextBox** コントロールのコンテンツにハイパーリンクを追加する例については、「[ハイパーリンク](#)」を参照してください。

## C1RichTextBoxToolBar を C1RichTextBox に接続する

**C1RichTextBoxToolBar** コントロールを **C1RichTextBox** コントロールに接続するには、**RichTextBox** プロパティをこのツールバーにリンクするコントロールの名前に設定します。設計時、XAML、およびコードで **C1RichTextBoxToolBar** コントロールを **C1RichTextBox** コントロールに接続できます。

### 設計時

Microsoft Expression Blend で **RichTextBox** プロパティを設定するには、次の手順に従います。

1. **C1RichTextBox** コントロールをクリックして選択します。
2. [プロパティ] ウィンドウに移動し、**RichTextBox** プロパティの横にある [詳細オプション] ボタンをクリックします。
3. [データのバインドの作成] ダイアログボックスで [要素プロパティ] タブをクリックします。
4. [要素プロパティ] タブで、**C1RichTextBox1** 項目を選択し、[OK] をクリックします。  
これで、**RichTextBox** プロパティが C1RichTextBox1 に設定されます。

### XAML の場合

たとえば、**RichTextBox** プロパティを設定するには、次に示すように `RichTextBox="{Binding ElementName=C1RichTextBox1}"` を `<c1rtb:C1RichTextBox>` タグに追加します。

XAML

```
<c1rtb:C1RichTextBox HorizontalAlignment="Left" Margin="10,10,0,0" Name="C1RichTextBox1"
VerticalAlignment="Top" c1:C1NagScreen.Nag="True" Height="83" Width="208"
RichTextBox="{Binding ElementName=C1RichTextBox1}" />
```

## コードの場合

たとえば、**RichTextBox** プロパティを設定するには、プロジェクトに次のコードを追加します。

## VisualBasic

```
Me.C1RichTextBoxToolBar1.RichTextBox = C1RichTextBox1
```

## C#

```
this.c1RichTextBoxToolBar1.RichTextBox = c1RichTextBox1;
```

## ここまでの成果

**C1RichTextBoxToolBar** を **C1RichTextBox** コントロールにリンクしました。**C1RichTextBox** にテキストを入力すると、フォントなどのコンテンツの外観が変化します。**C1RichTextBox** コントロールの詳細については、「[C1RichTextBoxToolBar の使い方](#)」を参照してください。

## 簡単な書式設定ツールバーの実装

**C1RichTextBoxToolBar** コントロールを使用すると、**C1RichTextBox** コントロールで使用できる完全なツールバーを追加できますが、独自のツールバーを簡単に作成することもできます。ほとんどの高機能エディタには、現在の選択に太字、斜体、または下線の書式設定を適用するボタンを含むツールバーがあります。選択範囲が移動すると、これらのボタンの状態も変化し、選択されているテキストに太字、斜体、下線などの書式設定が適用されているかどうかを示されます。

**C1RichTextBox** を使用してシンプルなツールバーを実装するのは簡単です。たとえば、次の手順に従います。

1. ソリューションエクスプローラで、プロジェクトを右クリックし、コンテキストメニューから **[参照の追加]** を選択します。
2. **[参照の追加]** ダイアログボックスで、**C1.WPF.RichTextBox.4** または **C1.Silverlight.RichTextBox.5** アセンブリを選択したら、**[OK]** をクリックします。
3. ページの XAML を更新します。次のようになります。

### XAML

```
<Window xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml" x:Class="
    C1RichTextBoxIntro.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow">
<Grid x:Name="LayoutRoot" Background="White">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition />
    </Grid.RowDefinitions>
```

```

<StackPanel Orientation="Horizontal" >
  <ToggleButton x:Name="_btnBold" Content="B" Click="_btnBold_Click"
    Checked="_btnBold_Click" />
  <ToggleButton x:Name="_btnItalic" Content="I" Click="_btnItalic_Click" />
  <ToggleButton x:Name="_btnUnderline" Content="U"
Click="_btnUnderline_Click" />
</StackPanel>
<c1:C1RichTextBox x:Name="_rtb" Grid.Row="1"
AcceptsReturn="True"
SelectionChanged="_rtb_SelectionChanged"/>
</Grid>

