RichTextBox for UWP

2018.04.10 更新

グレープシティ株式会社

RichTextBox for UWP

目次

RichTextBox for UWP

RichTextBox for UWP を使用して、書式設定されたテキストを HTML ドキュメントとして表示および編集します。**C1RichTextBox** コントロールは、基本的な HTML 書式設定、CSS、リスト、ハイパーリンク、テーブル、イメージなどの機能をサポートします。このコントロールを使用して、Web から取得した HTML コンテンツを表示します。リッチテキストエディタとして使用することもできます。

アセンブリとコントロール

モバイルアプリケーションにリッチテキストの強力な表示機能および編集機能を追加します。**RichTextBox for UWP** は、一般的な書式設定タグ、自動的な行の折り返し、HTML と RTF のインポート/エクスポート、テーブルのサポート、イメージ、注釈などをサポートします。

C1.Xaml.RichTextBox.dll アセンブリには、次の主要なクラスが含まれます。

- **C1RichTextBox**
  C1Document ドキュメントの表示および編集に使用するコントロールです。このコントロールを使用して、HTML をインポートおよびエクスポートすることができます。このコントロールは、フォント、前景色、背景色、境界線、段落の配置、画像、ハイパーリンク、リスト、任意の **UIElement** オブジェクトなどのさまざまな書式をサポートしています。
- **C1Document**
  ドキュメントを段落、リスト、画像などのドキュメントコンポーネントを表す要素の階層化リストとして表現するクラスです。**C1Document** オブジェクトによって公開されるオブジェクトモデルは、WPF FlowDocument クラスおよび HTML DOM で使用されるオブジェクトモデルに似ています。

主な特長

RichTextBox for UWP 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。**RichTextBox for UWP** は、次の主な特長を備えています。

- **HTML ドキュメントのロード、表示、保存**
  C1RichTextBox コントロールは、HTML として書式設定されたリッチテキストの表示および編集をサポートします。既存の HTML コンテンツを **C1RichTextBox** コントロールにロードし、ドキュメントを編集し、HTML またはプレーンテキストとして保存します。
- **マウスとタッチのサポート**
  テキストを選択し、マウスまたはタッチ入力を使用して、入力カレットを簡単に移動できます。テキスト選択は、ネイティブの **TextBox** の動作をモデルにしているので、タッチデバイス上で使いやすく簡単に操作できます。
- **豊富な書式設定**
  複数のフォント、装飾、サイズ、色など、CSS やインラインマークアップでサポートされている基本的な HTML スタイル属性を含むテキストを編集および書式設定できます。
- **クリップボードのサポート**
  C1RichTextBox コントロールは、キーボード（CTRL+C、CTRL+V、CTRL+X）やタッチディスプレイの UI コマンドを使用して、プレーンテキストとリッチテキストの両方でクリップボードを完全にサポートします。AppBar またはその他の考えられるすべての方法で切り取り/コピー/貼り付けのコマンドを実装します。
- **組み込みの AppBar ツール**
  C1.Xaml.RichTextBox.AppBar ライブラリには、**C1RichTextBox** コントロールと共に使用してテキスト書式設定、ドキュメント履歴などの一般的な編集機能のコマンドを迅速に作成できる組み込みツールが用意されています。組み込みツールは、Bold、Italic、Underline, Undo、Redo、Increase Font Size、Decrease Font Size、Center Align、Right Align、Left Align、Justify の各コマンドをサポートします。
- **ハイパーリンク：挿入とナビゲーション**
  C1RichTextBox コントロールは、ハイパーリンクの挿入とナビゲーションをサポートします。ユーザーがハイパーリンクをクリックすると、コントロールで **RequestNavigate** イベントが発生するので、そこから起こす動作を処理できます。
- **テーブルとイメージの挿入**
  C1RichTextBox コントロールは、Web またはユーザーのコンピュータから取得したイメージの挿入をサポートします。テーブルも挿入できます。ベータ版では、テーブルとイメージの編集のサポートは制限されています。
- **元に戻す/やり直しのサポート**
  C1RichTextBox コントロールは、すべてのドキュメント履歴を記録するので、容易に変更の元に戻す/やり直しができます。デフォルトでは、キーボードコマンド（CTRL+Z/CTRL+Y）でこのアクションを実行できます。さらに、**C1UndoTool** と **C1RedoTool** を使用すると、UI にボタンを追加して同じ動作を実行できます。

クイックスタート

このクイックスタートでは、Visual Studio で Windows ストアアプリケーションを作成し、アプリケーションに **C1RichTextBox** 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の操作を確認します。

手順1：Windows ストアアプリケーションの作成

この手順では、新しい Windows ストアアプリケーションを作成し、アプリケーションをセットアップし、アプリケーションに **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。

次の手順を実行します。

1. Visual Studio で、**[ファイル]**→**[新規作成]**→**[プロジェクト]**を選択します。
2. **[新しいプロジェクト]**ダイアログボックスで、**[テンプレート]**→**[Visual C#]**→**[Windows]**→**[ユニバーサル]**を選択します。テンプレートリストから、**[空白のアプリ(ユニバーサルWindows)]**を選択します。**[名前]**を入力し、**[OK]**をクリックしてプロジェクトを作成します。
3. ソリューションエクスプローラーでプロジェクト名を右クリックし、**[参照の追加]**を選択します。
4. **[参照マネージャー]**ダイアログボックスで**[ComponentOne for UWP]**を選択します。**[OK]**をクリックしてダイアログボックスを閉じ、参照を追加します。
5. **MainPage.xaml** をまだ開いていない場合は開き、`<Page>` タグ内に次のマークアップを追加します。

   XAML マークアップ
   ```
   xmlns:RichTextBox="using:C1.Xaml.RichTextBox"
   ```
   これにより、C1.Xaml.RichTextBox アセンブリへの参照がプロジェクトに追加されます。
6. プロジェクトの XAML ウィンドウで、カーソルを `<Grid x:Name="ContentPanel"></Grid>` タグの間に置き、1回クリックします。
7. `<Grid>` タグ内に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。このマークアップは、コントロールに名前を付けます。

   XAML マークアップ
   ```
   <RichTextBox:C1RichTextBox Name="C1RTB" />
   ```
8. 追加した C1RichTextBox コントロールのマークアップの下にカーソルを置きます。Visual Studio ツールボックスで C1RichTextBoxMenu コントロールを見つけてダブルクリックし、このコントロールをアプリケーションに追加します。
9. C1RichTextBoxMenu マークアップを次のサンプルのように編集します。これは、C1RichTextBoxMenu に名前を付け、これを C1RichTextBox コントロールに連結します。

   XAML マークアップ
   ```
   <RichTextBox:C1RichTextBoxMenu x:Name="rtbMenu" RichTextBox="{Binding ElementName=RTB1}" />
   ```

ここまでの成果

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て編集できます。次の手順では、アプリケーションをさらにカスタマイズします。

手順2：RichTextBox のカスタマイズ

前の手順では、新しい Windows ストアアプリケーションを作成し、アプリケーションをセットアップし、アプリケーションに C1RichTextBox コントロールを追加しました。この手順では、さらにアプリケーションをカスタマイズします。

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、**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。
2. 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。

Visual Basic

```
Imports C1.Xaml.RichTextBox
```

C#

```
using C1.Xaml.RichTextBox;
```

3. MainPage コンストラクタに次のコードを追加します。

Visual Basic

```
Me.C1RTB.FontSize = 24
Me.C1RTB.Text = "こんにちは！ UWPで有効である一番完全的なリッチ・テキストエディターへようこそ。
RichTextBox for
UWPを使用して、フォーマット済みのテキストをロード、編集やHTMLやRTFドキュメントとして保存することができます。
C1RichTextBox コントロールでリッチフォーマット、自動線折り返し、HTML と RTF のインポート・エクスポート、
テーブル、画像、注釈
など色々な機能が提供されています。"
```

C#

```
this.C1RTB.FontSize = 24;
this.C1RTB.Text = "こんにちは！ UWPで有効である一番完全的なリッチ・テキストエディターへようこそ。
RichTextBox for
UWPを使用して、フォーマット済みのテキストをロード、編集やHTMLやRTFドキュメントとして保存することができます。
C1RichTextBox コントロールでリッチフォーマット、自動線折り返し、HTML と RTF のインポート・エクスポート、
テーブル、画像、注釈
など色々な機能が提供されています。";
```

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。

ここまでの成果

この手順では、**C1RichTextBox**アプリケーションにコンテンツを追加しました。後はこれを実行するだけです。次の手順では、アプリケーションを実行し、**C1RichTextBox** コントロールの実行時に可能な操作をいくつか確認します。

手順3：アプリケーションの実行

前の手順では、新しい Windows ストアアプリケーションを作成し、**C1RichTextBox** 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しました。後は、アプリケーションを実行し、実行時に可能な操作をいくつか確認するだけです。

次の手順に従います。

1. メニューで[**デバッグ**]→[**デバッグ開始**]をクリックして、アプリケーションを実行します。
2. **C1RichTextBox** コントロール内をタップします。テキストを入力するためのオンスクリーンキーボードが表示されます。
3. オンスクリーンキーボードを使用して、**C1RichTextBox** 内にテキストを入力します。
4. "Welcome" などの単語を選択します。単語が強調表示されて選択されることを確認します。選択を変更する場合は、選択した単語の両側に表示されるハンドルを選択してドラッグします。たとえば、次の文の一部を選択します。

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 コントロールの使い方について少し学ぶことができました。このチュートリアルでは、新しい Windows ストアアプリケーションを作成し、**C1RichTextBox** コントロールを追加し、実行時に可能な操作をいくつか確認しました。

RichTextBox の使い方

RichTextBox for UWP は、UWP 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.Xaml.RichTextBox 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 クラスに相当します。WPF と同様に、**C1Document** はいくつかの要素（C1Block オブジェクト）を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。

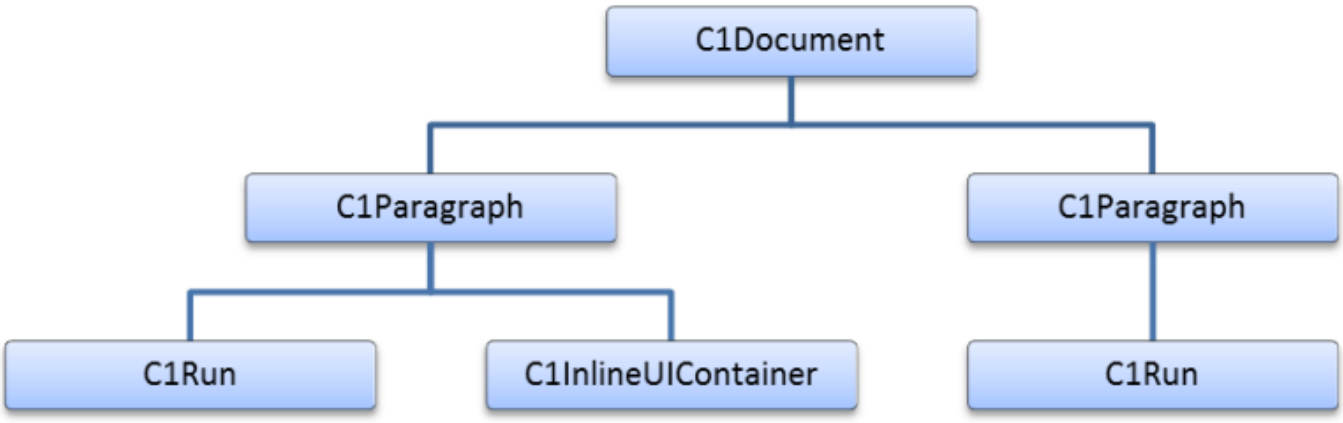
アプリケーションの多くはこの **C1RichTextBox** コントロールだけを処理し、このコントロールによってドキュメントは単純に線形的に表示されます。ドキュメント構造に完全にアクセスし、ドキュメントを直接作成したり管理するために、**C1Document** クラスが提供するリッチオブジェクトモデルを選択するアプリケーションもあります。

主要な概念と機能

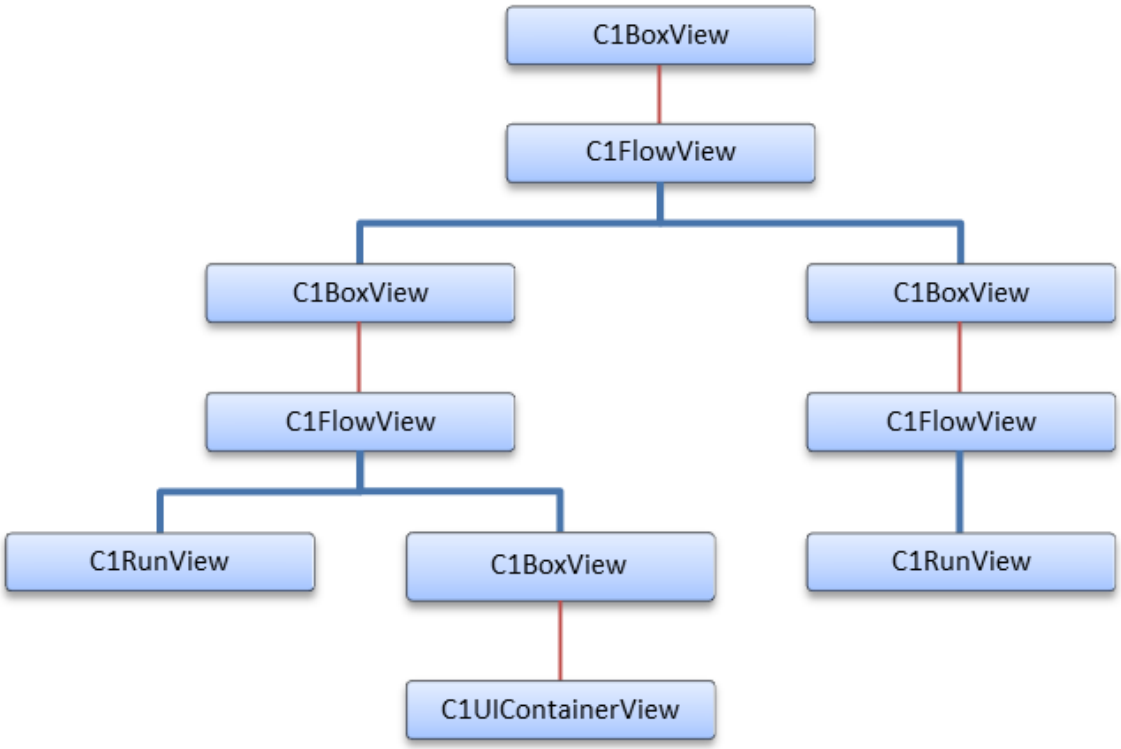
レイアウト情報へのアクセス

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並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。

C1TextElementView.Origin: ドキュメント座標によるビューの原点です。

C1TextElementView.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、C1TextElementView.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 C1TextElementView.Children コレクションに対応します。

ビューの構成は、**C1TextElementView.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、**C1TextElementView.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** オブジェクトには、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。

Visual Basic コードの書き方

Visual Basic

```
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# コードの書き方

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、C1RichTextViewManager.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 要素の行数だけ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。

テキストのコピー、切り取り、貼り付け

C1RichTextBox コントロールは、キーボード（CTRL+C、CTRL+V、CTRL+X）やタッチディスプレイの UI コマンドを使用して、プレーンテキストとリッチテキストの両方でクリップボードを完全にサポートします。**AppBar** またはその他の考えられるすべての方法で切り取り/コピー/貼り付けのコマンドを実装します。ClipboardMode プロパティを使用して、プレーンテキストとリッチテ

キストのどちらをコピー、切り取り、または貼り付けるかを示します。

button_Click イベントには次のコードがあります。

C#

```
private void btnCopy_Click(object sender, Windows.UI.Xaml.RoutedEventArgs e)
        {
            rtb.ClipboardCopy();
        }



private void btnCut_Click(object sender, Windows.UI.Xaml.RoutedEventArgs e)
        {
                rtb.ClipboardCopy();
        }



private void btnPaste_Click(object sender, Windows.UI.Xaml.RoutedEventArgs e)
        {
                rtb.ClipboardPaste();
            }
        }
```

これらのコードを使用して、コピー/切り取り/貼り付けのコマンドをすばやく簡単に実装することができます。

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。

Visual Basic コードの書き方

Visual Basic

