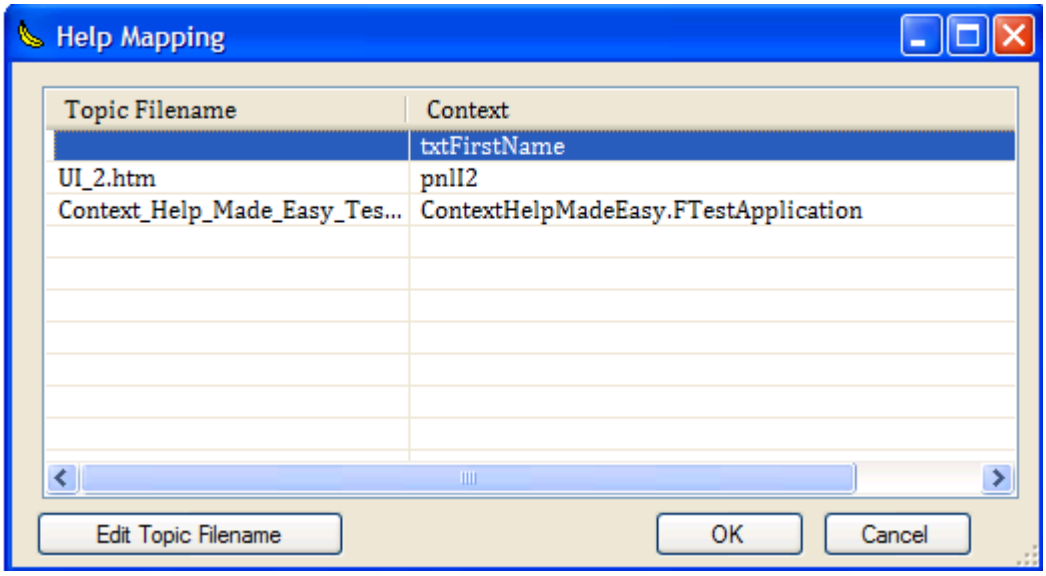
عنوان مقاله	روش آسان برای راهنمای متنی
عنوان اصلی	**Context Help Made Easy**
کلمات کلیدی	C#, Windows, .NET 2.0, .NETVisual Studio, VS2005, Architect
مؤلف	**Tom Clement**
مرجع	http://www.codeproject.com
سطح	مبتدی
مترجم	مهدی عبداللهی (http://m0911.wordpress.com)
تاریخ انتشار	۹ اسفند ۱۳۸۸
تعداد صفحه	۱۵
فایل های ضمیمه	src_ContextHelpMadeEasy.zip

فهرست

مقدمه

این چیزی است که همه توسعه دهندگان با آن مواجه می شوند. در مراحل نهایی پروژه متوجه می شوید که زمان زیادی را برای کار با برنامه سازنده راهنمای HTML صرف می کنید تا صفحات و پنجره های مکالمه برنامه را به راهنمای حساس به متن متصل نمایید.

در این مقاله شما یک روش جدید می بینید که این امکان را به تهیه کنندگان راهنمای برنامه می دهد تا موضوعات راهنما را به بخش های تصویری برنامه منتسب کنند. این کار بدون نیاز به دستکاری واسط کاربری برنامه یا سر و کار یافتن با توسعه دهندگان برنامه و حتی بعد از کامپایل آن انجام می گیرد. هیچ نیازی نیست که نویسنده راهنمای برنامه فایل ها را به صورت دستی ویرایش کند یا در مورد فایل های داخلی اطلاعاتی داشته باشد.



راهنمای متنی چیست؟

در زمان اجرای یک برنامه شما ممکن است در یک لحظه با چند موضوع مرتبط باشید. مثلا مکان نما روی یک کنترل جعبه متن (TextBox) است، موضوع راهنما می تواند موراد زیر باشد:

یک جعبه متن

یک کنترل Tab حاوی جعبه متن

یک کنترل Tab حاوی TabPage

یک پنجره مکالمه حاوی TabPage

و خود برنامه

این بسته به تصمیم مستند ساز برنامه است که کدام یک از این گزینه ها موضوع اصلی راهنما باشد.

