Tutorial – Advanced Controls

In this tutorial we are going to continue on with our work form the last lesson – Basic Windows Forms and Controls.

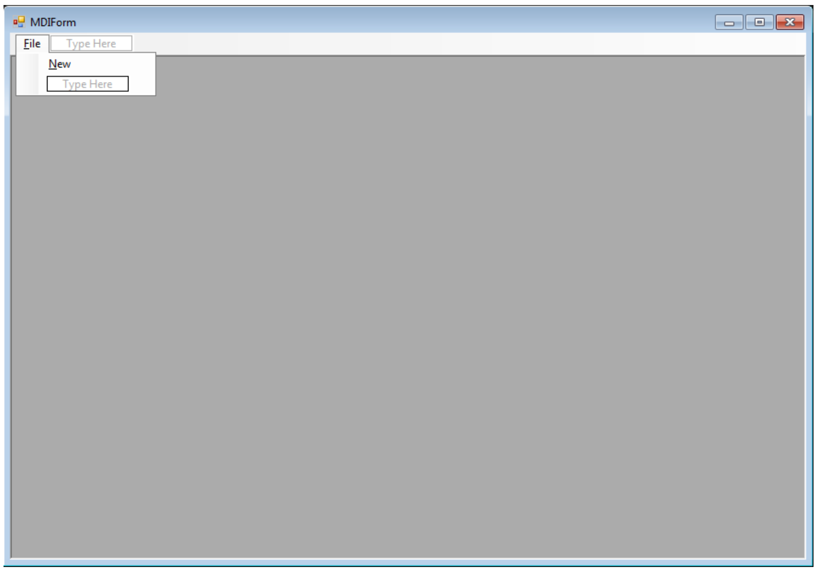
We will update our program so that we are using a Multiple Document Interface. This will allow us to add a menu strip to the program, to which we can add menu commands. Using menus will allow us to remove some components on our form and simplify the interface.

We'll also add a help system, which is essential for improving the usability of your tool.

Creating an MDI Application:

1. Add a new Windows Form to your project. From the *Project* menu select *Add Windows Form*.
2. With your new Form selected, set the *IsMdiContainer* property to *True*.
3. Add a *MenuStrip* control to the form.
4. Add a *File* (&File) menu item to the *MenuStrip*, and as the first child of the *File* menu, add the *New* (&New) menu item.

   Your Form should now look like this:



5. To get the program to show the MDI Form Container on launch (instead of the old Form), open Program.cs and update the code as follows:

```
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new MDIForm());
}
```

Menu Items:

We now want to add some processing behind our *New* menu command so that the user can open new Forms to edit their path nodes.

1. Open the Forms Designer and select the *MenuStrip* of the container form.
2. Select the *New* menu item and in the *Events* tab of the *Properties* window create a new event handler for the *Click* action. (Or, just double-click on the *New* menu item)
3. Edit the event handler as follows:

```
private void newToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form1 newMDIChild = new Form1();
    newMDIChild.MdiParent = this;
    newMDIChild.Show();
}
```

If you run your program now, you'll be able to display several Forms at once by selecting the *New* menu option multiple times.

Managing Multiple Windows:

We're going to add a *Windows* menu option that will allow the user to switch between the open forms in our MDI application.

1. In the *MenuStrip*, create a new menu item called *Window* (&Window)
2. With the *MenuStrip* selected, set the *MdiWindowListItem* property to *windowToolStripMenuItem* (or whatever name your *Window* menu item is called).

   If you can't see this property, make sure you actually have the *MenuStrip* selected and not the *MenuItem*

That's it. Windows handles the rest. Run your program now and your *Windows* menu should be updating according to the Forms you have open in your MDI container.

Adding a Save Menu Item:

We want to remove the 'Export' button on the form and replace it with a *Save* menu item under our *File* menu. This will make the interface much less cluttered and much more intuitive.

1. Open you Form in the Forms Designer and delete the *Export* button
2. The *Click* event handler for this Button will remain in the .cs file, but you will want to rename the function and remove the input arguments.
3. Open the MDI Container Form and add the *Save* (&Save) menu item to the *File* menu.