```
Public Sub New()
    InitializeComponent()
    ' テキストを設定します
    rtb.Text = "ハイパリンクを含むサンプルテキストです。"
    ' 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.Always
    AddHandler _rtb.RequestNavigate, AddressOf _rtb_RequestNavigate
End Sub
```

C# コードの書き方

C#

```
public MainPage()
{
  InitializeComponent();
  // テキストを設定します
  rtb.Text = "ハイパリンクを含むサンプルテキストです。";
  // 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 = C1.Xaml.RichTextBox.NavigationMode.Always;
  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、EditExtensions.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 C1Hyperlink.NavigateUri プロパティに割り当てられる URI です。

その後、**C1RichTextBox.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。デフォルトでは、ハイパーリンクの上にマウスポインタを移動してタップすると、**C1RichTextBox.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。

C1RichTextBox.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。

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。C1RichTextBox.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。

Visual Basic コードの書き方

Visual Basic

```
Public Sub New()
    InitializeComponent()
    ' テキストを設定します
    rtb.Text = "ハイパリンクを含むサンプルテキストです。"
    ' 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.Always
    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# コードの書き方

C#

```
public MainPage()
        {
            this.InitializeComponent();
            // テキストを設定します
                rtb.Text = "ハイパリンクを含むサンプルテキストです。";
            // 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 = C1.Xaml.RichTextBox.NavigationMode.Always;
            rtb.RequestNavigate += rtb_RequestNavigate;
        }
        private async void rtb_RequestNavigate(object sender,
RequestNavigateEventArgs e)
        {
            var md = new MessageDialog("The document is requesting to navigate to " +
e.Hyperlink.NavigateUri, "Navigate");
            md.Commands.Add(new UICommand("OK", (UICommandInvokedHandler) =>
            {
                Windows.System.Launcher.LaunchUriAsync(e.Hyperlink.NavigateUri);
            }));
            md.Commands.Add(new UICommand("Cancel", (UICommandInvokedHandler) =>
            {
                rtb.Select(e.Hyperlink.ContentStart.TextOffset,
e.Hyperlink.ContentRange.Text.Length);
            }));
            await md.ShowAsync();
        }
```

C1RichTextBox.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。

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、ElementTapped イベントと C1RichTextBox.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、**C1RichTextBox.GetPositionFromPoint** メソッドが必要です。**C1RichTextBox.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 C1RichTextBox.RightTapped イベントに関連付けます。

C# コードの書き方

C#

```
public sealed partial class MainPage : Page
    {
        C1RichTextBox rtb;

        public MainPage()
        {
            this.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
            rtb.RightTapped += rtb_RightTapped;
        }
```

1つの単語の C1TextElement.FontWeight 値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 C1RichTextBox.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を示します。

C# コードの書き方

C#

```
void rtb_RightTapped(object sender, RightTappedRoutedEventArgs e)
{
   // 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.Weight ==
```

```
FontWeights.Bold.Weight
          ? FontWeights.Normal
          : FontWeights.Bold;
         }
      }

   }

}
```

C1TextElement.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。**C1TextElement.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 メソッドを使用すると、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。

印刷

C1RichTextBox for UWP では、Windows の標準印刷技術を使用して印刷を行うことができます。このコントロールではページレイアウトもサポートされているため、ユーザーはドキュメントがどのように印刷されるかを確認することができます。

C1RichTextBox での印刷の詳細については、「C1RichTextBox コンテンツの印刷」トピックを参照してください。

スペルチェック

RichTextBox for UWP は、入力中スペルチェックをサポートしています。スペルミスの単語は赤い波線の下線で強調表示されます。

ドキュメント内のスペルミスを右クリックするとメニューが表示され、エンドユーザーは修正候補を選択するなどのオプションを選択できます。スペルミスの単語を無視したり、それを辞書に追加することもできます。

スペルチェックは、**ComponentOne for UWP** に含まれる **C1SpellChecker** コンポーネントを使用して行います。

スペルチェックをアプリケーションに追加するには、次の4つの手順を実行します。

1. 新しい **C1SpellChecker** コントロールを宣言します。
2. **C1SpellChecker** を C1RichTextBox コントロールの SpellChecker プロパティに割り当てます。
3. リソースファイルへのストリームを取得します。
4. 任意の MainDictionary をロードします。

スペルチェックをアプリケーションに追加するには、タスク別ヘルプトピック「スペルチェックの追加」を参照してください。

辞書のサポート

RichTextBox for UWP では、カスタマイズ可能な 22 個の辞書がサポートされています。

スペルチェックの追加

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 C1RichTextBoxMenu コントロールがページに追加されていることを前提としています。

このヘルプトピックでは、英語の辞書リソースを使用します。別の辞書を使用する場合は、、サポートされている 22 個の辞書から選択することができます。

スペルチェックを追加するには、新しい **C1SpellChecker** を宣言し、任意のメイン辞書をロードして、**C1RichTextBox** の SpellChecker プロパティに **C1SpellChecker** を割り当てます。

1. **C1RichTextBox** と **C1RichTextBoxMenu** の各コントロールを次のように編集します。

XAML マークアップ

```
<c1RTB:C1RichTextBoxMenu x:Name="rtbMenu" RichTextBox="{Binding ElementName=rtb}"/>
  <c1RTB:C1RichTextBox x:Name="rtb"  BorderThickness="2" BorderBrush="DarkGray" />
```

2. ソリューションエクスプローラーで、アプリケーション名を右クリックし、**[追加]**→**[新しいフォルダ]**を選択します。フォルダ名を **Resources** と指定します。
3. ソリューションエクスプローラーで、**Resources** フォルダを右クリックし、**[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。
4. **[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。
   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。
5. プロパティウィンドウで、**C1Spell_en-US.dct** ファイルの**[ビルドアクション]**を**[埋め込まれたリソース]**に設定します。
6. **MainPage.xaml** ページを右クリックし、コンテキストメニューから**[コードの表示]**を選択します。
7. 次の import 文をページの先頭に追加します。

C#

```
using C1.Xaml.SpellChecker;
using C1.Xaml.RichTextBox;
```

8. **MainPage()** コンストラクタで Text プロパティを設定します。

C#

```
rtb.Text = @"Some facts about Mark Twain (spelling errors intentional ;-) A
steambat pilot neded a vast knowldege of the
ever-chaging river to be able to stop at any of the hundreds of ports and wood-
lots along the river banks. Twain
meticulosly studied 2,000 miles (3,200 km) of the Mississipi for more than two
years before he received his steamboat
pilot license in 1859.";
```

9. 新しい **C1SpellChecker** を宣言します。

C#

```
var spell = new C1SpellChecker();
```

10. **C1SpellChecker** を **C1RichTextBox** コントロールの **SpellChecker** プロパティに割り当てます。

C#

```
rtb.SpellChecker = spell;
```

11. リソースファイルへのストリームを取得します。示されている場所に、実際のアプリケーション名を挿入してください。

C#

```
Assembly asm = typeof(MainPage).GetTypeInfo().Assembly;
Stream stream =
asm.GetManifestResourceStream("YourApplicationName.Resources.C1Spell_en-
US.dct");
```

12. 任意の **MainDictionary** をロードします。

C#

```
spell.MainDictionary.Load(stream);
```

ここまでの成果

このトピックでは、**C1RichTextBox** コントロールにテキストを追加し、次にスペルチェックを処理するいくつかのコードを追加しました。

コンテンツの設定と書式設定

C1RichTextBox コントロールを使用して、さまざまな種類のコンテンツを作成、ロード、または保存できます。以下のトピックでは、**C1RichTextBox** で指定できるコンテンツの種類、コントロールでのハイパーリンクの作成、スペルチェック、およびいくつかの編集タスクの実行について説明します。

テキストコンテンツ

C1RichTextBox.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ブランクで、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。

設計時

C1RichTextBox.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、**C1RichTextBox.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、こんにちは!)を入力します。
   これで、**C1RichTextBox.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、**C1RichTextBox.Text** プロパティを設定するには、次に示すように `Text="こんにちは!"` を `<c1rtb:C1RichTextBox>` タグに追加します。

XAML マークアップ

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

コードの場合

たとえば、**C1RichTextBox.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。

Visual Basic

```
Me.C1RichTextBox1.Text = "こんにちは!"
```

C#

```
this.c1RichTextBox1.Text = "こんにちは!";
```

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「こんにちは!」(または指定したテキスト)が表示されることを確認しました。

HTML コンテンツ

C1RichTextBox.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ブランクで、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。

設計時

C1RichTextBox.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、**C1RichTextBox.Html** プロパティの隣にあるテキストボックスに "**<h1>こんにちは!</h1>**" などのテキストを入力します。
   これで、**C1RichTextBox.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、**C1RichTextBox.Html** プロパティを設定するには、次に示すように `Html="こんにちは!"` を `<c1rtb:C1RichTextBox>` タグに追加します。

XAML マークアップ

```
<c1rtb:C1RichTextBox HorizontalAlignment="Left" Margin="10,10,0,0" Name="C1RichTextBox1"
VerticalAlignment="Top" Height="83" Width="208" Html="&lt;h1&gt;Hello World!&lt;/h1&gt;" />
```

コードの場合

たとえば、**C1RichTextBox.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。

Visual Basic

```
Me.C1RichTextBox1.Html = "&lt;b&gt;こんにちは!&lt;/b&gt;"
```

C#

```
this.c1RichTextBox1.Html = "&lt;b&gt;こんにちは!&lt;/b&gt;";
```

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「こんにちは!」(または指定したテキスト)が大きなフォントで表示されることを確認しました。

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」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。

RTF コンテンツの設定

C1RichTextBox コントロールを使用して、リッチテキストフォーマットドキュメントの読み取り、編集、および表示を行うことができます。C1RichTextBox コントロール内の RTF コンテンツを設定するには、別途 RtfFilter クラスを使用する必要があります。このクラスは、RTF を、コントロールが理解できるドキュメントに変換します。

RtfFilter は、Microsoft WordPad がサポートする画像、フォント、テーブル、およびほとんどの書式設定に対応します。C1RichTextBox コントロールは、HTML の読み書きにも対応するため、RTF と HTM の相互変換も可能です。

C1RichTextBox コントロールに RTF 文字列をロードするには、**RtfFilter** クラスから **ConvertToDocument** メソッドを呼び出します。

C# コードの書き方

C#

```
string rtf = @"{\rtf1\ansi\ansicpg1252\deff0\deflang1033{\fonttbl{\f0\fnil\fcharset0
```

```
Calibri;}}
{\*\generator Msftedit
5.41.21.2510;}\viewkind4\uc1\pard\sa200\sl276\slmult1\lang9\f0\fs22 some rtf
content\par
}";
C1RichTextBox1.Document = new RtfFilter().ConvertToDocument(rtf);
```

C1RichTextBox のコンテンツを RTF として出力するには、ConvertFromDocument メソッドを呼び出します。

C#

```
string rtf = new RtfFilter().ConvertFromDocument(c1RichTextBox1.Document);
```

HtmlFilter のカスタム

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。

- **HtmlFilter.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。
- **HtmlFilter.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。
- **HtmlFilter.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。
- **HtmlFilter.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.Xaml.Imaging** 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、HtmlFilter.ConvertingHtmlNode イベントと HtmlFilter.ConvertingTextElement イベントの両方を使用します。

HtmlFilter.ConvertinHtmlNode

これは **HtmlFilter.ConvertingHtmlNode** イベントハンドラです。

Visual Basic コードの書き方

Visual Basic

```
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# コードの書き方

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 <img/> 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。

HtmlFilter.ConvertingTextElement

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

Visual Basic コードの書き方

Visual Basic

```
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# コードの書き方

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。

スタイルのオーバーライド

C1RichTextBox ドキュメントに変更を適用する方法は2つあります。**C1TextRange** を使用して基底のドキュメントの各部を変更する方法と、基底のドキュメントではなくビューだけを変更する方法です。ドキュメントではなくビューを変更する例としては、選択範囲を異なる前景色と背景色で強調表示します。このスタイルの変更は、ドキュメント自体ではなく現在のビューに属します。この方法は、構文の色指定や入力中スペルチェックにも見られます。

実際の動作は、**SyntaxHighlight** サンプルで確認できます。このサンプルは、マシン にインストールされています。

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、スタイルの変更を **C1TextRange** オブジェクトに適用する方法に比べて2つの利点があります。

- 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。

以下のコード例は、**SyntaxHighlight** サンプルからの抜粋です。

スタイルのオーバーライドを使用してみるには、まず、**C1RangeStyleCollection** オブジェクト、**C1RichTextBox** オブジェクト、および C1RichTextBoxMenu オブジェクトを宣言する必要があります。また、ドキュメントの色指定に使用するスタイルを初期化し、**Page_Loaded** イベントを作成します。このイベントで、C1RangeStyleCollection をコントロールの **StyleOverrides** コレクションに追加します。

C# コードの書き方

C#

```
public sealed partial class SyntaxHighlight : UserControl
    {
        C1RichTextBox _rtb;
        C1RichTextBoxMenu _menu;
        C1RangeStyleCollection _rangeStyles = new C1RangeStyleCollection();

        // HTML の解析に使用される正規表現を初期化します
        string tagPattern =
            @"</?(?<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スタイルを初期化します
        C1TextElementStyle brDarkBlue = new C1TextElementStyle
        {
            { C1TextElement.ForegroundProperty, new
SolidColorBrush(Color.FromArgb(255, 0, 0, 180)) }
        };
        C1TextElementStyle brDarkRed = new C1TextElementStyle
        {
            { C1TextElement.ForegroundProperty, new
SolidColorBrush(Color.FromArgb(255, 180, 0, 0)) }
        };
        C1TextElementStyle brLightRed = new C1TextElementStyle
        {
            { C1TextElement.ForegroundProperty, new SolidColorBrush(Colors.Red) }
        };

        public SyntaxHighlight()
        {
            InitializeComponent();

            Loaded += SyntaxHighlight_Loaded;
        }

        void SyntaxHighlight_Loaded(object sender, RoutedEventArgs e)
        {
```

```
            if (_rtb == null)
            {
                _rtb = new C1RichTextBox
                {
                    ReturnMode = ReturnMode.SoftLineBreak,
                    TextWrapping = TextWrapping.NoWrap,
                    IsReadOnly = false,
                    Document = new C1Document
                    {
                        Background = new SolidColorBrush(Colors.White),
                        FontFamily = new FontFamily("Courier New"),
                        FontSize = 16,
                        Blocks =
                    {
                        new C1Paragraph
                        {
                            Children =
                            {
                                new C1Run
                                {
                                    Text = GetStringResource("w3c.htm")
                                },
                            },
                        }
                    }
                    },
                    StyleOverrides = { _rangeStyles }
                };
                if (_menu == null)
                {
                    _menu = new C1RichTextBoxMenu();
                }

                LayoutRoot.Children.Add(_rtb);
                _menu.RichTextBox = _rtb;
                LayoutRoot.Children.Add(_menu);

                _rtb.TextChanged += tb_TextChanged;
                UpdateSyntaxColoring(_rtb.Document.ContentRange);
            }
        }
```

次に、TextChanged イベントを設定します。このイベントでは、ドキュメントの変更箇所を検出して、**UpdateSyntaxColoring** メソッドをトリガします。

C# コードの書き方

C#

```
void tb_TextChanged(object sender, C1TextChangedEventArgs e)
        {
            var start = e.Range.Start.Enumerate(LogicalDirection.Backward)
                                     .FirstOrDefault(p => p.Symbol.Equals('\n'));
            if (start != null)
```

```
            {
                start = start.GetPositionAtOffset(1);
            }
            var end = e.Range.End.Enumerate().FirstOrDefault(p =>
p.Symbol.Equals('\n'));
            var doc = e.Range.Start.Element.Root;
            UpdateSyntaxColoring(new C1TextRange(start ?? doc.ContentStart, end ??
doc.ContentEnd));
        }
```

UpdateSyntaxColoring メソッドは、1つのタグ全体を選択し、それに色指定することで、書式設定をビューに適用します。

C# コードの書き方

C#

```
    // 構文を色指定します
            void UpdateSyntaxColoring(C1TextRange range)
            {
                // 以前の色指定を削除します
                _rangeStyles.RemoveRange(range);

                var input = range.Text;

                // 一致箇所を強調表示します
                foreach (Match m in Regex.Matches(input, tagPattern))
                {
                    // タグ全体を選択して濃い青色にします
                    _rangeStyles.Add(new
C1RangeStyle(GetRangeAtTextOffset(range.Start, m), brDarkBlue));