روش کلی اغلب این است که به هنگام اجرای راهنمای حساس به متن، برنامه کنترلی را که مکان نما روی آن است به دست می آورد و بررسی می کند که آیا موضوع راهنمای برنامه به آن منتسب شده است یا خیر. در صورت وجود این گزینه، برنامه راهنمای مربوط به آن را اجرا می کند و در غیر این صورت برنامه کنترل مافوق (در بردارنده این کنترل) را بررسی می کند و این روال را ادامه می دهد.

پیاده سازی راهنمای F1

می خواهیم که با زدن کلید F1، برنامه راهنمای مربوط را نمایش دهد. برای این کار از یک ContextID استفاده می کنیم و آن را به موضوعات راهنما که در یک فایل XML قرار دارد متصل می کنیم. بعد از این امکانی ر به برنامه اضافه می کنیم که مستند سازان برنامه بتوانند این ارتباطات را از داخل واسط کاربری برنامه انجام دهند.

واسط IContextHelp

اولین گام ساختن یک واسط IContextHelp است که شما بتوانید آن را به کنترل های مختلفی در پروژه تان اضافه کنید. این واسط، کنترلهایی را که قرار است به عنوان موضوع راهنمای برنامه باشند، شناسایی می کند و یک شناسه منحصر به فرد رشته ای (ContextID) را برای شناسایی متن آماده می کند.

```
public interface IContextHelp
{
  string ContextHelpID { get; }
}
```

از آنجایی که اغلب کنترل ها از کلاس UserControl مشتق شده اند، ما می توانیم مراحل را با استفاده از UserControlEx - که واسط مورد نظر را پیاده می سازد - ساده تر کنیم. برای استفاده از این، تمامی کنترل ها را به جای UserControl از UserControlEx مشتق (ارث بری) کنید. در پیاده سازی پیش فرض، ContextID برای کنترل دارای نام کامل کلاس کنترل مورد نظر است.

```
public virtual string ContextHelpID {get{return this.GetType().FullName;}}
```

به همین صورت ما یک کلاس FormEx - که ابزاری برای پیاده سازی IcontextHelp است - ایجاد می نماییم. زمانی که ما یک فرم روی برنامه مان تعریف می کنیم، آن را به جای Form از FormEx به ارث می بریم.

اگر شما بخواهید تا راهنمای حساس به متن را برای پنجره های استاندارد یا کنترل های دیگر آماده کنید، می توانید این کار را به سادگی با ارث بری کلاس کنترل های جدید و اضافه کردن واسط مورد بحث انجام بدهید. برای مثال:

```
// PanelEx is like a Panel, except that it implements IContextHelp

public class PanelEx : Panel, IContextHelp
{
  private string m_sContextID;
  public string ContextHelpID
  {
    get
    {
      if (string.IsNullOrEmpty(m_sContextID))
        return this.Name;
      return m_sContextID;
    }
    set { m_sContextID = value; }
  }
}
```

در این مورد، ما نام کنترل ها را شبیه id های پیش فرض استفاده می کنیم و در مواردی که تداخل نام وجود داشته باشد اجازه ی لغو (نام کلاس مافوق) داده می شود.

اگر شما نیاز دارید تا یک context ID برای پنجره های مختلف آماده کنید، باید یک IExtenderProvider (مطابق شرح چارلز ویلیامز) پیاده نمایید.

به دام انداختن کلید F1

شما می توانید از یک وقفه ی دستی برای مدیریت و عکس العمل به فشردن کلید F1 در سراسر برنامه تان استفاده کنید. برای این کار باید یک فیلتر پیغام (message filter) به برنامه تان اضافه نمایید و به هنگام دریافت پیغام WM_KEYDOWN فشرده شدن کلید F1 را بررسی نمایید.

```
static void Main()
{
   //add message filter

   MessageFilter oFilter = new MessageFilter();
   System.Windows.Forms.Application.AddMessageFilter(
                       (IMessageFilter)oFilter);
   // etc.

}
internal class MessageFilter : IMessageFilter
{
  #region IMessageFilter Members
  bool IMessageFilter.PreFilterMessage(ref Message m)
  {
     //Use a switch so we can trap other messages in the future.

     switch (m.Msg)
     {
       case 0x100 : // WM_KEYDOWN

         if ((int)m.WParam == (int)Keys.F1)
         {
           HelpUtility.ProcessHelpRequest(Control.FromHandle(m.HWnd));
           return true;
         }
         break;
     }
     return false;
  }
  #endregion
}
```