```

このマークアップは、**C1RichTextBox** コントロールと、その書式設定を制御する3つのボタン（太字、斜体、下線）を追加します。ボタンをクリックすると、関連付けられているイベントハンドラが選択範囲の書式設定を更新します。次の手順のコードがそれを実行します。

4. ページを右クリックし、[コードの表示] を選択してコードエディタに切り替えます。
5. 次のコードをアプリケーションに追加します。

## VisualBasic

```

Private Sub _btnBold_Click(sender As Object, e As RoutedEventArgs)
Dim fw As System.Nullable(Of FontWeight) = _rtb.Selection.FontWeight
_rtb.Selection.FontWeight = If(fw.HasValue AndAlso fw.Value
= FontWeights.Bold, FontWeights.Normal, FontWeights.Bold)
End Sub

```

## C#

```

private void _btnBold_Click(object sender, RoutedEventArgs e)
{
    FontWeight? fw = _rtb.Selection.FontWeight;
    _rtb.Selection.FontWeight = fw.HasValue && fw.Value == FontWeights.Bold
        ? FontWeights.Normal
        : FontWeights.Bold;
}

```

このコードは、最初に現在の選択範囲の **FontWeight** プロパティの値を取得します。null 可能な値が返されます（したがって、型宣言に '?' があります）。選択範囲に複数のフォントウェイトが混在している場合は、null 値が返されます。上のコードでは、選択範囲全体に単一のフォントウェイトとして太字が設定されている場合はフォントウェイトを "normal" に設定し、それ以外の場合はフォントウェイトを "bold" に設定しています。

6. 次のコードを追加して、斜体ボタンを初期化します。

## VisualBasic

```
Private Sub _btnItalic_Click(sender As Object, e As RoutedEventArgs)
    Dim fs As System.Nullable(Of FontStyle) = _rtb.Selection.FontStyle
    _rtb.Selection.FontStyle = If(fs.HasValue AndAlso fs.Value =
        FontStyles.Italic,FontStyles.Normal, FontStyles.Italic)
End Sub
```

## C#

```
private void _btnItalic_Click(object sender, RoutedEventArgs e)
{
    FontStyle? fs = _rtb.Selection.FontStyle;
    _rtb.Selection.FontStyle = fs.HasValue && fs.Value == FontStyles.Italic
        ? FontStyles.Normal
        : FontStyles.Italic;
}
```

斜体ボタンを処理するコードは、**FontWeight** プロパティの代わりに **FontStyle** プロパティを使用することを除いて、太字ボタンを処理するコードとほとんど同じです。

7. 次のコードを追加して、下線ボタンを初期化します。

## VisualBasic

```
Private Sub _btnUnderline_Click(sender As Object, e As RoutedEventArgs)
    If _btnUnderline.IsChecked.HasValue Then rtb.Selection.TextDecorations =
        If(_btnUnderline.IsChecked.Value, C1TextDecorations.Underline, Nothing)
    End If
End Sub
```

## C#

```
private void _btnUnderline_Click(object sender, RoutedEventArgs e)
{
    if (_btnUnderline.IsChecked.HasValue)
    {
        _rtb.Selection.TextDecorations = _btnUnderline.IsChecked.Value
            ? C1.WPF.RichTextBox.Documents.C1TextDecorations.Underline
            : null;
    }
}
```

下線ボタンを処理するコードも同様ですが、この場合は **TextDecorations** プロパティを使用します。**TextDecorations** プロパティは実際のオブジェクトを返すので、null 可能プロパティではありません。上のコードで、3つのボタンが動作します。

8. 次のコードを追加して、**SelectionChanged** イベントのイベントハンドラを実装します。

## VisualBasic

```
Private Sub _rtb_SelectionChanged(sender As Object, e As EventArgs)
    Dim fw As System.Nullable(Of FontWeight) = _rtb.Selection.FontWeight
    _btnBold.IsChecked = fw.HasValue AndAlso fw.Value = FontWeights.Bold
    Dim fs As System.Nullable(Of FontStyle) = _rtb.Selection.FontStyle
    _btnItalic.IsChecked = fs.HasValue AndAlso fs.Value = FontStyles.Italic
    _btnUnderline.IsChecked = (sel.TextDecorations IsNot Nothing)
End Sub
```