                    // タグ名を選択して濃い赤色にします
                    var tagName = m.Groups["tagName"];
                    _rangeStyles.Add(new
C1RangeStyle(GetRangeAtTextOffset(range.Start, tagName), 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];
                            _rangeStyles.Add(new
C1RangeStyle(GetRangeAtTextOffset(range.Start, att), brLightRed));
                        }
                    }
                }
        }
```

最後の2つのメソッドは、オフセットに基づいて C1TextRange の開始と終了を取得し、サンプルに付属するリソースファイルを取得します。

C# コードの書き方

C#

```
     C1TextRange GetRangeAtTextOffset(C1TextPointer pos, Capture capture)
            {
                var start = pos.GetPositionAtOffset(capture.Index,
C1TextRange.TextTagFilter);
                var end = start.GetPositionAtOffset(capture.Length,
C1TextRange.TextTagFilter);
                return new C1TextRange(start, end);
            }

            // ユーティリティ
            static string GetStringResource(string resourceName)
            {
                Assembly asm = typeof(SyntaxHighlight).GetTypeInfo().Assembly;
                Stream stream =
asm.GetManifestResourceStream(String.Format("RichTextBoxSamples.Resources.{0}",
resourceName));
                using (var sr = new StreamReader(stream))
                {
                    return sr.ReadToEnd();
                }
            }
        }
}
```

一般に、このトピックに示した方法は、変更を基底のドキュメントに直接適用するより数千倍高速な優れたパフォーマンスを備えています。

HTMLのロードと保存

コントロールのコンテンツを永続化する際に C1RichTextBox コントロールの書式設定を維持するには、C1RichTextBox.Html プロパティを使用します。

C1RichTextBox.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、C1RichTextBox.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「こんにちは!」と入力すると、C1RichTextBox.Html プロパティは次のマークアップを返します。

XAML マークアップ

```
<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">こんにちは!</p>
</body>
```

```
</html>
```

C1RichTextBox.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。

C1RichTextBoxMenu の使い方

C1RichTextBoxMenu は、ラジアルコンテキストメニューとして機能します。このメニューを使用して、CSS やインラインマークアップとしてサポートされているフォント、装飾、サイズ、色などの基本的な HTML スタイル属性や RTF スタイル属性を含むテキストを編集したり、書式設定することができます。

C1RichTextBoxMenu をアプリケーションに追加する方法は簡単です。C1RichTextBox コントロールをアプリケーションに追加して名前を付けたら、**C1RichTextBoxMenu** を追加し、それを RichTextBox プロパティで C1RichTextBox コントロールに連結します。

XAML

```
<c1RTB:C1RichTextBoxMenu x:Name="rtbMenu" RichTextBox="{Binding ElementName=rtb}"/>
<c1RTB:C1RichTextBox x:Name="rtb"  BorderThickness="2" BorderBrush="DarkGray"
RequestNavigate="rtb_RequestNavigate" />
```

C1RichTextBoxMenu アセンブリの使用方法の詳細については、**ComponentOne** サンプルフォルダにインストールされた **DemoRichTextBox** サンプルを参照してください。

メニューとコマンド

RichTextBox for UWP を使用して、AppBar と Menu の両方をカスタマイズできます。これらの柔軟なツールを使用すると、ユーザーに完全なテキスト編集機能をオンデマンドで提供でき、貴重な RichTextBox 領域を広げることができます。

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();
```

貼り付け

C#

```
if (!rtb.IsReadOnly)
```

```
{
    rtb.ClipboardPaste();
}
```

切り取り

C#

```
if (rtb.IsReadOnly)
    rtb.ClipboardCopy();
else
{
    rtb.ClipboardCut();
}
```

テキスト揃え機能

次のコードスニペットは、テキスト揃えに使用されるコードを示しています。

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。

フォント系

C#

```
rtb.Selection.FontFamily = new FontFamily("Arial");
```

フォントサイズ

C#

```
rtb.Selection.TrimRuns();
foreach (var run in rtb.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 = new SolidColorBrush(Colors.Red);
```

強調表示色（背景）

C#

```
rtb.Selection.InlineBackground = new SolidColorBrush(Colors.Yellow);
```

太字の切り替え

C#

```
if (rtb.Selection.FontWeight != null && rtb.Selection.FontWeight.Value.Weight ==
FontWeights.Bold.Weight)
{
    rtb.Selection.FontWeight = FontWeights.Normal;
}
else
{
    rtb.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

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();
}
```

AppBar の使用

C1.Xaml.RichTextBox.AppBar ライブラリには、簡潔なコマンドバーの作成に使用できるツールが組み込まれています。組み込みツールは、Bold、Italic、Underline, Undo、Redo、Increase Font Size、Decrease Font Size、Center Align、Right Align、Left Align、Justify の各コマンドをサポートします。アプリケーションバーコントロールは不可視ですが、アセンブリに含まれる **C1 ツール** をアプリケーションバーに追加できます。

AppBar アセンブリをアプリケーションに実装する完全な例については、チュートリアル「AppBar アプリケーションの作成」を参照してください。

AppBar アセンブリに含まれる **C1 ツール**を使用するには、マークアップを次のようにする必要があります。これを終了タグ `</Grid>` の後に配置します。

XAML

```
<Page.BottomAppBar>
 <AppBar x:Name="topAppBar" Padding="10,0,10,0" >
 </AppBar>
</Page.TopAppBar>
```

AppBar がページの先頭に表示されるように設定することもできます。それには、次のマークアップを使用します。

XAML

```
<Page.TopAppBar>
 <AppBar x:Name="bottomAppBar" Padding="10,0,10,0" >
 </AppBar>
</Page.BottomAppBar>
```

使用する **C1 ツール**は、`<AppBar> </AppBar>` タグの間の StackPanel コントロール内に配置します。

XAML

```
<StackPanel Orientation="Horizontal" HorizontalAlignment="Left">
  <RichTextBox:C1BoldTool x:Name="btnBold" Style="{StaticResource BoldAppBarButtonStyle}" />
  <RichTextBox:C1ItalicTool x:Name="btnItalic" Style="{StaticResource ItalicAppBarButtonStyle}" />
  <RichTextBox:C1UnderlineTool x:Name="btnUnderline" Style="{StaticResource UnderlineAppBarButtonStyle}" />
</StackPanel>
```

汎用 Button コントロールを使用するには、次のコードを追加します。

XAML

```
<StackPanel Orientation="Horizontal" HorizontalAlignment="Left">
  <Button x:Name="btnCopy" Style="{StaticResource CopyAppBarButtonStyle}" Click="btnCopy_Click"/>
  <Button x:Name="btnPaste" Style="{StaticResource PasteAppBarButtonStyle}" Click="btnPaste_Click"/>
  <Button x:Name="btnCut" Style="{StaticResource CutAppBarButtonStyle}" Click="btnCut_Click"/>
</StackPanel>
```

これらの汎用 Button コントロールにはクリックイベントがあり、次のコードを追加して、クリックイベントを処理する必要があります。

C#

```
#region Clipboard
        private void btnCopy_Click(object sender, Windows.UI.Xaml.RoutedEventArgs e)
        {
            rtb.ClipboardCopy();
        }

        private void btnCut_Click(object sender, Windows.UI.Xaml.RoutedEventArgs e)
        {
            if (rtb.IsReadOnly)
                rtb.ClipboardCopy();
            else
                rtb.ClipboardCut();
        }

        private void btnPaste_Click(object sender, Windows.UI.Xaml.RoutedEventArgs e)
        {
            if (!rtb.IsReadOnly)
            {
                rtb.ClipboardPaste();
            }
        }
        #endregion
```

C1Document オブジェクトの使い方

C1Document 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。これまでは、C1Document オブジェクトの編集可能なビューとしての C1RichTextBox コントロールのオブジェクトモデルに注目していました。**C1Document** オブジェクト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。

C1Document はドキュメントの構造を公開し、ここで **C1RichTextBox** は、主にテキストをコントロールの線形的でフラットなビューとして処理します。**C1Document** のドキュメントモデルは、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。

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.Document 要素の定義

C1TextElement クラス

C1Document のすべての要素の基本クラスは、C1TextElement クラスです。**C1TextElement** クラスは、さらに、C1Block クラス、C1Document クラス、C1Inline クラスの3つのクラスで構成されます。

- **C1Block クラス**
  C1Block 要素は、すべてのブロックレベルのフローコンテンツ要素の基礎となる抽象クラスです。ブロックレベルのフローコンテンツ要素は、**C1Block** を継承する C1Paragraph などのクラスです。

  次の要素は、**C1Block** クラスを継承します。

要素	説明
C1BlockUIContainer	UIElement 要素をフローコンテンツに埋め込めるようにするブロックレベルのフローコンテンツ要素。
C1List	順序付きまたは順序なしリスト内にコンテンツを表示する機能を提供するブロックレベルのフロー要素。
C1ListItem	C1List 内の特定のコンテンツ項目を表すフローコンテンツ要素。
C1Paragraph	コンテンツを段落としてグループ化するために使用されるブロックレベルのフローコンテンツ要素。
C1Section	他の C1Block 要素をグループ化するために使用されるブロックレベルのフローコンテンツ要素。
C1Table	行と列で構成されるグリッドベースの表現を提供するブロックレベルのフローコンテンツ要素。
C1TableCell	C1Table 内の1つのセルコンテンツを定義するフローコンテンツ要素。
C1TableRow	C1Table 内の1つの行を定義するフローコンテンツ要素。
C1TableRowGroup	C1Table 内の C1TableRow 要素のグループ化に使用されるフローコンテンツ要素。

- **C1Document クラス**
  C1Document クラスは、1つのフロー要素を表します。
- **C1Inline クラス**
  C1Inline 抽象クラスは、すべてのインラインフロー要素の基礎となります。C1Inline クラスは、次の要素で構成されま

す。

C1Inline 要素	説明
C1InlineUIContainer	UIElement 要素をフローコンテンツに埋め込めるようにするインラインレベルのフローコンテンツ要素。
C1Run	書式設定された、または書式設定されていないテキストのランを保持するためのインラインレベルのフロー要素。
C1Span	C1Span クラスは、C1Hyperlink 要素などの他の C1Inline フローコンテンツ要素をグループ化します。

他の C1RichTextBox.Documents 要素

Documents 名前空間には、**C1TextElement** クラスに含まれない要素もあります。

- **C1HtmlDocument**
  この要素は、1つの HTML ドキュメントを表します。
- **C1TextRange**
  この要素は、C1Document 内の1つのテキスト範囲を表します。
- **C1TextPointer**
  この要素は、C1Document 内の位置を表します。

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.Xaml アセンブリと C1.Xaml.RichTextBox 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。

C# コードの書き方

C#

```
using C1.Xaml;
using C1.Xaml.RichTextBox;
using C1.Xaml.RichTextBox.Documents;
using System.Reflection;
using System.Text;
using Windows.UI;
using Windows.UI.Text;

// Blank Page 項目テンプレートは、http://msdn.microsoft.com/ja-
jp/library/windows/apps/xaml/hh768232.aspx で文書化されています。

namespace RTBTestsDoc
{
    /// <summary>
    /// 自身で使用したり、Frame 内で移動先になることができる空のページ。
    /// </summary>
    public sealed partial class MainPage : Page
    {
        // C1Document ドキュメントを表示する C1RichTextBox
        C1RichTextBox _rtb;
```

```
        public MainPage()
        {
            this.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).GetTypeInfo().Assembly);
          _rtb.IsReadOnly = true;

        }
```

このコードは、C1RichTextBox を作成し、その **C1RichTextBox.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。

C# コードの書き方

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");
            //Heading1 H = new Heading1("Assembly\r\n" + asm.FullName.Split(',')[0]);
            // アセンブリ
            Heading1 p = new Heading1();
            p.Inlines.Add(new C1Run() { Text = "Assembly\r\n" +
asm.FullName.Split(',')[0] });
            doc.Blocks.Add(p);

            // タイプ
            foreach (Type t in asm.ExportedTypes)
                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 **C1TextElement.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。

C# コードの書き方

C#

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

Heading 段落は、Normal を拡張してテキストを太字にします。

C# コードの書き方

C#

```
class Heading : Normal
        {
            public Heading()
            {
                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。

C# コードの書き方

C#

```
class Heading1 : Heading
        {
            public Heading1()
            {
                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()
            {
                this.FontSize = 18;
                this.FontStyle = FontStyle.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()
            {
                this.FontSize = 14;
                this.Background = new SolidColorBrush(Colors.LightGray);
                this.Padding = new Thickness(20, 3, 0, 0);
            }
        }
        class Heading4 : Heading
        {
            public Heading4()
            {
                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。

C# コードの書き方

C#

```
void DocumentType(C1Document doc, Type t)
        {

            // タイプ
            Heading2 h2 = new Heading2();
            h2.Inlines.Add(new C1Run() { Text = "Class " + t.Name });
            doc.Blocks.Add(h2);

            // プロパティ
            Heading3 h3 = new Heading3();
            h3.Inlines.Add(new C1Run() { Text = "Properties" });
            doc.Blocks.Add(h3);
            foreach (PropertyInfo pi in t.GetRuntimeProperties())
            {
                if (pi.DeclaringType == t)
                    DocumentProperty(doc, pi);
            }

            // メソッド
            h3 = new Heading3();
            h3.Inlines.Add(new C1Run() { Text = "Methods" });
            doc.Blocks.Add(h3);
            foreach (MethodInfo mi in t.GetRuntimeMethods())
            {
                if (mi.DeclaringType == t)
                    DocumentMethod(doc, mi);
            }
```

```
            // イベント
            h3 = new Heading3();
            h3.Inlines.Add(new C1Run() { Text = "Events" });
            doc.Blocks.Add(h3);
            foreach (EventInfo ei in t.GetRuntimeEvents())
            {
                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。

C# コードの書き方

C#

```
void DocumentProperty(C1Document doc, PropertyInfo pi)
        {
            if (pi.PropertyType.IsGenericParameter)
                return;

            Heading4 h4 = new Heading4();
            h4.Inlines.Add(new C1Run() { Text = pi.Name });
            doc.Blocks.Add(h4);

            var text = string.Format("public {0} {1} {{ {2}{3} }}",
              pi.PropertyType.Name,
              pi.Name,
              pi.CanRead ? "get; " : string.Empty,
              pi.CanWrite ? "set; " : string.Empty);
            Normal n = new Normal();
            n.Inlines.Add(new C1Run() { Text = text });
            doc.Blocks.Add(n);
        }
```

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。

C# コードの書き方

C#

```
 void DocumentMethod(C1Document doc, MethodInfo mi)
        {
            if (mi.IsSpecialName)
                return;

            Heading4 h4 = new Heading4();
            h4.Inlines.Add(new C1Run() { Text = mi.Name });
            doc.Blocks.Add(h4);
            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());

            Normal n = new Normal();
            n.Inlines.Add(new C1Run() { Text = text });
            doc.Blocks.Add(n);
        }

        void DocumentEvent(C1Document doc, EventInfo ei)
        {
            Heading4 h4 = new Heading4();
            h4.Inlines.Add(new C1Run() { Text = ei.Name });
            doc.Blocks.Add(h4);

            var text = string.Format("public {0} {1}",
              ei.EventHandlerType.Name,
              ei.Name);

            Normal n = new Normal();
            n.Inlines.Add(new C1Run() { Text = text });
            doc.Blocks.Add(n);
        }
    }
}
```

ここでプロジェクトを実行すると、次の図のようになります。

Assembly
C1.Xaml.RichTextBox

Class C1RichTextBox

Properties

LastFlowDirection
public FlowDirection LastFlowDirection { get; set; }
ContextMenu
public Object ContextMenu { get; set; }
TranslateY
public Double TranslateY { get; }
ViewManager
public C1RichTextViewManager ViewManager { get; }
DocumentHistory
public DocumentHistory DocumentHistory { get; }
Text
public String Text { get; set; }
Selection
public C1TextRange Selection { get; set; }

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 C1RichTextBox.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**
  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 **C1List.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 **C1Span.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 NavigateUri 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 **C1BlockUIContainer.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。

C# コードの書き方

C#

```
// 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 **C1RichTextBox.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、下部のコントロールが編集可能になっていることがわかります（**C1RichTextBox.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 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。**InitializeComponent()** メソッド内で **KeyDown** イベントを処理するだけです。