بررسی کلید F1

حال که به ساختار اصلی کار پی بردیم، می توانیم نشان دهیم کلید F1 چطور کارمی کند. بعدها ما این متد را بر روی کلید Ctrl امتحان می کنیم، اما فعلا به این صورت نمایش داده می شود:

```
public static class HelpUtility
{
  public static void ProcessHelpRequest(Control ctrContext)
  {
    ShowContextHelp(ctrContext);
  }
}
```

متدShowContextHelp() به طور متوالی دنبال کنترل (هایی) می گردد که ویژگی های زیر را داشته باشد:

۱.IcontextHelp را پیاده سازی کرده است

۲. یک IContextHelp.ContextHelpID مقدار دهی شده دارد

۳. یک ورودی متناظر با آن در فایل xml (فهرست) موضوعات راهنما دارد.

اگر موردی پیدا شد برنامه، نمایشگر راهنما را اجرا می کند تا آن را نشان دهد و در غیر این صورت موضوع پیش فرض را نشان می دهد.

```
// Process a request to display help

// for the context specified by ctrContext.

public static void ShowContextHelp(Control ctrContext)
{
  Control ctr = ctrContext;

  string sHTMLFileName = null;
  while (ctr != null)
  {
    // Get the first control in the parent chain

    // with the IContextHelp interface.

    IContextHelp help = GetIContextHelpControl(ctr);
    // If there isn't one, display the default help for the application.

    if (help == null)
      break;
    // Check to see if it has a ContextHelpID value.

    if (help.ContextHelpID != null)
    {
      // Check to see if the ID has a mapped HTML file name.

      sHTMLFileName = LookupHTMLHelpPathFromID(help.ContextHelpID);
      if (sHTMLFileName != null && ShowHelp(ctrContext, sHTMLFileName))
        return;
    }
    // Get the parent control and repeat.

    ctr = ((Control)help).Parent;
  }
  // Show the default topic.

  ShowHelp(ctrContext, "");
}
```

متد ()GetIContextHelpControl جستجو را برای یک کنترل IContextHelp انجام می دهد.

```
// Get the first control in the parent chain

// (including the control passed in)

// that implements IContextHelp.

private static IContextHelp GetIContextHelpControl(Control ctl)
{
  while (ctl != null)
  {
    IContextHelp help = ctl as IContextHelp;
    if (help != null)
    {
      return help;
    }
    ctl = ctl.Parent;
  }
  return null;
}
```

متد ShowHelp() برنامه help html را اجرا می نماید وموضوع مشخص شده توسط sHTMLHelp را نمایش می دهد.

```
// Display the specified help page.

private static bool ShowHelp(Control ctlContext, string sHTMLHelp)
{
  try
  {
    if (string.IsNullOrEmpty(sHTMLHelp))
      Help.ShowHelp(ctlContext, HelpUtility.HelpFilePath);
    else
      Help.ShowHelp(ctlContext, HelpUtility.HelpFilePath,
                    HelpNavigator.Topic, sHTMLHelp);
  }
  catch (ArgumentException)
  {
    // Ideally, we would return false when

    // the HTML file isn't found in the CHM file.

    // Unfortunately, there doesn't seem to be

    // a way to do this without parsing the CHM.

    return false;
  }
  return true;
}
// Define this contstant at the top of the file.

private const string mc_sHELPFILE = "ContextHelpMadeEasy.chm";
// Return the path to the CHM file.

private static string HelpFilePath
{
  get
  {
    return Path.Combine(System.Windows.Forms.Application.StartupPath,
                        mc_sHELPFILE);
  }
}
```

خواندن فایل فهرست راهنمای موضوعات راهنما (mapping file)