4. Add a *Click* event handler to the *Save* menu item.
5. In order to call the export function for the correct form, we need to get the active form from the MDI container. This is done via the ActiveMdiChild property.

   The lecture slides discuss this when implementing the Cut and Paste feature.

   Add the following code to your *Click* event handler for the *Save* menu item:

```
private void saveToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form activeChild = this.ActiveMdiChild;
    if(activeChild != null)
    {
        if(activeChild.GetType() == typeof(Form1))
        {
              // Cast the activeChild and call the appropriate function here
        }
    }
}
```

   (You can use the *as* keyword to cast)

Creating a Help File:

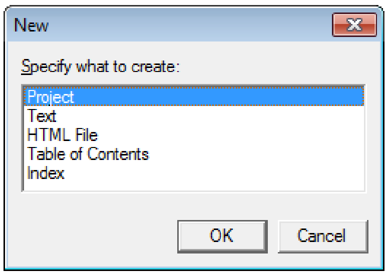
Creating a help system can get a little involved. Although the lecture slides talk about HTML files, we're going to go one step up and create a .chm help file (which is basically a collection of HTML files compiled into one file).

Using the *Start* menu, search for the program **HTML Help Workshop**, as it should already be installed on your machines.

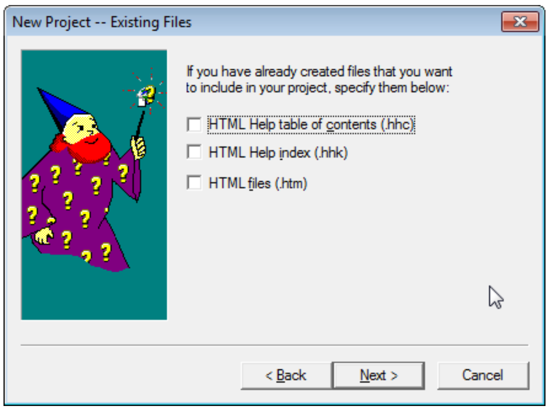
In the case it isn't, you can download the program via this link: http://www.microsoft.com/en-us/download/details.aspx?id=21138

This program is a bit archaic, but it's not too hard to figure out. Use this as a point of caution – don't make your programs this unintuitive.

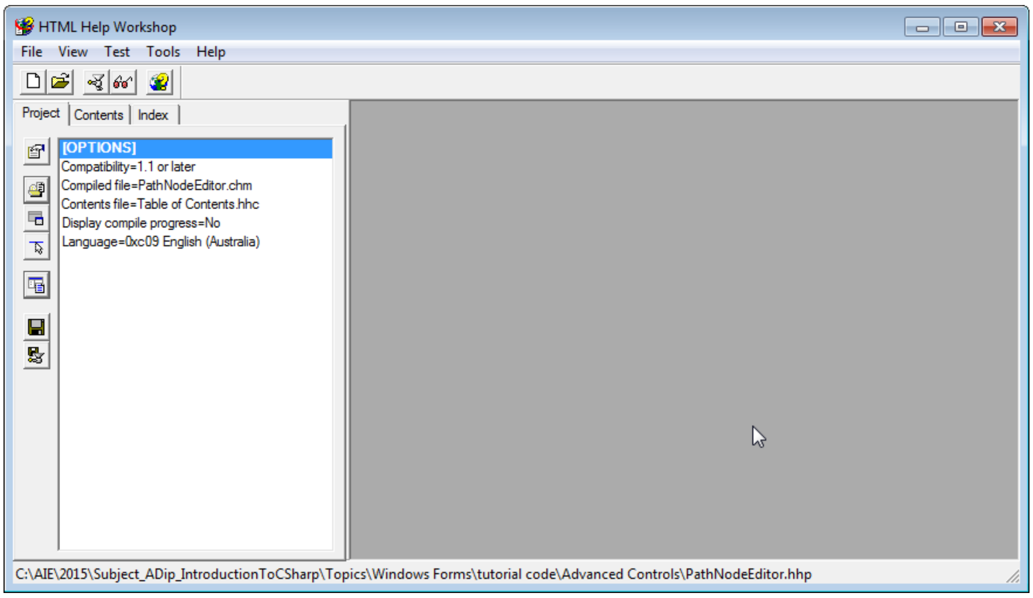
1. Open the program and select *File*, *New*, *Project*



   Complete the wizard. When you get to the *Existing Files* dialog, ensure nothing is selected.