C# コードの書き方

C#

```
public MainPage()
        {
            this.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, KeyRoutedEventArgs e)
        {
            if (e.Key == VirtualKey.Control)
```

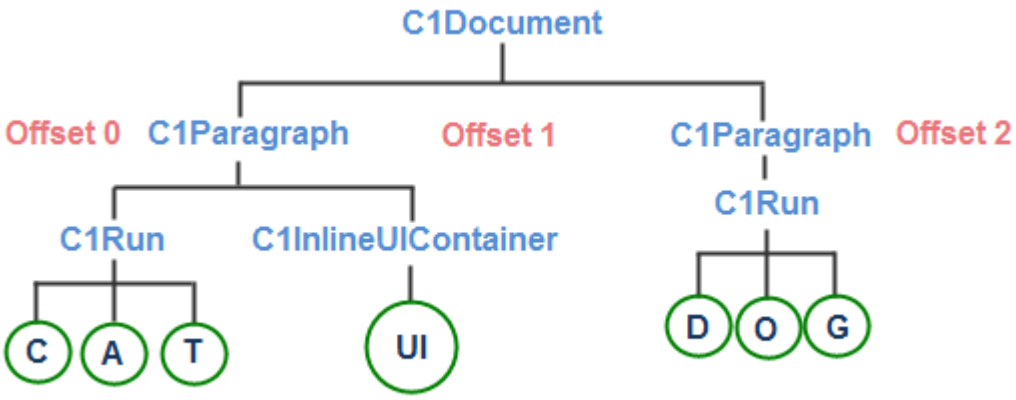
```
            {
                var h2 = _rtb.Document.Blocks.OfType<Heading2>().FirstOrDefault();
                if (h2 != null)
                    h2.ContentRange.ToUppercase();
            }
        }
```

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。

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、この図を例にして説明します。



上の図の青色のテキストのノードは、C1TextElement です。3つのオフセット位置が2つの **C1Paragraph** 要素の前、要素の間、および要素の後にマークされていることもわかります。マークされたオフセット位置は、C1Document 要素内の C1TextPointer の位置を示します。

C1Run 要素では、そのテキスト内の各文字がその要素の子と見なされ、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、その子の前と後の2つのオフセット位置があります。

ドキュメントを一連のシンボルとして可視化することもできます。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。そのため、上の図を XML で再作成すると、以下のようになります。

XAML

```
<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 に明確な順序を提供します。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、**C1TextPointer.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す。これは、**SyntaxHighlight** サンプルの次のコードでわかるように、単に指定された整数オフセットの位置を返します。

C#

```
C1TextRange GetRangeAtTextOffset(C1TextPointer pos, Capture capture)
        {
            var start = pos.GetPositionAtOffset(capture.Index,
C1TextRange.TextTagFilter);
            var end = start.GetPositionAtOffset(capture.Length,
C1TextRange.TextTagFilter);
            return new C1TextRange(start, end);
        }
```

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<C1TextPointer> 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。

C#

```
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。

Visual Basic コードの書き方

Visual Basic

```
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# コードの書き方

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ので、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。

Visual Basic

```
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。

Visual Basic コードの書き方

Visual Basic

```
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
    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# コードの書き方

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。

Visual Basic コードの書き方

Visual Basic

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

C# コードの書き方

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 **C1TextRange.Text** プロパティを割り当ててテキストを置換します。

サポート要素

HTML 要素

RichTextBox for UWP は、数多くの HTML 要素をサポートします。以下の表に、HTML 要素の名前をリストし、それぞれが **RichTextBox for UWP** でサポートされてい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	○
DIV	○
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 UWP は、数多くの HTML 属性をサポートします。以下の表に、HTML 属性の名前と要素をリストし、それぞれが **RichTextBox for UWP** でサポートされているかどうかを示します。

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	BASE	×
hreflang	A、LINK	×
hspace	APPLET、IMG、OBJECT	○
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	×
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	UI	×
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	×

CSS プロパティ

RichTextBox for UWP は、数多くの CSS プロパティをサポートします。以下の表に、CSS プロパティの名前とメディアグループをリストし、それぞれが **RichTextBox for UWP** でサポートされてい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	○	色と画像だけをサポートします。

borer-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-width	visual	○	
borer-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	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	○	
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。
windows	visual、paged	×	
width	visual	○	img 要素内のみ。
word-spacing	visual	×	
z-index	visual	×	

CSS セレクタ

RichTextBox for UWP は、いくつかの CSS セレクタをサポートします。以下の表に、CSS セレクタのパターンと CSS レベルをリストし、それぞれが **RichTextBox for UWP** でサポートされてい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	○
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	○
E + F	2	○
E ~ F	3	○

チュートリアル

AppBar アプリケーションの作成

このチュートリアルでは、C1.RichTextBox.AppBar.dll アセンブリを使用して、アプリケーションを作成します。上部と下部にカスタムアプリケーションバーを作成するためのマークアップとコードを追加します。コードを使用してコンテンツも追加します。このアプリケーションで使用するファイルは、**StandardStyles.xaml** と **simple.htm** の2つあります。

手順1：アプリケーションを作成する

この手順では、新しい Windows ストアアプリケーションを作成し、**C1RichTextBox** コントロールを追加します。さらに、上部と下部のアプリケーションバーを作成するためのマークアップを追加します。

1. **[ファイル]**→**[新規作成]**→**[プロジェクト]**を選択し、**[新しいプロジェクト]**ダイアログボックスを開きます。
    1. 右側のペインで **C#** の下にある**[Windows ストア]**を選択します。
    2. 左側のペインで**[新しいアプリケーション (XAML)]**を選択します。
    3. アプリケーションの名前を入力し、**[OK]**をクリックします。新しい空の Windows ストアアプリケーションが開きます。
2. ソリューションエクスプローラーで、**[参照]**ファイルを右クリックし、リストから**[参照の追加]**を選択します。次のアセンブリ参照を参照して選択します。
    - C1.Xaml
    - C1.Xaml.RichTextBox
    - C1.Xaml.RichTextBox.AppBar
    - C1.Xaml.RichTextBox.Menu
3. **MainPage.xaml** ファイルをダブルクリックして開きます。
4. ページ先頭の `<Page>` タグに次の名前空間宣言を追加します。

```
XAML
xmlns:RichTextBox="using:C1.Xaml.RichTextBox"
xmlns:Xaml="using:C1.Xaml"
```

`<Page>` タグは次のようになります。

```
XAML
<Page xmlns:RichTextBox="using:C1.Xaml.RichTextBox"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:AppBarTest3"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:Xaml="using:C1.Xaml"
    x:Class="AppBarTest3.MainPage"
    mc:Ignorable="d">
```

5. ページ内の `<Grid> </Grid>` タグのすぐ上に、`<Page.Resources>` を追加します。

```
XAML
<Page.Resources>
        <ResourceDictionary Source="Common/StandardStyles.xaml"/>
    </Page.Resources>
```

StandardStyles.xaml ファイルは**手順2**で追加します。

6. 次のマークアップを <Grid> </Grid> タグの間に追加して、いくつかの行定義を追加します。

XAML

```
<Grid.RowDefinitions>
  <RowDefinition Height="140"/>
  <RowDefinition Height="*"/>
</Grid.RowDefinitions>
```

7. 終了タグ </Grid.RowDefinition> の下にカーソルを置きます。Visual Studio ツールボックスを開き、**C1RichTextBox** コントロールを見つけます。このコントロールをダブルクリックしてページに追加します。
8. <RichTextBox:C1RichTextBox/> タグを次のように編集します。コントロールの名前と2つのイベントを追加します。

XAML

```
<RichTextBox:C1RichTextBox x:Name="rtb" Grid.Row="1"  Margin="116,0,100,100"
RequestNavigate="rtb_RequestNavigate" PointerPressed="rtb_PointerPressed" />
```

9. 終了タグ </Grid> をページで見つけ、そのタグのすぐ下にカーソルを置きます。次のマークアップを追加して、上部と下部の AppBar のフレームワークを作成します。

XAML

```
<Page.TopAppBar>
  <AppBar x:Name="topAppBar" IsSticky="True" Padding="10,0,10,0">
    <Grid>
      <StackPanel Orientation="Horizontal" HorizontalAlignment="Right">

      </StackPanel>
    </Grid>
  </AppBar>
</Page.TopAppBar>
<Page.BottomAppBar>
 <AppBar x:Name="bottomAppBar" IsSticky="True" Padding="10,0,10,0">
   <Grid>
     <StackPanel Orientation="Horizontal" HorizontalAlignment="Left">

     </StackPanel>
     <StackPanel Orientation="Horizontal" HorizontalAlignment="Right">

     </StackPanel>
   </Grid>
 </AppBar>
</Page.BottomAppBar>
```

10. カーソルを TopAppBar の <StackPanel> </StackPanel> タグの間に置きます。次のマークアップを追加して、3つの **C1RichTextBox** ツールを追加します。

XAML

```
<RichTextBox:C1LeftAlignTool x:Name="btnLeftAlign" Style="{StaticResource AlignLeftAppBarButtonStyle}"
/>
<RichTextBox:C1CenterAlignTool x:Name="btnCenterAlign" Style="{StaticResource
AlignCenterAppBarButtonStyle}" />
<RichTextBox:C1RightAlignTool x:Name="btnRightAlign" Style="{StaticResource
AlignRightAppBarButtonStyle}" />
```

11. 次に、BottomAppBar で最初の <StackPanel> </StackPanel> タグを探します。次のマークアップを追加して、3つの汎用 Button コントロールを追加します。

XAML

```
<Button x:Name="btnCopy" Style="{StaticResource CopyAppBarButtonStyle}" Click="btnCopy_Click"/>
<Button x:Name="btnPaste" Style="{StaticResource PasteAppBarButtonStyle}" Click="btnPaste_Click"/>
<Button x:Name="btnCut" Style="{StaticResource CutAppBarButtonStyle}" Click="btnCut_Click"/>
```

12. 2番目の `<StackPanel> </StackPanel>` タグを探します。次のマークアップを追加して、5つの **C1RichTextBox ツール**と1つの汎用 Button コントロールを追加します。

XAML

```
<RichTextBox:C1IncreaseFontSizeTool x:Name="btnIncreaseFontSize" Style="{StaticResource
FontIncreaseAppBarButtonStyle}"/>
<RichTextBox:C1DecreaseFontSizeTool x:Name="btnDecreaseFontSize" Style="{StaticResource
FontDecreaseAppBarButtonStyle}"/>
<RichTextBox:C1BoldTool x:Name="btnBold" Style="{StaticResource BoldAppBarButtonStyle}" />
<RichTextBox:C1ItalicTool x:Name="btnItalic" Style="{StaticResource ItalicAppBarButtonStyle}" />
<RichTextBox:C1UnderlineTool x:Name="btnUnderline" Style="{StaticResource UnderlineAppBarButtonStyle}"
/>
<Button x:Name="btnMore" Style="{StaticResource MoreAppBarButtonStyle}" Click="btnMore_Click"/>
```

この手順では、新しい Windows ストアアプリケーションを作成し、**C1RichTextBox** コントロールを追加しました。さらに、2つの **AppBar** のマークアップを追加し、いくつかの **C1 ツール**と汎用コントロールを **AppBar** のマークアップに追加しました。次の手順では、リソースファイルと汎用のアプリケーションコードを追加します。

手順2：リソースファイルと一般的なアプリケーションコードの追加

1. この手順では、スタイルを含むリソースファイルと、アプリケーションが使用するコンテンツを追加します。アプリケーション名を右クリックし、**[追加]**→**[新しいフォルダ]**を選択します。新しいフォルダ名を **Common** と指定します。
    1. **Common** フォルダを右クリックし、**[追加]**→**[新しい項目]**を選択します。
    2. **[新しい項目の追加]** ダイアログウィンドウの利用可能なテンプレートから**[空白のページ]**を選択します。これに、**StandardStyles.xaml** という名前を付け、**[OK]**をクリックします。
    3. ファイルが開いたら、次の XAML マークアップを追加します。

XAML でマークアップの書き方

XAML マークアップ

```
<ResourceDictionary
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
   <!-- Non-brush values that vary across themes -->
   <ResourceDictionary.ThemeDictionaries>
      <ResourceDictionary x:Key="Default">
         <x:String x:Key="BackButtonGlyph">&#xE071;</x:String>
         <x:String x:Key="BackButtonSnappedGlyph">&#xE0BA;</x:String>
      </ResourceDictionary>
      <ResourceDictionary x:Key="HighContrast">
         <x:String x:Key="BackButtonGlyph">&#xE071;</x:String>
         <x:String x:Key="BackButtonSnappedGlyph">&#xE0C4;</x:String>
      </ResourceDictionary>
   </ResourceDictionary.ThemeDictionaries>
   <x:String x:Key="ChevronGlyph">&#xE26B;</x:String>
   <!-- RichTextBlock styles -->
   <Style x:Key="BasicRichTextStyle" TargetType="RichTextBlock">
      <Setter Property="Foreground" Value="{ThemeResource ApplicationForegroundThemeBrush}"/>
```

```
        <Setter Property="FontSize" Value="{ThemeResource ControlContentThemeFontSize}"/>
        <Setter Property="FontFamily" Value="{ThemeResource ContentControlThemeFontFamily}"/>
        <Setter Property="TextTrimming" Value="WordEllipsis"/>
        <Setter Property="TextWrapping" Value="Wrap"/>
        <Setter Property="Typography.StylisticSet20" Value="True"/>
        <Setter Property="Typography.DiscretionaryLigatures" Value="True"/>
        <Setter Property="Typography.CaseSensitiveForms" Value="True"/>
    </Style>
    <Style x:Key="BaselineRichTextStyle" TargetType="RichTextBlock" BasedOn="{StaticResource
BasicRichTextStyle}">
        <Setter Property="LineHeight" Value="20"/>
        <Setter Property="LineStackingStrategy" Value="BlockLineHeight"/>
        <!-- Properly align text along its baseline -->
        <Setter Property="RenderTransform">
            <Setter.Value>
                <TranslateTransform X="-1" Y="4"/>
            </Setter.Value>
        </Setter>
    </Style>
    <Style x:Key="ItemRichTextStyle" TargetType="RichTextBlock" BasedOn="{StaticResource
BaselineRichTextStyle}"/>
    <Style x:Key="BodyRichTextStyle" TargetType="RichTextBlock" BasedOn="{StaticResource
BaselineRichTextStyle}">
        <Setter Property="FontWeight" Value="SemiLight"/>
    </Style>
    <!-- TextBlock styles -->
    <Style x:Key="BasicTextStyle" TargetType="TextBlock">
        <Setter Property="Foreground" Value="{ThemeResource ApplicationForegroundThemeBrush}"/>
        <Setter Property="FontSize" Value="{ThemeResource ControlContentThemeFontSize}"/>
        <Setter Property="FontFamily" Value="{ThemeResource ContentControlThemeFontFamily}"/>
        <Setter Property="TextTrimming" Value="WordEllipsis"/>
        <Setter Property="TextWrapping" Value="Wrap"/>
        <Setter Property="Typography.StylisticSet20" Value="True"/>
        <Setter Property="Typography.DiscretionaryLigatures" Value="True"/>
        <Setter Property="Typography.CaseSensitiveForms" Value="True"/>
    </Style>
    <Style x:Key="BaselineTextStyle" TargetType="TextBlock" BasedOn="{StaticResource
BasicTextStyle}">
        <Setter Property="LineHeight" Value="20"/>
        <Setter Property="LineStackingStrategy" Value="BlockLineHeight"/>
        <!-- Properly align text along its baseline -->
        <Setter Property="RenderTransform">
            <Setter.Value>
                <TranslateTransform X="-1" Y="4"/>
            </Setter.Value>
        </Setter>
    </Style>
    <Style x:Key="HeaderTextStyle" TargetType="TextBlock" BasedOn="{StaticResource
BaselineTextStyle}">
        <Setter Property="FontSize" Value="56"/>
        <Setter Property="FontWeight" Value="Light"/>
        <Setter Property="LineHeight" Value="40"/>
```