برای تکمیل راهنمای F1 شما باید تابع LookupHTMLHelpPathFromID() را - که از داخل ShowContextHelp() فرا خوانده می شود – پیاده سازی نمایید. این متد id (شناسه) موضوع را در فایل فهرست موضوعات جستجو می کند و در صورت موجود بودن آن نام فایل حاوی راهنمای موضوع را برمی گرداند. فهرست مورد بحث در نخستین دسترسی از فایل XML خوانده می شود و در حافظه رم (cache) می ماند تا با هر بار فشار دادن کلید F1 از روی دیسک خوانده نشود. ما کار را با تعریف StringDictionary برای ذخیره فهرست موضوعات ادامه می دهیم. بدین منظور تعدادی ثابت تعریف می کنیم تا نام فایل حاوی فهرست موضوعات و همچنین عناصر مختلف XML و نام مشخصه ها را نمایش بدهد.

```
static private StringDictionary ms_sdContextPaths = null;
private const string mc_sMAPPING_FILE_NAME = "HelpContextMapping.Config";
private const string mc_sIDMAP_ELEMENT_NAME = "IDMap";
private const string mc_sCONTEXTID_ELEMENT_NAME = "ContextID";
private const string mc_sID_ATTRIBUTE_NAME = "ID";
private const string mc_sHTMLPATH_ATTRIBUTE_NAME = "HTMLPath";
```

مشخصه MappingFilePath مسیر فایل فهرست راهنما را برمی گرداند که به صورت فرضی در همان پوشه ای قرار می گیرد که برنامه اجرایی در آن است.

```
private static string MappingFilePath
{
  get { return Path.Combine(Application.StartupPath, mc_sMAPPING_FILE_NAME); }
}
```

مشخصه ContextPaths در نخستین فراخوانی فهرست راهنما را از داخل فایل می خواند.

```
private static StringDictionary ContextPaths
{
  get
  {
    if (ms_sdContextPaths == null)
    {
      ms_sdContextPaths = ReadMappingFile();
    }
    return ms_sdContextPaths;
  }
}
```

متد ReadMappingFile() یک StringDictionary را می سازد و آن را با اطلاعاتی که از فایل پیکربندی فهرست موضوعات به صورت XML خوانده است، مقدار دهی می نماید.

```
// Read the mapping file to create a list of ID to HTML file mappings.

private static StringDictionary ReadMappingFile()
{
  StringDictionary sdMapping = new StringDictionary();
  XmlDocument docMapping = new XmlDocument();
  if (File.Exists(MappingFilePath) == true)
  {
    try { docMapping.Load(MappingFilePath); }
    catch
    {
      MessageBox.Show(string.Format("Could not read help mapping file '{0}'.",
            MappingFilePath), "Context Help Made Easy", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
      throw;
    }
    XmlNodeList nlMappings = docMapping.SelectNodes("//" +
                                    mc_sCONTEXTID_ELEMENT_NAME);
    foreach (XmlElement el in nlMappings)
    {
      string sID = el.GetAttribute(mc_sID_ATTRIBUTE_NAME);
      string sPath = el.GetAttribute(mc_sHTMLPATH_ATTRIBUTE_NAME);
```

```
      if (sID != "" && sPath != "")
          sdMapping.Add(sID, sPath);
    }
  }
  return sdMapping;
}
```

فایل فهرست موضوعات راهنما چیزی شبیه به زیر است:

```
<?xml version="1.0"?>
<IDMap>
<ContextID ID="namespace.fsettings" HTMLPath="SettingsTopic.htm" />
<ContextID ID="controlname1" HTMLPath="Control1Topic.htm" />
<ContextID ID="overridenidctl2" HTMLPath="Control2Topic.htm" />
</IDMap>
```

با پیاده سازی متدهای فوق الذکر، پیاده سازی LookupHTMLHelpPathFromID() بسیار آسان خواهد بود.

```
// Given an ID, return the associated HTML Help path

private static string LookupHTMLHelpPathFromID(string sContextID)
{
  if (ContextPaths.ContainsKey(sContextID))
    return ContextPaths[sContextID];
  return null;
}
```

در نهایت پس از پیاده سازی کلید F1 ما بخش اصلی این برنامه را انجام می دهیم. یعنی ایجاد امکان برای مستند سازان برنامه تا بتوانند راهنمای حساس به متن را بدون نیاز به شما (کد نویس نرم افزار) ایجاد نمایند.

پیاده سازی کد- توانمند کردن مستند سازان برنامه

تقریبا تمام کارهایی را که برای جداسازی تیم مستند ساز از تیم توسعه نرم افزار لازم است، انجام داده ایم. به وسیله کدهای زیر افراد تیم مستندسازی می توانند به طور دستی فایل پیکربندی xml را برای اضافه کردن یا ویرایش فهرست موضوعات راهنما تغییر دهند. به شرط این که ContextID ی مربوط به هر بخش برنامه را که می خواهند راهنمای آن را بنویسند، بدانند.