New Project -- Existing Files
If you have already created files that you want to include in your project, specify them below:
HTML Help table of contents (.hhc)
HTML Help index (.hhk)
HTML files (.htm)
< Back
Next >
Cancel

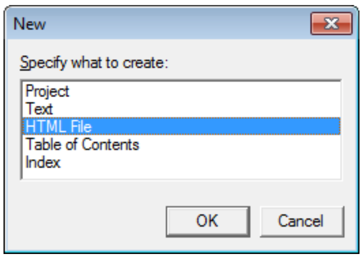
2. Once completed, you should see a screen that looks like this:



```
[OPTIONS]
Compatibility=1.1 or later
Compiled file=PathNodeEditor.chm
Contents file=Table of Contents.hhc
Display compile progress=No
Language=0xc09 English (Australia)
```

C:\AIE\2015\Subject_ADip_IntroductionToCSharp\Topics\Windows Forms\tutorial code\Advanced Controls\PathNodeEditor.hhp

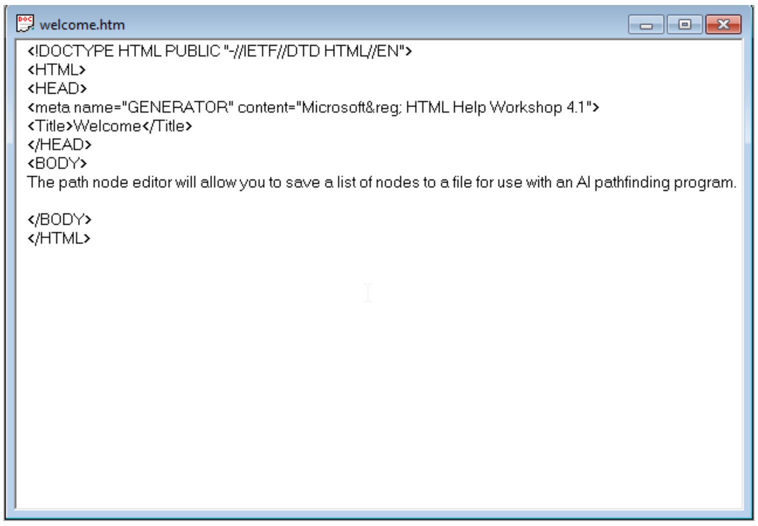
3. Use the *File*, *New* command again, but this time create a new *HTML File*



4. This is going to be our default welcome page.

   Enter a title for the page (for example “Welcome”), and fill the body with whatever you want to write (well, be helpful).

   You can format your page with HTML tags.



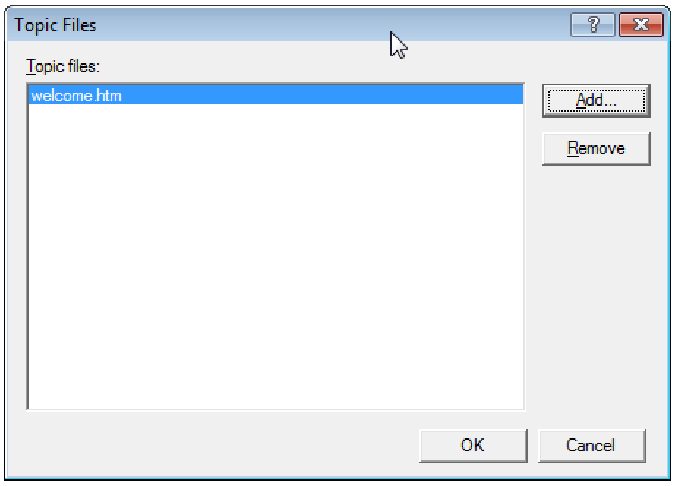
```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
<HEAD>
<meta name="GENERATOR" content="Microsoft&reg; HTML Help Workshop 4.1">
<Title>Welcome</Title>
</HEAD>
<BODY>
The path node editor will allow you to save a list of nodes to a file for use with an AI pathfinding program.

</BODY>
</HTML>
```