```
        <Setter Property="RenderTransform">
            <Setter.Value>
                <TranslateTransform X="-2" Y="8"/>
            </Setter.Value>
        </Setter>
    </Style>
    <Style x:Key="SubheaderTextStyle" TargetType="TextBlock" BasedOn="{StaticResource
BaselineTextStyle}">
        <Setter Property="FontSize" Value="26.667"/>
        <Setter Property="FontWeight" Value="Light"/>
        <Setter Property="LineHeight" Value="30"/>
        <Setter Property="RenderTransform">
            <Setter.Value>
                <TranslateTransform X="-1" Y="6"/>
            </Setter.Value>
        </Setter>
    </Style>
    <Style x:Key="TitleTextStyle" TargetType="TextBlock" BasedOn="{StaticResource
BaselineTextStyle}">
        <Setter Property="FontWeight" Value="SemiBold"/>
    </Style>
    <Style x:Key="SubtitleTextStyle" TargetType="TextBlock" BasedOn="{StaticResource
BaselineTextStyle}">
        <Setter Property="FontWeight" Value="Normal"/>
    </Style>
    <Style x:Key="ItemTextStyle" TargetType="TextBlock" BasedOn="{StaticResource
BaselineTextStyle}"/>
    <Style x:Key="BodyTextStyle" TargetType="TextBlock" BasedOn="{StaticResource
BaselineTextStyle}">
        <Setter Property="FontWeight" Value="SemiLight"/>
    </Style>
    <Style x:Key="CaptionTextStyle" TargetType="TextBlock" BasedOn="{StaticResource
BaselineTextStyle}">
        <Setter Property="FontSize" Value="12"/>
        <Setter Property="Foreground" Value="{ThemeResource
ApplicationSecondaryForegroundThemeBrush}"/>
    </Style>
    <Style x:Key="GroupHeaderTextStyle" TargetType="TextBlock">
        <Setter Property="FontFamily" Value="{ThemeResource ContentControlThemeFontFamily}"/>
        <Setter Property="TextTrimming" Value="WordEllipsis"/>
        <Setter Property="TextWrapping" Value="NoWrap"/>
        <Setter Property="Typography.StylisticSet20" Value="True"/>
        <Setter Property="Typography.DiscretionaryLigatures" Value="True"/>
        <Setter Property="Typography.CaseSensitiveForms" Value="True"/>
        <Setter Property="FontSize" Value="26.667"/>
        <Setter Property="LineStackingStrategy" Value="BlockLineHeight"/>
        <Setter Property="FontWeight" Value="Light"/>
        <Setter Property="LineHeight" Value="30"/>
        <Setter Property="RenderTransform">
            <Setter.Value>
                <TranslateTransform X="-1" Y="6"/>
            </Setter.Value>
```

```
        </Setter>
    </Style>
    <!-- Button styles -->
    <!--
        TextButtonStyle is used to style a Button using subheader-styled text with no other adornment.
There
        are two styles that are based on TextButtonStyle (TextPrimaryButtonStyle and
TextSecondaryButtonStyle)
        which are used in the GroupedItemsPage as a group header and in the FileOpenPickerPage for
triggering
        commands.
    -->
    <Style x:Key="TextButtonStyle" TargetType="ButtonBase">
        <Setter Property="MinWidth" Value="0"/>
        <Setter Property="MinHeight" Value="0"/>
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="ButtonBase">
                    <Grid Background="Transparent">
                        <ContentPresenter x:Name="Text" Content="{TemplateBinding Content}" />
                        <Rectangle
                            x:Name="FocusVisualWhite"
                            IsHitTestVisible="False"
                            Stroke="{ThemeResource FocusVisualWhiteStrokeThemeBrush}"
                            StrokeEndLineCap="Square"
                            StrokeDashArray="1,1"
                            Opacity="0"
                            StrokeDashOffset="1.5"/>
                        <Rectangle
                            x:Name="FocusVisualBlack"
                            IsHitTestVisible="False"
                            Stroke="{ThemeResource FocusVisualBlackStrokeThemeBrush}"
                            StrokeEndLineCap="Square"
                            StrokeDashArray="1,1"
                            Opacity="0"
                            StrokeDashOffset="0.5"/>
                        <VisualStateManager.VisualStateGroups>
                            <VisualStateGroup x:Name="CommonStates">
                                <VisualState x:Name="Normal"/>
                                <VisualState x:Name="PointerOver">
                                    <Storyboard>
                                        <ObjectAnimationUsingKeyFrames Storyboard.TargetName="Text"
Storyboard.TargetProperty="Foreground">
                                            <DiscreteObjectKeyFrame KeyTime="0" Value="{ThemeResource
ApplicationPointerOverForegroundThemeBrush}"/>
                                        </ObjectAnimationUsingKeyFrames>
                                    </Storyboard>
                                </VisualState>
                                <VisualState x:Name="Pressed">
                                    <Storyboard>
                                        <ObjectAnimationUsingKeyFrames Storyboard.TargetName="Text"
Storyboard.TargetProperty="Foreground">
```

```
                                <DiscreteObjectKeyFrame KeyTime="0" Value="{ThemeResource
ApplicationPressedForegroundThemeBrush}"/>
                            </ObjectAnimationUsingKeyFrames>
                        </Storyboard>
                    </VisualState>
                    <VisualState x:Name="Disabled">
                        <Storyboard>
                            <ObjectAnimationUsingKeyFrames Storyboard.TargetName="Text"
Storyboard.TargetProperty="Foreground">
                                <DiscreteObjectKeyFrame KeyTime="0" Value="{ThemeResource
ApplicationPressedForegroundThemeBrush}"/>
                            </ObjectAnimationUsingKeyFrames>
                        </Storyboard>
                    </VisualState>
                </VisualStateGroup>
                <VisualStateGroup x:Name="FocusStates">
                    <VisualState x:Name="Focused">
                        <Storyboard>
                            <DoubleAnimation Duration="0" To="1"
Storyboard.TargetName="FocusVisualWhite" Storyboard.TargetProperty="Opacity"/>
                            <DoubleAnimation Duration="0" To="1"
Storyboard.TargetName="FocusVisualBlack" Storyboard.TargetProperty="Opacity"/>
                        </Storyboard>
                    </VisualState>
                    <VisualState x:Name="Unfocused"/>
                </VisualStateGroup>
                <VisualStateGroup x:Name="CheckStates">
                    <VisualState x:Name="Checked"/>
                    <VisualState x:Name="Unchecked">
                        <Storyboard>
                            <ObjectAnimationUsingKeyFrames Storyboard.TargetName="Text"
Storyboard.TargetProperty="Foreground">
                                <DiscreteObjectKeyFrame KeyTime="0" Value="{ThemeResource
ApplicationSecondaryForegroundThemeBrush}"/>
                            </ObjectAnimationUsingKeyFrames>
                        </Storyboard>
                    </VisualState>
                    <VisualState x:Name="Indeterminate"/>
                </VisualStateGroup>
            </VisualStateManager.VisualStateGroups>
        </Grid>
    </ControlTemplate>
  </Setter.Value>
 </Setter>
</Style>
<Style x:Key="MenuTextButtonStyle" TargetType="ButtonBase">
 <Setter Property="MinWidth" Value="0"/>
 <Setter Property="MinHeight" Value="0"/>
 <Setter Property="Template">
  <Setter.Value>
   <ControlTemplate TargetType="ButtonBase">
    <Grid Background="Transparent">
```

```
                <ContentPresenter x:Name="Text" Content="{TemplateBinding Content}" />
                <Rectangle
                  x:Name="FocusVisualWhite"
                  IsHitTestVisible="False"
                  Stroke="{ThemeResource FocusVisualWhiteStrokeThemeBrush}"
                  StrokeEndLineCap="Square"
                  StrokeDashArray="1,1"
                  Opacity="0"
                  StrokeDashOffset="1.5"/>
                <Rectangle
                  x:Name="FocusVisualBlack"
                  IsHitTestVisible="False"
                  Stroke="{ThemeResource FocusVisualBlackStrokeThemeBrush}"
                  StrokeEndLineCap="Square"
                  StrokeDashArray="1,1"
                  Opacity="0"
                  StrokeDashOffset="0.5"/>
                <VisualStateManager.VisualStateGroups>
                  <VisualStateGroup x:Name="CommonStates">
                    <VisualState x:Name="Normal"/>
                    <VisualState x:Name="PointerOver">
                      <Storyboard>
                        <ObjectAnimationUsingKeyFrames Storyboard.TargetName="Text"
Storyboard.TargetProperty="Foreground">
                          <DiscreteObjectKeyFrame KeyTime="0" Value="{ThemeResource
ApplicationPointerOverForegroundThemeBrush}"/>
                        </ObjectAnimationUsingKeyFrames>
                      </Storyboard>
                    </VisualState>
                    <VisualState x:Name="Pressed">
                      <Storyboard>
                        <ObjectAnimationUsingKeyFrames Storyboard.TargetName="Text"
Storyboard.TargetProperty="Foreground">
                          <DiscreteObjectKeyFrame KeyTime="0" Value="{ThemeResource
ApplicationPressedForegroundThemeBrush}"/>
                        </ObjectAnimationUsingKeyFrames>
                      </Storyboard>
                    </VisualState>
                    <VisualState x:Name="Disabled">
                      <Storyboard>
                        <ObjectAnimationUsingKeyFrames Storyboard.TargetName="Text"
Storyboard.TargetProperty="Foreground">
                          <DiscreteObjectKeyFrame KeyTime="0" Value="{ThemeResource
ApplicationPressedForegroundThemeBrush}"/>
                        </ObjectAnimationUsingKeyFrames>
                      </Storyboard>
                    </VisualState>
                  </VisualStateGroup>
                  <VisualStateGroup x:Name="FocusStates">
                    <VisualState x:Name="Focused">
                      <Storyboard>
                        <DoubleAnimation Duration="0" To="1"
```

```
Storyboard.TargetName="FocusVisualWhite" Storyboard.TargetProperty="Opacity"/>
                                <DoubleAnimation Duration="0" To="1"
Storyboard.TargetName="FocusVisualBlack" Storyboard.TargetProperty="Opacity"/>
                            </Storyboard>
                        </VisualState>
                        <VisualState x:Name="Unfocused"/>
                    </VisualStateGroup>
                    <VisualStateGroup x:Name="CheckStates">
                        <VisualState x:Name="Checked"/>
                        <VisualState x:Name="Unchecked">
                            <Storyboard>
                                <ObjectAnimationUsingKeyFrames Storyboard.TargetName="Text"
Storyboard.TargetProperty="Foreground">
                                    <DiscreteObjectKeyFrame KeyTime="0" Value="{ThemeResource
ApplicationSecondaryForegroundThemeBrush}"/>
                                </ObjectAnimationUsingKeyFrames>
                            </Storyboard>
                        </VisualState>
                        <VisualState x:Name="Indeterminate"/>
                    </VisualStateGroup>
                </VisualStateManager.VisualStateGroups>
            </Grid>
        </ControlTemplate>
    </Setter.Value>
  </Setter>
 </Style>
 <Style x:Key="TextPrimaryButtonStyle" TargetType="ButtonBase" BasedOn="{StaticResource
TextButtonStyle}">
    <Setter Property="Foreground" Value="{ThemeResource
ApplicationHeaderForegroundThemeBrush}"/>
 </Style>
 <Style x:Key="TextSecondaryButtonStyle" TargetType="ButtonBase" BasedOn="{StaticResource
TextButtonStyle}">
    <Setter Property="Foreground" Value="{ThemeResource
ApplicationSecondaryForegroundThemeBrush}"/>
 </Style>
 <!--
    TextRadioButtonStyle is used to style a RadioButton using subheader-styled text with no other
adornment.
    This style is used in the SearchResultsPage to allow selection among filters.
 -->
 <Style x:Key="TextRadioButtonStyle" TargetType="RadioButton" BasedOn="{StaticResource
TextButtonStyle}">
    <Setter Property="Margin" Value="0,0,30,0"/>
 </Style>
 <!--
    AppBarButtonStyle is used to style a Button (or ToggleButton) for use in an App Bar.  Content will
be centered
    and should fit within the 40 pixel radius glyph provided.  16-point Segoe UI Symbol is used for
content text
    to simplify the use of glyphs from that font.  AutomationProperties.Name is used for the text
below the glyph.
```

```
    -->
    <Style x:Key="AppBarButtonStyle" TargetType="ButtonBase">
        <Setter Property="Foreground" Value="{ThemeResource
AppBarItemForegroundThemeBrush}"/>
        <Setter Property="VerticalAlignment" Value="Stretch"/>
        <Setter Property="FontFamily" Value="Segoe UI Symbol"/>
        <Setter Property="FontWeight" Value="Normal"/>
        <Setter Property="FontSize" Value="20"/>
        <Setter Property="AutomationProperties.ItemType" Value="App Bar Button"/>
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="ButtonBase">
                    <Grid x:Name="RootGrid" Width="100" Background="Transparent">
                        <StackPanel VerticalAlignment="Top" Margin="0,12,0,11">
                            <Grid Width="40" Height="40" Margin="0,0,0,5" HorizontalAlignment="Center">
                                <TextBlock x:Name="BackgroundGlyph" Text="&#xE0A8;" FontFamily="Segoe UI
Symbol" FontSize="53.333" Margin="-4,-19,0,0" Foreground="{ThemeResource
AppBarItemBackgroundThemeBrush}"/>
                                <TextBlock x:Name="OutlineGlyph" Text="&#xE0A7;" FontFamily="Segoe UI
Symbol" FontSize="53.333" Margin="-4,-19,0,0"/>
                                <ContentPresenter x:Name="Content" HorizontalAlignment="Center" Margin="-
1,-1,0,0" VerticalAlignment="Center"/>
                            </Grid>
                            <TextBlock
                                x:Name="TextLabel"
                                Text="{TemplateBinding AutomationProperties.Name}"
                                Foreground="{ThemeResource AppBarItemForegroundThemeBrush}"
                                Margin="0,0,2,0"
                                FontSize="12"
                                TextAlignment="Center"
                                Width="88"
                                MaxHeight="32"
                                TextTrimming="WordEllipsis"
                                Style="{StaticResource BasicTextStyle}"/>
                        </StackPanel>
                        <Rectangle
                            x:Name="FocusVisualWhite"
                            IsHitTestVisible="False"
                            Stroke="{ThemeResource FocusVisualWhiteStrokeThemeBrush}"
                            StrokeEndLineCap="Square"
                            StrokeDashArray="1,1"
                            Opacity="0"
                            StrokeDashOffset="1.5"/>
                        <Rectangle
                            x:Name="FocusVisualBlack"
                            IsHitTestVisible="False"
                            Stroke="{ThemeResource FocusVisualBlackStrokeThemeBrush}"
                            StrokeEndLineCap="Square"
                            StrokeDashArray="1,1"
                            Opacity="0"
                            StrokeDashOffset="0.5"/>
                        <VisualStateManager.VisualStateGroups>
```

```
                    <VisualStateGroup x:Name="ApplicationViewStates">
                        <VisualState x:Name="FullScreenLandscape"/>
                        <VisualState x:Name="Filled"/>
                        <VisualState x:Name="FullScreenPortrait">
                            <Storyboard>
                                <ObjectAnimationUsingKeyFrames Storyboard.TargetName="TextLabel"
Storyboard.TargetProperty="Visibility">
                                    <DiscreteObjectKeyFrame KeyTime="0" Value="Collapsed"/>
                                </ObjectAnimationUsingKeyFrames>
                                <ObjectAnimationUsingKeyFrames Storyboard.TargetName="RootGrid"
Storyboard.TargetProperty="Width">
                                    <DiscreteObjectKeyFrame KeyTime="0" Value="60"/>
                                </ObjectAnimationUsingKeyFrames>
                            </Storyboard>
                        </VisualState>
                        <VisualState x:Name="Snapped">
                            <Storyboard>
                                <ObjectAnimationUsingKeyFrames Storyboard.TargetName="TextLabel"
Storyboard.TargetProperty="Visibility">
                                    <DiscreteObjectKeyFrame KeyTime="0" Value="Collapsed"/>
                                </ObjectAnimationUsingKeyFrames>
                                <ObjectAnimationUsingKeyFrames Storyboard.TargetName="RootGrid"
Storyboard.TargetProperty="Width">
                                    <DiscreteObjectKeyFrame KeyTime="0" Value="60"/>
                                </ObjectAnimationUsingKeyFrames>
                            </Storyboard>
                        </VisualState>
                    </VisualStateGroup>
                    <VisualStateGroup x:Name="CommonStates">
                        <VisualState x:Name="Normal"/>
                        <VisualState x:Name="PointerOver">
                            <Storyboard>
                                <ObjectAnimationUsingKeyFrames
Storyboard.TargetName="BackgroundGlyph" Storyboard.TargetProperty="Foreground">
                                    <DiscreteObjectKeyFrame KeyTime="0" Value="{ThemeResource
AppBarItemPointerOverBackgroundThemeBrush}"/>
                                </ObjectAnimationUsingKeyFrames>
                                <ObjectAnimationUsingKeyFrames Storyboard.TargetName="Content"
Storyboard.TargetProperty="Foreground">
                                    <DiscreteObjectKeyFrame KeyTime="0" Value="{ThemeResource
AppBarItemPointerOverForegroundThemeBrush}"/>
                                </ObjectAnimationUsingKeyFrames>
                            </Storyboard>
                        </VisualState>
                        <VisualState x:Name="Pressed">
                            <Storyboard>
                                <ObjectAnimationUsingKeyFrames Storyboard.TargetName="OutlineGlyph"
Storyboard.TargetProperty="Foreground">
                                    <DiscreteObjectKeyFrame KeyTime="0" Value="{ThemeResource
AppBarItemForegroundThemeBrush}"/>
                                </ObjectAnimationUsingKeyFrames>
                                <ObjectAnimationUsingKeyFrames
```