ما کد را طوی می نویسیم که مستند ساز برنامه در هر بخش از آن با زدن کلید ترکیبی Ctrl-F1 فهرست تمام ContextID های مربوط به آن پنجره را ببیند و پس از آن می توانند با استفاده از پنجره مکالمه، ContextID مربوط به یک موضوع را به نام فایل html منتسب نمایند. این تغییر در فایل حاوی فهرست موضوعات راهنما نوشته می شود و بلافاصله بعد از آن با زدن کلید F1 راهنمای (حساس به متن) مربوط به آن بخش را خواند دید.

مدیریت Ctrl-F1

ما می توانیم متد ProcessHelpRequest() - که دربالا شرح داده شد- برای بررسی کلید Ctrl به شرح ذیل اصلاح کنیم:

```
public static void ProcessHelpRequest(Control ctrContext)
{
  if (Control.ModifierKeys == Keys.Control)
  {
    ShowHelpMappingDialog(ctrContext);
    return;
  }
  ShowContextHelp(ctrContext);
}
```

در ضمن ممکن است بخواهید کاری بکنید که به هنگام زدن کلید Ctrl-F1 توسط کاربران نهایی برنامه جعبه محاوره مربوط به ContextID ها نمایش داده نشوند. در پیاده سازی که ما انجام داده ایم، این تست برای زدن کلید Ctrl از طریق عمل ضرب بیتی با یک کلید در رجیستری ویندوز انجام می شود و بررسی می کند که آیا این امکان (برای مستند ساز) فعال هست یا نه. شما می توانید از روش های دیگری نیز برای تشخیص این امر استفاده کنید.

نمایش پنجره مکالمه انتساب موضوعات راهنما

با فشردن کلید Ctrl و دریافت پیغام WM_KEYDOWN به جای نمایش راهنما برای کنترل (یا موضوع) مورد نظر، متد ShowHelpMappingDialog() را فراخوانی می کنیم.

```
// Traverse the parent control chain looking for controls that implement the

// IContextHelp interface.  For each one found, add it to the list of available

// contexts.  Include the associated HTML path if it's define

// Finally, show the dialog for the help author to edit the mappings.

public static void ShowHelpMappingDialog(Control ctrContext)
{
  IContextHelp help = GetIContextHelpControl(ctrContext);
  List<ContextIDHTMLPathMap> alContextPaths = new List<ContextIDHTMLPathMap>();
  // Create a list of contexts starting with the current help context

  // and moving up the parent chain.

  while (help != null)
  {
    string sContextID = help.ContextHelpID;
    if (sContextID != null)
    {
      string sHTMLHelpPath = LookupHTMLHelpPathFromID(sContextID);
      alContextPaths.Add(new ContextIDHTMLPathMap(sContextID, sHTMLHelpPath));
    }
    help = GetIContextHelpControl(((Control)help).Parent);
  }
  // Pop up the mapping dialog. If it returns true, this means a change was made

  // so we rewrite the XML mapping file with the new information.

  if (FHelpMappingDialog.ShowHelpWriterHelper(alContextPaths) == true)
  {
    foreach (ContextIDHTMLPathMap pathMap in alContextPaths)
    {
      if (!string.IsNullOrEmpty(pathMap.ContextID))
      {
        if (!string.IsNullOrEmpty(pathMap.HTMLPath))
        {
          ContextPaths[pathMap.ContextID] = pathMap.HTMLPath;
        }
        else
        {
          if (ContextPaths.ContainsKey(pathMap.ContextID))
            ContextPaths.Remove(pathMap.ContextID);
        }
      }
    }
    SaveMappingFile(ContextPaths);
  }
 }
}
```

پنجره مکالمه انتساب موضوعات راهنما، با فراخوانی FHelpMappingDialog.ShowHelpWriterHelper() - که در زیر شرح داده شده است - باز می شود و به صورت یک ویراستار فهرست ساختار ContextIDHTMLPathMap عمل خواهد کرد.