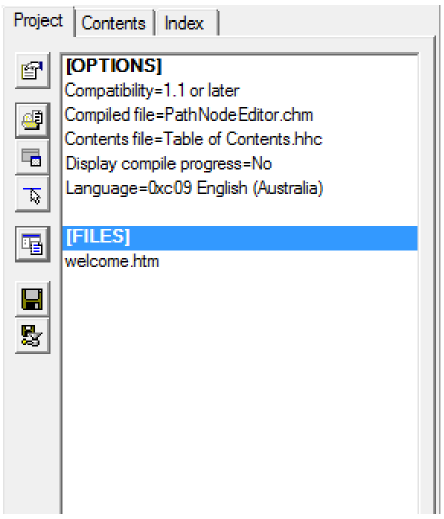
5. Save the file.

6. With the *Project* tab open, press the *Add/Remove topic files* button

   This opens a new window that allows you to add topic files. Press *Add* and select the welcome HTML file we just created



7. The project tab will update and display the name of the file you just added.



```
[OPTIONS]
Compatibility=1.1 or later
Compiled file=PathNodeEditor.chm
Contents file=Table of Contents.hhc
Display compile progress=No
Language=0xc09 English (Australia)

[FILES]
welcome.htm
```

This adds a default landing page, which will be displayed if anyone ever opens the .chm file directly.

The next step is to add Content pages. In your program you should have a new Content page for each control or form that launces the help system. Since our program currently only has the one MDI child form, we'll just create the one help page.

1. Select the *Contents* tab from the window on the left.

   You will be prompted to either create a new contents file, or open a new one. You will want to select the option for creating a new file.

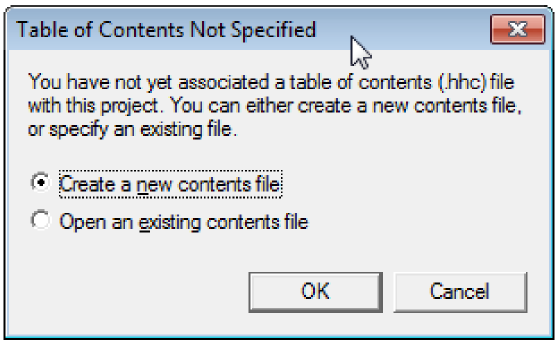


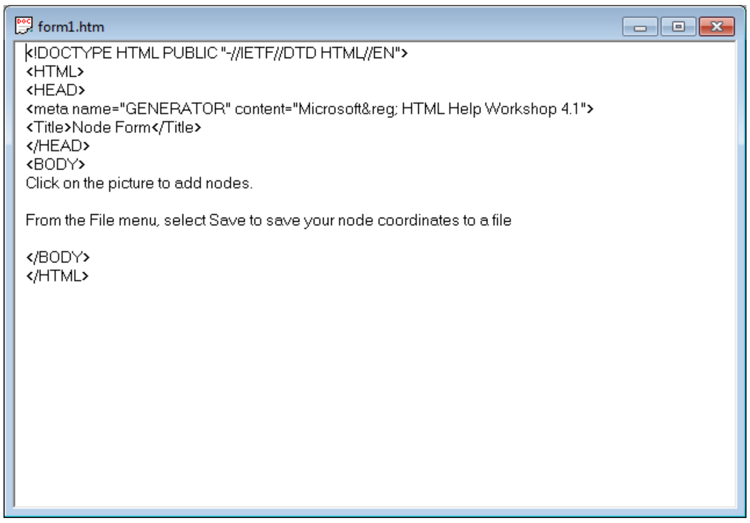
Table of Contents Not Specified

You have not yet associated a table of contents (.hhc) file with this project. You can either create a new contents file, or specify an existing file.