```
Storyboard.TargetName="BackgroundGlyph" Storyboard.TargetProperty="Foreground">
                                <DiscreteObjectKeyFrame KeyTime="0" Value="{ThemeResource
AppBarItemForegroundThemeBrush}"/>
                            </ObjectAnimationUsingKeyFrames>
                            <ObjectAnimationUsingKeyFrames Storyboard.TargetName="Content"
Storyboard.TargetProperty="Foreground">
                                <DiscreteObjectKeyFrame KeyTime="0" Value="{ThemeResource
AppBarItemPressedForegroundThemeBrush}"/>
                            </ObjectAnimationUsingKeyFrames>
                        </Storyboard>
                    </VisualState>
                    <VisualState x:Name="Disabled">
                        <Storyboard>
                            <ObjectAnimationUsingKeyFrames Storyboard.TargetName="OutlineGlyph"
Storyboard.TargetProperty="Foreground">
                                <DiscreteObjectKeyFrame KeyTime="0" Value="{ThemeResource
AppBarItemDisabledForegroundThemeBrush}"/>
                            </ObjectAnimationUsingKeyFrames>
                            <ObjectAnimationUsingKeyFrames Storyboard.TargetName="Content"
Storyboard.TargetProperty="Foreground">
                                <DiscreteObjectKeyFrame KeyTime="0" Value="{ThemeResource
AppBarItemDisabledForegroundThemeBrush}"/>
                            </ObjectAnimationUsingKeyFrames>
                            <ObjectAnimationUsingKeyFrames Storyboard.TargetName="TextLabel"
Storyboard.TargetProperty="Foreground">
                                <DiscreteObjectKeyFrame KeyTime="0" Value="{ThemeResource
AppBarItemDisabledForegroundThemeBrush}"/>
                            </ObjectAnimationUsingKeyFrames>
                        </Storyboard>
                    </VisualState>
                </VisualStateGroup>
                <VisualStateGroup x:Name="FocusStates">
                    <VisualState x:Name="Focused">
                        <Storyboard>
                            <DoubleAnimation
                                Storyboard.TargetName="FocusVisualWhite"
                                Storyboard.TargetProperty="Opacity"
                                To="1"
                                Duration="0"/>
                            <DoubleAnimation
                                Storyboard.TargetName="FocusVisualBlack"
                                Storyboard.TargetProperty="Opacity"
                                To="1"
                                Duration="0"/>
                        </Storyboard>
                    </VisualState>
                    <VisualState x:Name="Unfocused" />
                    <VisualState x:Name="PointerFocused" />
                </VisualStateGroup>
                <VisualStateGroup x:Name="CheckStates">
                    <VisualState x:Name="Checked">
                        <Storyboard>
```

```
                            <DoubleAnimation Duration="0" To="0"
Storyboard.TargetName="OutlineGlyph" Storyboard.TargetProperty="Opacity"/>
                            <ObjectAnimationUsingKeyFrames
Storyboard.TargetName="BackgroundGlyph" Storyboard.TargetProperty="Foreground">
                              <DiscreteObjectKeyFrame KeyTime="0" Value="{ThemeResource
AppBarItemForegroundThemeBrush}"/>
                            </ObjectAnimationUsingKeyFrames>
                            <ObjectAnimationUsingKeyFrames
Storyboard.TargetName="BackgroundCheckedGlyph" Storyboard.TargetProperty="Visibility">
                              <DiscreteObjectKeyFrame KeyTime="0" Value="Visible"/>
                            </ObjectAnimationUsingKeyFrames>
                            <ObjectAnimationUsingKeyFrames Storyboard.TargetName="Content"
Storyboard.TargetProperty="Foreground">
                              <DiscreteObjectKeyFrame KeyTime="0" Value="{ThemeResource
AppBarItemPressedForegroundThemeBrush}"/>
                            </ObjectAnimationUsingKeyFrames>
                          </Storyboard>
                        </VisualState>
                        <VisualState x:Name="Unchecked"/>
                        <VisualState x:Name="Indeterminate"/>
                      </VisualStateGroup>
                    </VisualStateManager.VisualStateGroups>
                  </Grid>
                </ControlTemplate>
              </Setter.Value>
            </Setter>
          </Style>
          <!--
            Standard AppBarButton Styles for use with Button and ToggleButton

            An AppBarButton Style is provided for each of the glyphs in the Segoe UI Symbol font.
            Uncomment any style you reference (as not all may be required).
          -->
          <Style x:Key="CopyAppBarButtonStyle" TargetType="ButtonBase" BasedOn="{StaticResource
AppBarButtonStyle}">
            <Setter Property="AutomationProperties.AutomationId" Value="CopyAppBarButton"/>
            <Setter Property="AutomationProperties.Name" Value="Copy"/>
            <Setter Property="Content" Value="&#xE16F;"/>
          </Style>
          <Style x:Key="PasteAppBarButtonStyle" TargetType="ButtonBase" BasedOn="{StaticResource
AppBarButtonStyle}">
            <Setter Property="AutomationProperties.AutomationId" Value="PasteAppBarButton"/>
            <Setter Property="AutomationProperties.Name" Value="Paste"/>
            <Setter Property="Content" Value="&#xE16D;"/>
          </Style>
          <Style x:Key="CutAppBarButtonStyle" TargetType="ButtonBase" BasedOn="{StaticResource
AppBarButtonStyle}">
            <Setter Property="AutomationProperties.AutomationId" Value="CutAppBarButton"/>
            <Setter Property="AutomationProperties.Name" Value="Cut"/>
            <Setter Property="Content" Value="&#xE16B;"/>
          </Style>
          <Style x:Key="UnderlineAppBarButtonStyle" TargetType="ButtonBase" BasedOn="{StaticResource
```

```
AppBarButtonStyle}">
        <Setter Property="AutomationProperties.AutomationId" Value="UnderlineAppBarButton"/>
        <Setter Property="AutomationProperties.Name" Value="Underline"/>
        <Setter Property="Content" Value="&#xE19A;"/>
    </Style>
    <Style x:Key="BoldAppBarButtonStyle" TargetType="ButtonBase" BasedOn="{StaticResource
AppBarButtonStyle}">
        <Setter Property="AutomationProperties.AutomationId" Value="BoldAppBarButton"/>
        <Setter Property="AutomationProperties.Name" Value="Bold"/>
        <Setter Property="Content" Value="&#xE19B;"/>
    </Style>
    <Style x:Key="ItalicAppBarButtonStyle" TargetType="ButtonBase" BasedOn="{StaticResource
AppBarButtonStyle}">
        <Setter Property="AutomationProperties.AutomationId" Value="ItalicAppBarButton"/>
        <Setter Property="AutomationProperties.Name" Value="Italic"/>
        <Setter Property="Content" Value="&#xE199;"/>
    </Style>
    <Style x:Key="MoreAppBarButtonStyle" TargetType="ButtonBase" BasedOn="{StaticResource
AppBarButtonStyle}">
        <Setter Property="AutomationProperties.AutomationId" Value="MoreAppBarButton"/>
        <Setter Property="AutomationProperties.Name" Value="More"/>
        <Setter Property="Content" Value="&#xE10C;"/>
    </Style>
    <Style x:Key="RedoAppBarButtonStyle" TargetType="ButtonBase" BasedOn="{StaticResource
AppBarButtonStyle}">
        <Setter Property="AutomationProperties.AutomationId" <Setter
Property="AutomationProperties.Name" Value="Align Right"/>
        <Setter Property="Content" Value="&#xE1A0;"/>
    </Style>
</ResourceDictionary>
```

2. アプリケーション名を再度右クリックし、**[追加]**→**[新しいフォルダ]**を選択します。新しいフォルダ名を **Resources** と指定します。
    1. **Resources** フォルダを右クリックし、**[追加]**→**[新しい項目]**を選択します。
    2. **[新しい項目の追加]** ダイアログウィンドウの利用可能なテンプレートから**[HTML ページ]**を選択します。新しいファイルに **simple.htm** と名前を付けます。
    3. 空白ページが開いたら、次のマークアップを追加します。

    XAML でマークアップの書き方

    XAML マークアップ

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<body style="font-size:medium">
    <p>
        Display and edit formatted text as HTML documents with <strong> RichTextBox<span
style="font-size:x-small"><sup>TM</sup></span> for WinRT XAML</strong>. Use the control to
display HTML content from the Web or use it as a rich text editor. The
<strong>C1RichTextBox</strong> control supports:
    </p>
    <ul style="line-height:1.5em">
        <li>
             Most text <span style="background-color: yellow"><strong>decorations</strong>
```

```
</span>, <span style="color: red"><em>alignments</em></span> and <u>styles</u>
      </li>
      <li>
         Bulleted and numbered lists
      </li>
      <li>
         Clipboard and document history (undo and redo)
      </li>
      <li>
         Hyperlinks: <a href="http://www.componentone.com">ComponentOne website</a>
      </li>
      <li>
         Insert tables and pictures
      </li>
    </ul>
</body>
</html>
```

4. ソリューションエクスプローラーで **simple.htm** を選択します。[プロパティ]ウィンドウで、**[ビルドアクション]**を**[埋め込みリソース]**に設定します。
5. アプリケーションをリビルドします。

3. ソリューションエクスプローラーで **App.xaml** ファイルをダブルクリックし、次のマークアップを追加します。

XAML

```
<Application.Resources>
  <ResourceDictionary>
    <ResourceDictionary.MergedDictionaries>
     <!-- Styles that define common aspects of the platform look and feel required by Visual Studio project
and item templates -->
          <ResourceDictionary Source="Common/StandardStyles.xaml"/>
        </ResourceDictionary.MergedDictionaries>
      </ResourceDictionary>
    </Application.Resources>
```

この手順では、アプリケーションリソースのコンテンツを追加および設定し、これらのリソースをアプリケーション全体に適用するためのマークアップを追加しました。次の手順では、いくつかの **AppBar** 項目の汎用コードと、最初の手順で追加した **C1RichTextBox** イベントを処理する汎用コードを追加します。

手順3：一般的なアプリケーションコードの追加

この手順では、いくつかの AppBar 項目を作成するコードと、C1RichTextBox コントロールのイベントを処理するコードを追加します。

1. MainPage.xaml を再度開き、ページを右クリックします。コンテキストメニューから[コードの表示]を選択します。
2. 次の using 文をページの先頭に追加します。

C# コードの書き方

C#

```
using C1.Xaml;
using C1.Xaml.RichTextBox;
using C1.Xaml.RichTextBox.Documents;
using System.Reflection;
```

```
using Windows.UI.Text;
using Windows.UI;
using Windows.UI.Popups;
```

3. **InitializeComponent()** メソッドのすぐ下に、**InitMorePopup()** メソッドを追加します。

```
C#
this.InitMorePopup();
```

4. 次に、C1RichTextBox コントロールを8つの C1 ツールで更新するためのコードを追加します。

C# コードの書き方

```
C#
btnBold.RichTextBox = rtb;
btnItalic.RichTextBox = rtb;
btnUnderline.RichTextBox = rtb;
btnIncreaseFontSize.RichTextBox = rtb;
btnDecreaseFontSize.RichTextBox = rtb;
btnLeftAlign.RichTextBox = rtb;
btnCenterAlign.RichTextBox = rtb;
btnRightAlign.RichTextBox = rtb;
```

5. ここで、コンテンツリソースをロードするコードを追加します。コードの **"YourApplicationName.Resources.simple.htm":** の箇所には、実際のアプリケーション名を挿入することを忘れないでください。

C# コードの書き方

```
C#
Assembly asm = typeof(MainPage).GetTypeInfo().Assembly;
Stream stream =
asm.GetManifestResourceStream("YourApplicationName.Resources.simple.htm");
var html = new StreamReader(stream).ReadToEnd();
rtb.Html = html;
```

6. 最後に、RequestNavigate イベントのイベントハンドラを追加します。

C# コードの書き方

```
C#
private async void rtb_RequestNavigate(object sender, RequestNavigateEventArgs
e)
        {
            var md = new MessageDialog("The document is requesting to navigate
to " + e.Hyperlink.NavigateUri, "Navigate");

            md.Commands.Add(new UICommand("OK", (UICommandInvokedHandler) =>
            {
                Windows.System.Launcher.LaunchUriAsync(e.Hyperlink.NavigateUri);
            }));

            md.Commands.Add(new UICommand("Cancel", (UICommandInvokedHandler) =>
            {
                rtb.Select(e.Hyperlink.ContentStart.TextOffset,
e.Hyperlink.ContentRange.Text.Length);
```

```
            }));

            await md.ShowAsync();
```

この手順では、C1RichTextBox イベントを処理するコードと、AppBar で使用するいくつかの **C1 ツール**を作成するコードを追加しました。次の手順では、下部の AppBar のコードを追加します。

手順4：BottomAppBar 用コードの追加

この手順では、**BottomAppBar** 項目のフライアウトイベントとクリックイベントを処理するコードを追加します。

1. 最初にページに追加するコードセクションには、クリップボード、元に戻す/やり直す、リスト、書式のクリア、下付き/上付き文字に対応する5つの領域があります。

   C# コードの書き方

C#

```
#region Clipboard
        private void btnCopy_Click(object sender,
Windows.UI.Xaml.RoutedEventArgs e)
        {
            rtb.ClipboardCopy();
        }

        private void btnCut_Click(object sender, Windows.UI.Xaml.RoutedEventArgs
e)
        {
            if (rtb.IsReadOnly)
                rtb.ClipboardCopy();
            else
                rtb.ClipboardCut();
        }

        private void btnPaste_Click(object sender,
Windows.UI.Xaml.RoutedEventArgs e)
        {
            if (!rtb.IsReadOnly)
            {
                rtb.ClipboardPaste();
            }
        }
        #endregion

        #region Undo/Redo
        void btnRedo_Click(object sender, RoutedEventArgs e)
        {
            if (rtb.DocumentHistory.CanRedo)
            {
                rtb.DocumentHistory.Redo();
            }
            morePopUp.Hide();
        }
```

```
        void btnUndo_Click(object sender, RoutedEventArgs e)
        {
            if (rtb.DocumentHistory.CanUndo)
            {
                rtb.DocumentHistory.Undo();
            }
            morePopUp.Hide();
        }
        #endregion

        #region Lists
        void btnBulletedList_Click(object sender, RoutedEventArgs e)
        {
            // 選択範囲が既にリストかどうかをチェックします
            if (rtb.Selection.Lists.Count<C1.Xaml.RichTextBox.Documents.C1List>
() > 0)
            {
                // リストを解除します
                rtb.Selection.UndoList();
            }
            else
            {
                // 箇条書きリストを作成します

rtb.Selection.MakeList(C1.Xaml.RichTextBox.Documents.TextMarkerStyle.Disc);
            }
            morePopUp.Hide();
        }

        void btnNumberedList_Click(object sender, RoutedEventArgs e)
        {
            // 選択範囲が既にリストかどうかをチェックします
            if (rtb.Selection.Lists.Count<C1.Xaml.RichTextBox.Documents.C1List>
() > 0)
            {
                // リストを解除します
                rtb.Selection.UndoList();
            }
            else
            {
                // 番号付きリストを作成します

rtb.Selection.MakeList(C1.Xaml.RichTextBox.Documents.TextMarkerStyle.Decimal);
            }
            morePopUp.Hide();
        }
        #endregion

        #region clear formatting
        void btnClear_Click(object sender, RoutedEventArgs e)
        {
            // 前景色と背景色をクリアします
```

```
            rtb.Selection.InlineBackground = null;
            rtb.Selection.Foreground = rtb.Foreground;

            // フォントをクリアします
            rtb.Selection.FontWeight = FontWeights.Normal;
            rtb.Selection.FontStyle = FontStyle.Normal;
            rtb.Selection.TextDecorations = null;
        }
        #endregion

        #region sub/super script
        void btnSubscript_Click(object sender, RoutedEventArgs e)
        {
            // 下付き文字
            if (rtb.Selection.InlineAlignment != C1VerticalAlignment.Sub &&
rtb.Selection.InlineAlignment != null)
            {
                rtb.Selection.InlineAlignment = C1VerticalAlignment.Sub;
                ShrinkFont(4);
            }
            else
            {
                rtb.Selection.InlineAlignment = C1VerticalAlignment.Baseline;
                GrowFont(4);
            }
            morePopUp.Hide();
        }

        void btnSuperscript_Click(object sender, RoutedEventArgs e)
        {
            // 上付き文字
            if (rtb.Selection.InlineAlignment != C1VerticalAlignment.Super &&
rtb.Selection.InlineAlignment != null)
            {
                rtb.Selection.InlineAlignment = C1VerticalAlignment.Super;
                ShrinkFont(4);
            }
            else
            {
                rtb.Selection.InlineAlignment = C1VerticalAlignment.Baseline;
                GrowFont(4);
            }
            morePopUp.Hide();
        }

        private void GrowFont(int size)
        {
            // フォント拡大
            rtb.Selection.TrimRuns();
            foreach (var run in rtb.Selection.Runs)
            {
                run.FontSize += size;
```

```
            }
        }

        private void ShrinkFont(int size)
        {
            // フォント縮小
            rtb.Selection.TrimRuns();
            foreach (var run in rtb.Selection.Runs)
            {
                run.FontSize -= size;
            }
        }
        #endregion
```