## C#

```
void _rtb_SelectionChanged(object sender, EventArgs e)
{
    var sel = _rtb.Selection;
    FontWeight? fw = _rtb.Selection.FontWeight;
    _btnBold.IsChecked = fw.HasValue && fw.Value == FontWeights.Bold;
    FontStyle? fs = _rtb.Selection.FontStyle;
    _btnItalic.IsChecked = fs.HasValue && fs.Value == FontStyles.Italic;
    _btnUnderline.IsChecked = (sel.TextDecorations != null);
}
```

このイベントハンドラは、ユーザーが選択範囲を移動するのに合わせてボタンの状態を変更します。たとえば、太字で下線付きの単語を選択すると、それらのボタンが押下した状態になります。このコードは、前と同様に **FontWeight**、**FontStyle**、**TextDecorations** の各プロパティを使用し、対応するボタンの **IsChecked** プロパティを設定します。

### ここまでの成果

簡単なツールバーを作成しました。実行すると、アプリケーションは次の画像のようになります。



テキストを入力し、太字、斜体、下線の各ボタンを押下すると、上の図のようにテキストを書式設定できます。

完全なツールバーには、さらに多くのボタンとコントロールが含まれますが、同様の方法で処理されます。

**ComponentOne for WPF** には、完全なツールバー **C1RichTextBoxToolbar** が別のアセンブリとして含まれます。**C1RichTextBoxToolbar** コントロールのソースコードが付属しているため、カスタマイズしたツールバーを作成できます。詳細については、「[C1RichTextBoxToolbar の使い方](#)」を参照してください。

## スペルチェックの追加

# RichTextBox for WPF

このトピックでは、アプリケーションにスペルチェックを追加します。このトピックは、**C1RichTextBox** コントロールと **C1RichTextBoxToolBar** コントロールをページに追加し、この2つをリンクしていることを前提とします。実行時にツールバーの **[スペルチェック]** ボタンをクリックすると、現在はスペルチェックが設定されていないというメッセージが表示されます。この手順では、辞書を追加してスペルチェックを設定します。

次の手順に従います。

1. ソリューションエクスプローラで、**.Web** プロジェクトを右クリックし、**[追加] → [既存の項目]** を選択します。**[既存項目の追加]** ダイアログボックスが表示されます。
2. **[既存項目の追加]** ダイアログボックスで、**RichTextBoxSamples** サンプルフォルダ内にある **C1Spell\_en-US.dct** ファイルを見つけます。デフォルトでは、**ComponentOne Samples\WPF 4.0\C1.WPF.RichTextBox\RichTextBoxSamples\RichTextBoxSamples.Web** または **ComponentOne Samples\Silverlight 4.0\C1.Silverlight.RichTextBox\RichTextBoxSamples\RichTextBoxSamples.Web** の **Documents** フォルダにインストールされます。  
これはアメリカ英語の辞書ファイルです。代わりに別のファイルを追加する場合は、適切なコードを使用して以下の手順を変更できます。
3. ソリューションエクスプローラで、**MainPage.xaml** ファイルを右クリックし、**[コードの表示]** を選択して、コードファイルを開きます。
4. コードエディタで、次のコードを追加して次の名前空間をインポートします。

## VisualBasic

```
Imports C1.WPF.RichTextBox
Imports C1.WPF.SpellChecker
```

## C#

```
using C1.WPF.RichTextBox;
using C1.WPF.SpellChecker;
```

5. 次のコードを **MainPage** コンストラクタに追加します。

## VisualBasic

```
Public Sub New()
    InitializeComponent()
    Dim spell As New C1SpellChecker()
    spell.MainDictionary.LoadAsync("C1Spell_en-US.dct")
    Me.C1RichTextBox.SpellChecker = spell
End Sub
```

## C#



```
public MainPage()
{
    InitializeComponent();
    var spell = new C1SpellChecker();
    spell.MainDictionary.LoadAsync("C1Spell_en-US.dct");
    this.c1RichTextBox.SpellChecker = spell;
}
```

このコードは、入力中スペルチェックを含むスペルチェックをアプリケーションに追加します。

## ここまでの成果

この手順では、**C1RichTextBox** アプリケーションにスペルチェック機能を追加しました。**C1RichTextBox** に入力すると、入力時スペルチェックが起動し、スペルミスの単語に赤い下線が表示されます。**C1RichTextBoxToolbar** の [スペルチェック] ボタンをクリックすると、今度は [スペル] ダイアログボックスが表示されます。