[✓ Create a new contents file]
Open an existing contents file

2. We need to create a new HTML file to store the content of our help page. From the *File* menu, again select *New*, *HTML File.*

   This will be the page that we'll display when someone presses *F1* when an MDI child form is open, so fill it out appropriately.



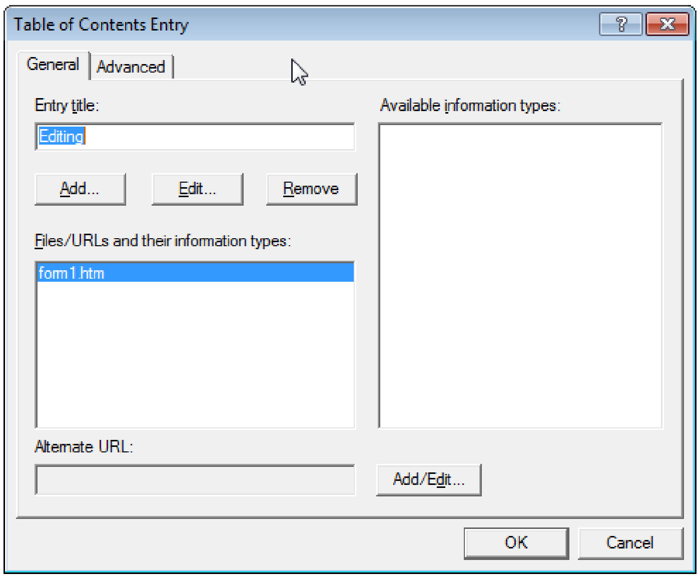
form1.htm

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
<HEAD>
<meta name="GENERATOR" content="Microsoft&reg; HTML Help Workshop 4.1">
<Title>Node Form</Title>
</HEAD>
<BODY>
Click on the picture to add nodes.