2. 次に追加するコード領域には、[その他]ボタンとメニューのイベントがあります。

C# コードの書き方

C#

```
#region More
        Flyout morePopUp = new Flyout();
        private void btnMore_Click(object sender, RoutedEventArgs e)
        {
            morePopUp.Placement = FlyoutPlacementMode.Top;
            morePopUp.ShowAt(sender as FrameworkElement);
        }

        void InitMorePopup()
        {
            Border menuBorder = new Border();
            menuBorder.Height = 260;
            menuBorder.Width = 150;
            StackPanel panel = new StackPanel();
            panel.VerticalAlignment = Windows.UI.Xaml.VerticalAlignment.Center;

            // 箇条書きリスト
            Button btnBulletedList = new Button();
            btnBulletedList.Content = "Bulleted List";
            btnBulletedList.Style =
Application.Current.Resources["MenuTextButtonStyle"] as Style;
            btnBulletedList.Margin = new Thickness(20, 5, 20, 5);
            btnBulletedList.Click += btnBulletedList_Click;

            // 番号付きリスト
            Button btnNumberedList = new Button();
            btnNumberedList.Content = "Numbered List";
            btnNumberedList.Style =
Application.Current.Resources["MenuTextButtonStyle"] as Style;
            btnNumberedList.Margin = new Thickness(20, 5, 20, 5);
            btnNumberedList.Click += btnNumberedList_Click;

            // 元に戻す
            Button btnUndo = new Button();
```

```
            btnUndo.Content = "Undo";
            btnUndo.Style = Application.Current.Resources["MenuTextButtonStyle"]
as Style;
            btnUndo.Margin = new Thickness(20, 5, 20, 5);
            btnUndo.Click += btnUndo_Click;

            // 元に戻す
            Button btnRedo = new Button();
            btnRedo.Content = "Redo";
            btnRedo.Style = Application.Current.Resources["MenuTextButtonStyle"]
as Style;
            btnRedo.Margin = new Thickness(20, 5, 20, 5);
            btnRedo.Click += btnRedo_Click;

            // 書式のクリア
            Button btnClear = new Button();
            btnClear.Content = "Clear Formatting";
            btnClear.Style =
Application.Current.Resources["MenuTextButtonStyle"] as Style;
            btnClear.Margin = new Thickness(20, 5, 20, 5);
            btnClear.Click += btnClear_Click;

            // 上付き文字
            Button btnSuperscript = new Button();
            btnSuperscript.Content = "Superscript";
            btnSuperscript.Style =
Application.Current.Resources["MenuTextButtonStyle"] as Style;
            btnSuperscript.Margin = new Thickness(20, 5, 20, 5);
            btnSuperscript.Click += btnSuperscript_Click;

            // 下付き文字
            Button btnSubscript = new Button();
            btnSubscript.Content = "Subscript";
            btnSubscript.Style =
Application.Current.Resources["MenuTextButtonStyle"] as Style;
            btnSubscript.Margin = new Thickness(20, 5, 20, 5);
            btnSubscript.Click += btnSubscript_Click;

            // 取り消し線
            Button btnStrikethrough = new Button();
            btnStrikethrough.Content = "Strikethrough";
            btnStrikethrough.Style =
Application.Current.Resources["MenuTextButtonStyle"] as Style;
            btnStrikethrough.Margin = new Thickness(20, 5, 20, 5);
            btnStrikethrough.Click += btnStrikethrough_Click;

            panel.Children.Add(btnBulletedList);
            panel.Children.Add(btnNumberedList);
            panel.Children.Add(btnSubscript);
            panel.Children.Add(btnSuperscript);
            panel.Children.Add(btnStrikethrough);
            panel.Children.Add(btnUndo);
            panel.Children.Add(btnRedo);
```

```
            panel.Children.Add(btnClear);
            menuBorder.Child = panel;
            morePopUp.Content = menuBorder;

        }

        void btnStrikethrough_Click(object sender, RoutedEventArgs e)
        {
            // 取り消し線
            var range = rtb.Selection;
            var collection = new C1TextDecorationCollection();
            if (range.TextDecorations == null)
            {
                collection.Add(C1TextDecorations.Strikethrough[0]);
            }
            else if
(!range.TextDecorations.Contains(C1TextDecorations.Strikethrough[0]))
            {
                foreach (var decoration in range.TextDecorations)
                    collection.Add(decoration);

                collection.Add(C1TextDecorations.Strikethrough[0]);
            }
            else
            {
                foreach (var decoration in range.TextDecorations)
                    collection.Add(decoration);

                collection.Remove(C1TextDecorations.Strikethrough[0]);
                if (collection.Count == 0)
                    collection = null;
            }
            range.TextDecorations = collection;
            morePopUp.Hide();
        }
        #endregion
```

3. 最後のコードセクションでは、PointerPressed イベントを処理します。

C# コードの書き方

C#

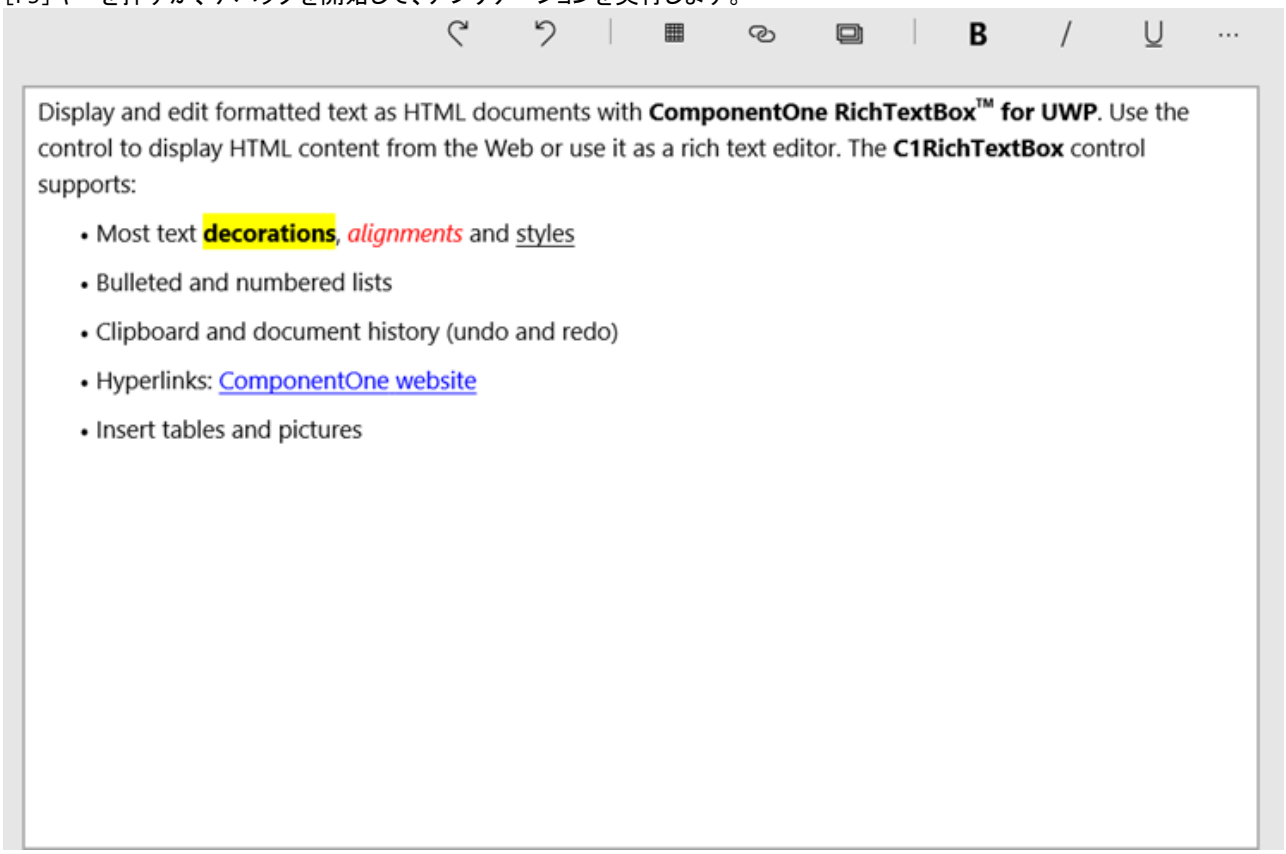
```
private void rtb_PointerPressed(object sender, PointerRoutedEventArgs e)
        {
            bottomAppBar.IsOpen = true;
            topAppBar.IsOpen = true;
        }
    }
}
```

この手順では、**下部の AppBar** のイベントコードを追加しました。次の手順では、このアプリケーションを実行します。

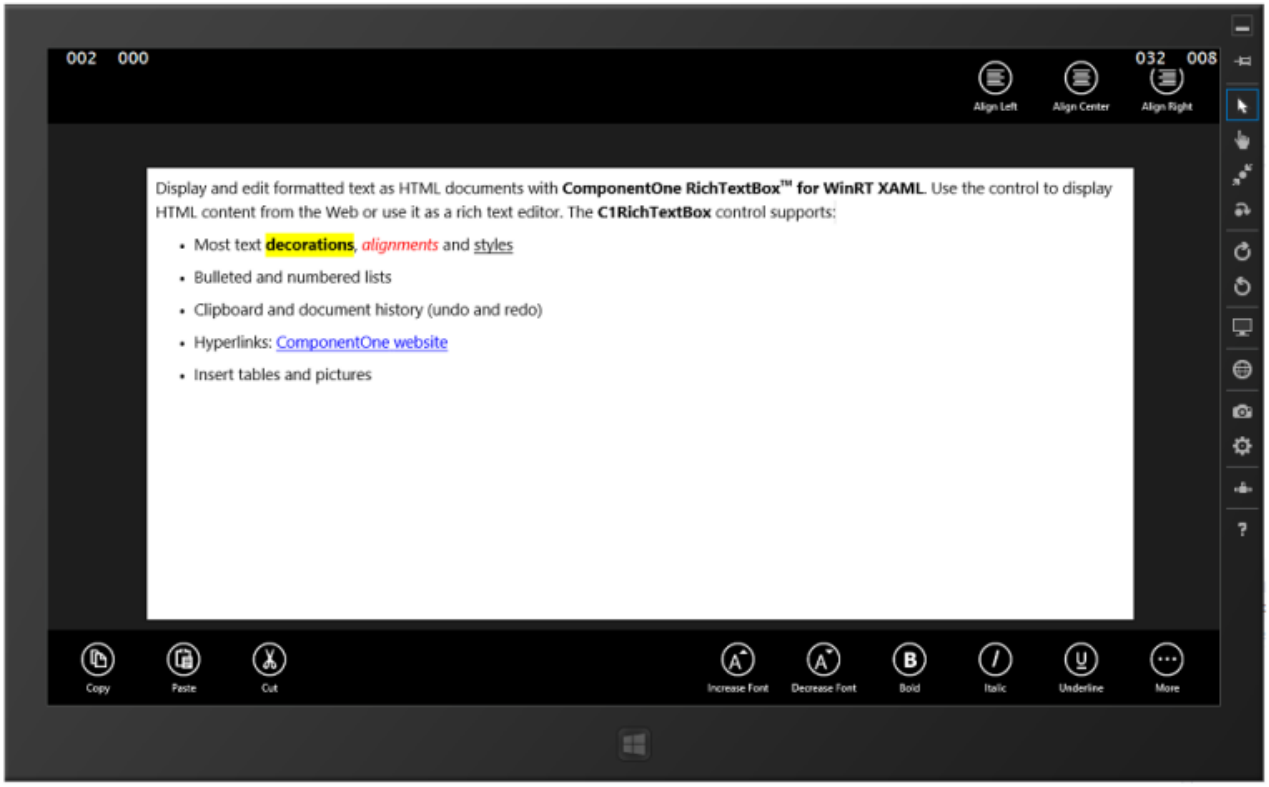
手順5：アプリケーションの実行

この手順では、アプリケーションを実行します。

1. [F5]キーを押すか、デバッグを開始して、アプリケーションを実行します。



2. **C1RichTextBox** テキスト領域内でタップまたはクリックします。追加した2つの AppBar が表示されます。



3. 用意されたコントロールを使用して、**C1RichTextBox** 内のテキストの書式を再設定できます。

ここまでの成果

おめでとうございます。これで、「**AppBar アプリケーションの作成**」チュートリアルは終了です。このチュートリアルでは、新しい Windows ストアアプリケーションを作成し、マークアップとコードを追加し、リソースファイルを追加しました。

C1RichTextBox コンテンツの印刷

このチュートリアルでは、新しい Windows ストアアプリケーションを作成し、**C1RichTextBox** コントロールに印刷機能を追加します。このチュートリアルは、次の **C1RichTextBox サンプル**フォルダにあるサンプルに基づいています。

手順1：アプリケーションの設定

この手順では、新しい Windows ストアアプリケーションを作成し、**C1RichTextBox** コントロールを追加します。さらに、**C1RichTextBox** コントロールと C1RichTextBoxMenu コントロールを作成するためのマークアップを追加します。

1. **[ファイル]**→**[新規作成]**→**[プロジェクト]**を選択し、**[新しいプロジェクト]**ダイアログボックスを開きます。
    1. 右側のペインで C# の下にある[Windows ストア]を選択します。
    2. 左側のペインで**[新しいアプリケーション (XAML)]**を選択します。
    3. アプリケーションの名前を入力し、**[OK]**をクリックします。新しい空の Windows ストアアプリケーションが開きます。
2. ソリューションエクスプローラーで、**[参照]**ファイルを右クリックし、リストから**[参照の追加]**を選択します。次のアセンブリ参照を参照して選択します。
    - C1.Xaml
    - C1.Xaml.RichTextBox
    - C1.Xaml.RichTextBox.Menu
3. **MainPage.xaml** ファイルをダブルクリックして開きます。
4. ページ先頭の `<Page>` タグに次の名前空間宣言を追加します。

XAML マークアップ

```
xmlns:c1RTB="using:C1.Xaml.RichTextBox"
```

5. 次に、いくつかの **Grid.Resources** と **RowDefinitions** を追加します。次のマークアップは、`<Grid> </Grid>` タグの間に置く必要があります。