ساختار ContextIDHTMLPathMap شامل ۲ رشته و ۱ سازنده می باشد.

```
// Utility class for maintaining relationship between context Id and path.

public class ContextIDHTMLPathMap
{
  public string ContextID;
  public string HTMLPath;
  public ContextIDHTMLPathMap(string ID, string Path)
  {
    ContextID = ID;
    HTMLPath = Path;
  }
}
```

نوشتن نتایج

متد SaveMappingFile() ازداخل ShowHelpMappingDialog() فراخوانی می شود:

```
//  Saves the specified StringDictionary that contains ID to Path mappings to the

//  XML mapping file.

private static void SaveMappingFile(StringDictionary sdMappings)
{
  // Create a new XML document and initialize it with the XML declaration and the

  // outer IDMap element.

  XmlDocument docMapping = new XmlDocument();
  XmlDeclaration xmlDecl = docMapping.CreateXmlDeclaration("1.0", null, null);
  docMapping.InsertBefore(xmlDecl, docMapping.DocumentElement);
  XmlElement elIDMap = AddChildElementToNode(docMapping, docMapping,
    mc_sIDMAP_ELEMENT_NAME);
  // Add the defined mappings between contextID and filename.

  foreach (DictionaryEntry de in sdMappings)
  {
    XmlElement elMapping = AddChildElementToNode(elIDMap, docMapping,
      mc_sCONTEXTID_ELEMENT_NAME);
    elMapping.SetAttribute(mc_sID_ATTRIBUTE_NAME, de.Key as string);
    elMapping.SetAttribute(mc_sHTMLPATH_ATTRIBUTE_NAME, de.Value as string);
  }
  try
  {
    docMapping.Save(MappingFilePath);
  }
  catch
  {
    MessageBox.Show(string.Format("Could not write help mapping file '{0}'",
      MappingFilePath), "Context Help Made Easy", MessageBoxButtons.OK,
      MessageBoxIcon.Error);
    throw;
  }
}
```

تابع AddChildElementToNode() کد را قدری خوانا تر می سازد:

```
// Small utility method to add XML elements to a parent node.



private static XmlElement AddChildElementToNode(XmlNode node,

                XmlDocument doc, string elementName)

{
```

```
  XmlElement el = doc.CreateElement(elementName);

  node.AppendChild(el);

  return el;

}
```

پنجره مکالمه فهرست موضوعات راهنما

پنجره مکالمه فهرست موضوعات راهنما دارای یک کنترل ListView و دکمه هایی برای ویرایش نام فایل موضوعی، ok و cancel می باشد. این یک ویرایشگر برای فهرست ساختارContextIDHTMLPathMap - که محتوی عناوین موضوعات (context) کنترل های فعلی می باشد - است. پنجره مکالمه یک ListView نمایش می دهد و امکان می دهد نام فایل های html توسط کاربر (مستند ساز) ویرایش شود و نتایج در فهرست<List<ContextIDHTMLPathMap نوشته شود.

ورود به این پنجره مکالمه توسط متد ShowHelpWriterHelper() می باشد که فرم و مقادیر اولیه را معرفی می کند.

```
// Static entry point to pop up this form.

public static bool
       ShowHelpWriterHelper(List<ContextIDHTMLPathMap> contextIDs)
{
  FHelpMappingDialog frmHelper = new FHelpMappingDialog();
  frmHelper.IDList = contextIDs;  // Populate the treelist.

  if( frmHelper.lvMapping.Items.Count > 0 )
    frmHelper.lvMapping.SelectedIndices.Add(0);
  frmHelper.ShowDialog();         // Popup the form.

  if (frmHelper.Changed)
  {
    // For each item in the ListView,

    // change the path map to correspond to the UI.

    foreach (ListViewItem lvi in frmHelper.lvMapping.Items)
    {
      ContextIDHTMLPathMap pathMap = (ContextIDHTMLPathMap)lvi.Tag;
      pathMap.HTMLPath = (string)lvi.SubItems[0].Text.Trim();
    }
  }
  return frmHelper.Changed;
}
```

ListView در متد property set IDList مقدار دهی می شود:

```
// Gets and sets the list of ids.  The setter updates the UI.

public List<ContextIDHTMLPathMap> IDList
{
  set
  {
    lvMapping.Items.Clear();
    foreach (ContextIDHTMLPathMap pathMap in value)
    {
      AddMappingNode(pathMap);
    }
  }
}

// Utility to add a node to the treelist.

private void AddMappingNode(ContextIDHTMLPathMap pathMap)
{
  ListViewItem lvi = new ListViewItem(pathMap.HTMLPath);
  lvi.SubItems.Add(pathMap.ContextID);
  lvi.Tag = pathMap;
  lvMapping.Items.Add(lvi);
}
```