From the File menu, select Save to save your node coordinates to a file

</BODY>
</HTML>
```

3. From the buttons on the left, select *Insert a Page*
4. Fill out the dialog with a heading (for example, *Editing*), and add your HTML file using the *Add* button



Entry title: Editing

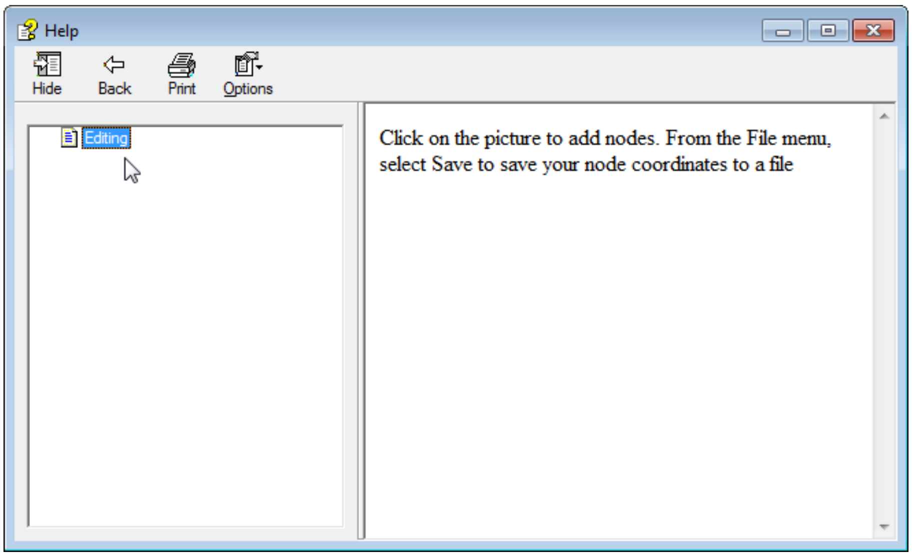
Available information types:

Files/URLs and their information types: form1.htm

Alternate URL:

Everything is set to go. Press the *Compile HTML File* button on the toolbar. This will save all documents and create the compiled .chm file.

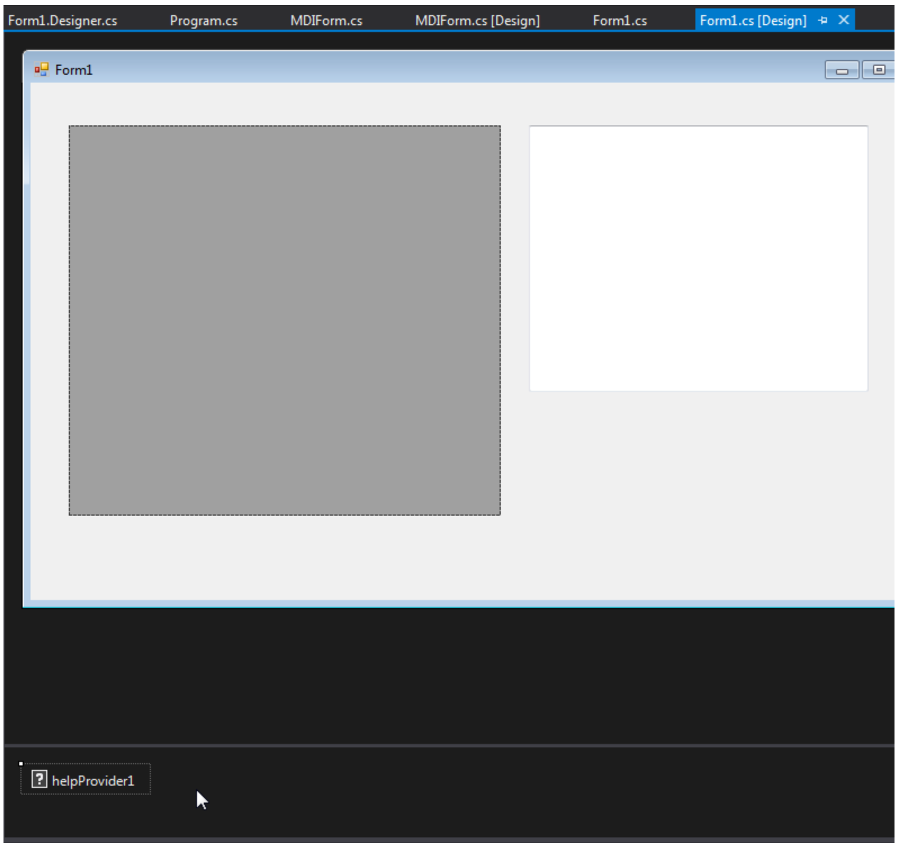
This is a good time to verify everything worked. Locate your .chm file and double-click on it. You should see the Windows Help program open and your content displayed.



Click on the picture to add nodes. From the File menu, select Save to save your node coordinates to a file

Adding Help to your Application:

1. Open your MDI child Form in the Forms Designer
2. Add the *HelpProvider* component to the Form. The control isn't visible in the form, but will be added to a special section below the form



3. Select the Form and modify the following properties:
   a. Set *HelpKeyword* to the name of the html file for the topic (*Editing*) you want to display. It should be the actual filename (for example, *form1.html*)
   b. Set the *HelpNavigator* to *Topic*

   This will ensure that when you press the *F1* key and a Form is active, the help system will open and go to your *Editing* topic.

If you desire you can set the *HelpButton* to *True* (make sure *MinimiseButton* and *MaximizeButton* are set to *False*). You could also make more topic pages and set the *HelpKeyword* settings for each control on the page.

Exercises:

1. Ensure your help pages are actually helpful and linked correctly with your program. You should have at least one help page for your program.
2. For the adventurous, add Cut and Paste functionality to the text box. You will need to ensure that when pasting your text is validated and any valid coordinates will add new node objects to your list (from the last tutorial).