XAML でマークアップの書き方

XAML マークアップ

```
<Grid.Resources>
        <DataTemplate x:Name="printTemplate">
          <Grid Height="{Binding ViewManager.PresenterInfo.Height}" Width="{Binding
ViewManager.PresenterInfo.Width}">
            <c1RTB:C1RichTextPresenter Source="{Binding}" Margin="{Binding
ViewManager.PresenterInfo.Padding}" />
          </Grid>
        </DataTemplate>
    </Grid.Resources>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition/>
    </Grid.RowDefinitions>
```

6. 終了タグ `</Grid.RowDefinitions>` のすぐ下に、汎用の Button コントロールを追加します。次のようにマークアップを編集します。名前、コンテンツ、およびクリックイベントを追加します。

XAML マークアップ

```
<Button x:Name="btnPrint" Content="Print" HorizontalAlignment="Left" Click="btnPrint_Click" />
```

7. 最後に、C1RichTextBoxMenu コントロールと C1RichTextBox コントロールを Visual Studio ツールボックスで見つけ、それらのコントロールをダブルクリックして追加します。マークアップを次のように編集します。

XAML でマークアップの書き方

XAML マークアップ

```
<c1RTB:C1RichTextBoxMenu x:Name="rtbMenu" RichTextBox="{Binding ElementName=rtb}" Grid.Row="1"
/>
<c1RTB:C1RichTextBox x:Name="rtb"  Grid.Row="1"
                     FontFamily="Times New Roman"
                     FontSize="20"
                     ViewMode="Print"
                     HorizontalContentAlignment="Center"
                     Background="#EEEEEE"/>
```

この手順では、新しい Windows ストアアプリケーションを設定し、アプリケーションに適切な参照を追加し、アプリケーションに **C1RichTextBox** コントロールを追加しました。次の手順では、適切なリソースファイルとアプリケーションコードを追加します。

手順2：リソースファイルとコードの追加

前の手順では、アプリケーションを設定し、コントロールを追加しました。この手順では、必要なリソースファイルを追加し、印刷を制御するコードを追加します。この手順では、**ComponentOne for UWP サンプル** と共にインストールされたリソースファイルを追加します。

1. ソリューションエクスプローラーでアプリケーション名を右クリックし、コンテキストメニューから**[追加]**→**[新しいフォルダ]**を選択します。新しいフォルダ名を **Resources** と指定し、**[OK]** をクリックします。
2. **Resources** フォルダを右クリックし、コンテキストメニューから**[追加]**→**[既存の項目]**を選択します。サンプルの場所を参照し、**dickens.htm** を **RichTextBoxSamples\Resources** フォルダから選択します。**[OK]**をクリックして、このファイルを **Resources** フォルダに追加します。
3. ソリューションエクスプローラーで **dickens.htm** ファイルを選択し、[プロパティ]ウィンドウで、**Build Action** プロパティを**[埋め込みリソース]**に設定します。アプリケーションをリビルドします。
4. アプリケーションのページを右クリックし、コンテキストメニューから**[コードの表示]**を選択します。**MainPage.xaml.cs** ページが表示されます。
5. 次の名前空間をインポートします。

C# コードの書き方

C#

```
using Windows.UI.Xaml.Printing;
using C1.Xaml.RichTextBox;
using Windows.UI.ViewManagement;
using Windows.Graphics.Printing;
using System.Reflection;
using Windows.UI.Popups;
```

6. 次のコードのように MainPage クラスを編集します。

C# コードの書き方

C#

```
public sealed partial class MainPage : Page
    {
```

```
        /// <summary>
        /// PrintDocument は、ページの印刷を準備するために使用します。
        /// Paginate イベント、GetPreviewPage イベント、および AddPages イベントのハンドラで、
印刷するページを準備します。
        /// </summary>
        protected PrintDocument printDocument = null;
        /// <summary>
        /// ドキュメントソースのマーカーインタフェース
        /// </summary>
        protected IPrintDocumentSource printDocumentSource = null;
        /// <summary>
        /// rtb ページの格納に使用する UIElement のリスト。
        /// </summary>
        internal List<FrameworkElement> pages = null;
        /// <summary>
        /// C1RichTextBox のドキュメントの印刷に使用します。
        /// </summary>
        C1RichTextViewManager viewManager;
```

7. 次のコードのように **MainPage()** コンストラクタを編集します。**GetManifestResourceStream()** の "YourApplicationName" を実際のアプリケーション名に置き換えることを忘れないでください。

C# コードの書き方

C#

```
public MainPage()
        {
            this.InitializeComponent();

            Assembly asm = typeof(MainPage).GetTypeInfo().Assembly;
            Stream stream =
asm.GetManifestResourceStream("YourApplicationName.Resources.dickens.htm");
            var html = new StreamReader(stream).ReadToEnd();
            rtb.Html = html;

            pages = new List<FrameworkElement>();
            this.Loaded += Printing_Loaded;
            this.Unloaded += Printing_Unloaded;
        }
```

この手順では、**Resources** ファイルと、適切な既存の **dickens.htm** ファイルを追加しました。また、**MainPage.xaml.cs** ファイルにコードを追加しました。次の手順では、汎用の **Button** コントロールの **Button_Click** イベントと、**MainPage()** コンストラクタに追加した **Printing** イベントを処理する残りのコードを追加します。

手順3：アプリケーションコードの追加

前の手順では、リソースファイルを追加し、アプリケーションへのコードの追加を開始しました。この手順では、残りのアプリケーションコードを追加します。

1. まず、**Printing_Loaded** イベント、**Printing_Unloaded** イベント、および **btnPrint_Click** イベントをイベントハンドラ領域内に追加します。

C# コードの書き方

C#

```
#region event handlers
        void Printing_Unloaded(object sender, RoutedEventArgs e)
        {
            UnregisterForPrinting();
        }

        void Printing_Loaded(object sender, RoutedEventArgs e)
        {
            // 印刷を初期化します
            RegisterForPrinting();
        }

        private async void btnPrint_Click(object sender, RoutedEventArgs e)
        {
            await Windows.Graphics.Printing.PrintManager.ShowPrintUIAsync();
        }
        #endregion
```

2. 印刷を実装するための領域を作成します。

C#

```
 #region implemention
 #endregion
```

3. 実装領域内に、**PrintTaskRequested** イベントハンドラを追加します。

C# コードの書き方

C#

```
 /// <summary>
        /// これは、PrintManager.PrintTaskRequested のイベントハンドラです。
        /// </summary>
        /// <param name="sender">PrintManager</param>
        /// <param name="e">PrintTaskRequestedEventArgs </param>
        protected void PrintTaskRequested(PrintManager sender,
PrintTaskRequestedEventArgs e)
        {
            PrintTask printTask = null;
            printTask = e.Request.CreatePrintTask("SamplePDF", sourceRequested
=>
            {
                // 印刷ジョブが完了すると、印刷タスクのイベントハンドラが呼び出されます。
                printTask.Completed += async (s, args) =>
                {
                    // 印刷操作が失敗したらユーザーに通知します。
                    if (args.Completion == PrintTaskCompletion.Failed)
                    {
                        await
Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, () =>
                        {
                            MessageDialog dialog = new MessageDialog("Failed to
print.");
                            dialog.ShowAsync();
```

```
                        });
                    }
                };
```

4. **PrintTaskRequested** イベントハンドラのすぐ下に、印刷サイズや印刷方向などのオプションを設定します。

C# コードの書き方

C#

```
// 用紙サイズや用紙方向などの印刷オプションを設定します

Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, () =>
                {
                    var layout = rtb.ViewManager.PresenterInfo as C1PageLayout;
                    if (layout != null && layout.Width > layout.Height)
                    {
                        printTask.Options.Orientation =
PrintOrientation.Landscape;
                    }
                });

                sourceRequested.SetSource(printDocumentSource);
            });

        }
```

5. 次に追加するメソッドは、印刷用アプリケーションを Windows に登録し、印刷プロセスのイベントハンドラを設定します。

C# コードの書き方

C#

```
 /// <summary>
        /// このメソッドは、印刷用アプリケーションを Windows に登録し、印刷プロセスに必要なイベントハンドラを設定します。
        /// </summary>
        protected void RegisterForPrinting()
        {
            // PrintDocument を作成します。
            printDocument = new PrintDocument();

            // DocumentSource を保存します。
            printDocumentSource = printDocument.DocumentSource;

            // 印刷プレビューを設定するイベントハンドラを追加します。
            printDocument.Paginate += Paginate;

            // 指定されたプレビューページを提供するイベントハンドラを追加します。
            printDocument.GetPreviewPage += GetPrintPreviewPage;

            // すべての最終印刷ページを提供するイベントハンドラを追加します。
            printDocument.AddPages += AddPrintPages;

            // PrintManager を作成し、印刷初期化のハンドラを追加します。
```

```
            PrintManager printMan = PrintManager.GetForCurrentView();
            printMan.PrintTaskRequested += PrintTaskRequested;
        }
```

6. 印刷用アプリケーションを登録解除するメソッドを追加します。

C# コードの書き方

C#

```
/// <summary>
        /// このメソッドは、印刷用アプリケーションの Windows への登録を解除します。
        /// </summary>
        protected void UnregisterForPrinting()
        {
            if (printDocument == null)
                return;

            printDocument.Paginate -= Paginate;
            printDocument.GetPreviewPage -= GetPrintPreviewPage;
            printDocument.AddPages -= AddPrintPages;

            // 印刷初期化のハンドラを削除します。
            PrintManager printMan = PrintManager.GetForCurrentView();
            printMan.PrintTaskRequested -= PrintTaskRequested;
        }
```

7. 実装領域内に最後に追加するコードには、3つのイベントハンドラがあります。

C# コードの書き方

C#

```
 /// <summary>
        /// これは、PrintDocument.Paginate のイベントハンドラです。
        /// PrintManager が印刷プレビューを要求すると起動します
        /// </summary>
        /// <param name="sender">PrintDocument</param>
        /// <param name="e">Paginate Event Arguments</param>
        protected void Paginate(object sender, PaginateEventArgs e)
        {
            pages.Clear();

            viewManager = new C1RichTextViewManager
            {
                Document = rtb.Document,
                PresenterInfo = rtb.ViewManager.PresenterInfo
            };

            PrintDocument printDoc = (PrintDocument)sender;
            // プレビューページ数を報告します
            printDoc.SetPreviewPageCount(rtb.ViewManager.Presenters.Count,
PreviewPageCountType.Intermediate);
        }

        /// <summary>
```

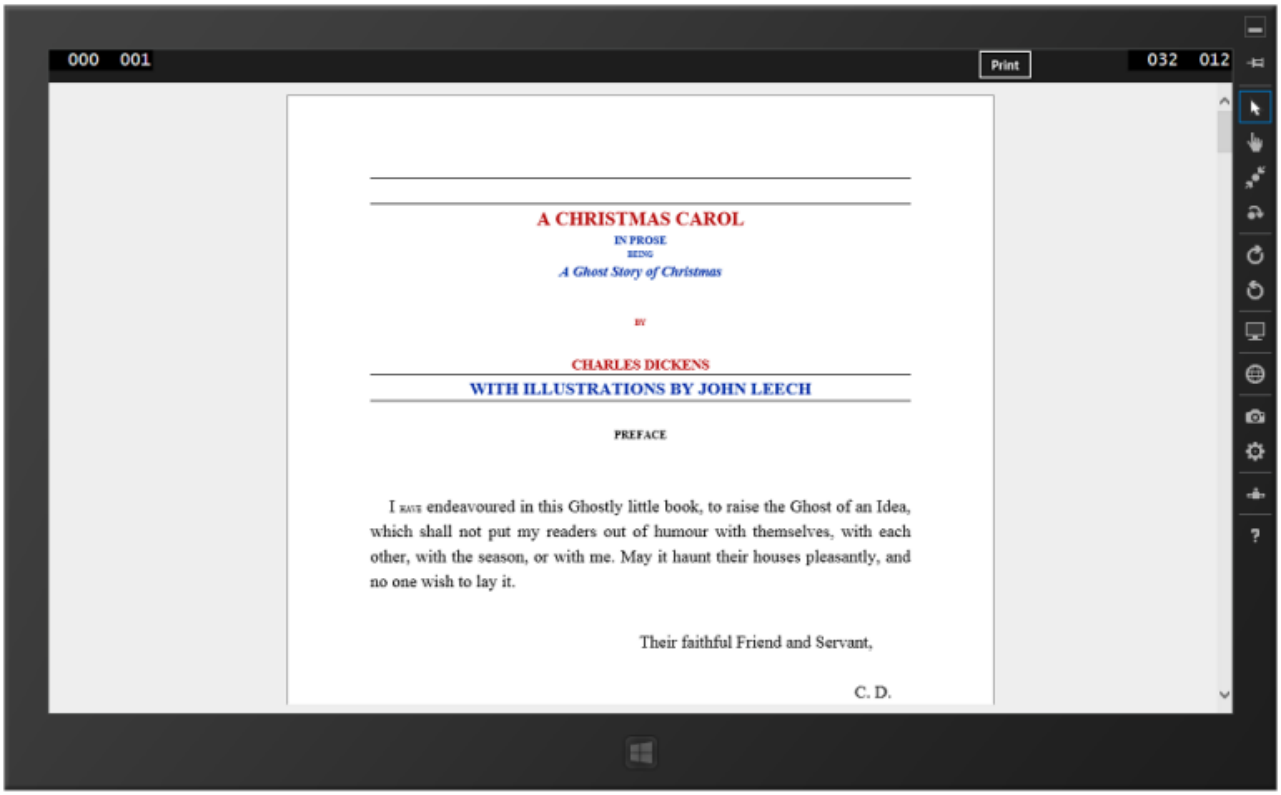
```
        /// これは、PrintDocument.GetPrintPreviewPage のイベントハンドラです。特定の印刷プレビューページを
        /// UIElement の形式で PrintDocument のインスタンスに提供します。次に PrintDocument は、この UIElement を
        /// Windows 印刷システムが処理できるページに変換します。
        /// </summary>
        /// <param name="sender">PrintDocument</param>
        /// <param name="e">プレビュー要求されたページを含む引数</param>
        protected void GetPrintPreviewPage(object sender, GetPreviewPageEventArgs e)
        {
            // 最初のページを追加します
            if (e.PageNumber == 1)
                AddOnePrintPreviewPage(0);

            PrintDocument printDoc = (PrintDocument)sender;
            printDoc.SetPreviewPage(e.PageNumber, pages[e.PageNumber - 1]);
            // 他のページを追加します
            if (e.PageNumber == 1)
                for (int i = 1; i < viewManager.Presenters.Count; i++)
                    AddOnePrintPreviewPage(i);
        }

        /// <summary>
        /// これは、PrintDocument.AddPages のイベントハンドラです。印刷するすべてのページを
        /// UIElement の形式で PrintDocument のインスタンスに提供します。次に PrintDocument は、これらの UIElement を
        /// Windows 印刷システムが処理できるページに変換します。
        /// </summary>
        /// <param name="sender">PrintDocument</param>
        /// <param name="e">印刷タスクのオプション参照を含むページ追加イベントの引数</param>
        protected void AddPrintPages(object sender, AddPagesEventArgs e)
        {
            // すべてのページをループし、印刷する各ページを追加します
            foreach (FrameworkElement page in pages)
            {
                printDocument.AddPage(page);
            }
            PrintDocument printDoc = (PrintDocument)sender;
            // すべての印刷ページが準備できたことを示します
            printDoc.AddPagesComplete();
        }

        void AddOnePrintPreviewPage(int index)
        {
            var page = (FrameworkElement)printTemplate.LoadContent();
            page.DataContext = viewManager.Presenters[index];
            if (!pages.Contains(page))
                pages.Add(page);
        }
```

この手順では、アプリケーションコードを追加しました。次の手順では、このアプリケーションを実行します。

手順4：アプリケーションの実行

前の手順では、アプリケーションコードを追加しました。この手順では、アプリケーションを実行します。

1. **[F5]**キーを押すか、デバッグを開始して、アプリケーションを実行します。アプリケーションは次の図のようになります。



2. [印刷]ボタンをタップまたはクリックします。使用するプリンタを選択できます。

3. プリンタを選択します。[印刷]ダイアログボックスが開き、印刷後のドキュメントをプレビューしたり、印刷設定を変更することができます。

ここまでの成果

このチュートリアルでは、ネイティブな Windows 印刷イベントを使用して、C1RichTextBox コントロールに印刷機能を追加しました。また、ドキュメントを **Resources** フォルダからロードし、XAML マークアップを使用して **C1RichTextBox** コントロールを作成しました。