بقیه ی متدها سر جای خودشان هستند:

```
// Begin editing the label of the selected

// item when they click this button

private void btnEditTopicFile_Click(object sender, EventArgs e)
{
  if( lvMapping.SelectedItems.Count == 1 )
  {
    ListViewItem lvi = lvMapping.SelectedItems[0];
    lvi.BeginEdit();
  }
}

// If the item has changed after editing

// the label, flag the dialog as changed.

private void lvMapping_AfterLabelEdit(object sender,
            LabelEditEventArgs e)
{
  this.Changed = true;
};
}
```

جزئیات پنجره مکالمه را داخل کد منبع –ضمیمه همین مقاله- می توانید ببینید.

دستکاری

ما حالا یک سیستم راهنما داریم که می تواند توسط تیم مستند ساز (بعد از کامپایل برنامه) استفاده گردد. در این بخش یک سری ویژگی های دیگری را نیز برای توسعه این سیستم توضیح می دهیم.

مدیریت دکمه ?

زمانی که یک پنجره مکالمه با یک ContexID منتسب شده به آن داشته باشید (که از FormEx مشتق شده باشد) ممکن است بخواهید که کاربر با کلیک کردن دکمه «?» راهنمای html متناظر با آن را ببیند. ما FormEx را طوری تغییر می دهیم که این امکان را نیز داشته باشد. مشخصه های زیر را باید در محیط طراحی FormEx مقدار دهی کنیم:

```
HelpButton = true;
MaximizeBox = false;
MinimizeBox = false;
```

حال باید متد OnHelpButtonClicked() را لغو (بازنویسی) نماییم تا به جای عملیات پیش فرض به هنگام تغییر شکل مکان نما (مکان نمای ماؤس به علامت ?) تابع مورد نظر ما را فراخوانی نماید.

```
protected override void OnHelpButtonClicked(CancelEventArgs e)

{

  HelpUtility.ProcessHelpRequest(this);

  base.OnHelpButtonClicked(e);

  e.Cancel = true;

}
```

مدیریت کنترل های Tab

تمامی این کارها را ممکن است بخواهیم برای کنترل TabPage نیز انجام دهیم. برای به دست آوردن ContextID مربوط به یک کنترل که داخل یک TabPage است، کافی است مشخصه ContextID از فرم را - که tab control داخل آن است - را لغو (باز نویسی) کنیم.

```
public override string ContextHelpID
{
  get
  {
    switch (this.tcSettings.SelectedIndex)
    {
      case 0: return settingsGeneral1.ContextHelpID;
      case 1: return settingsConfiguration1.ContextHelpID;
    }
    return base.ContextHelpID;
  }
}
```

کد منبع و برنامه آزمایشی

برنامه آزمایشی ضمیمه این مقاله همراه با کد منبع است. در ضمن یک فایل راهنما به نام " *ContextHelpMadeEasy.chm* " است که شما می توانید برای آزمایش برنامه استفاده کنید. این فایل باید در پوشه برنامه اجرایی قرار گیرد. فایل های موضوعی در *ContextHelpMadeEasy.chm* به شرح زیر هستند:

- *Configuration_Settings.htm*
- *Configuration_Settings.htm*
- *Context_Help_Made_Easy_Test_Application.htm*
- *General_Settings.htm*
- *Settings.htm*
- *Topic_A.htm*
- *Topic_B.htm*
- *Topic_C.htm*
- *Topic_D.htm*
- *UI_1.htm*
- *UI_2.htm*
- *UI_1_Left_Side.htm*
- *UI_1_Right_Side.htm*
- *UI_2.htm*

برای آزمایش برنامه آن را اجرا کنید و در هر کدام از پنجره های آن مکان نما را روی یک کنترل دلخواه قرار دهید و کلید ترکیبی Ctrl-F1 را بزنید. پنجره مخصوص مستند ساز برنامه نمایش داده می شود و می توانید موضوعات راهنما را به آن منتسب کنید.