Chapter 7

Accessibility and Globalization

Application Accessibility refers to the usability of an application for a large audience, including those users with impairments or with reduced abilities.

An application can have a geographically dispersed user base. In such a situation, the application should be able to adapt to the culture-specific needs of users. You can create such an application by implementing Globalization and Localization features in the application.

It is a good practice to develop the help file through Help Workshop and incorporate it in the application through the HelpProvider control.

This chapter discusses the use of accessibility and globalization features to create international application. In addition, this chapter explains how to implement the help files system to provide information to the application users.

Objectives

In this chapter, you will learn to:

- Implement Accessibility features in a .NET application
- Identify features of international applications
- Implement Globalization and Localization in a .NET application
- Create a Help System using HTML Workshop
- Implement Help System in .NET applications

Implementing Accessibility Features in a .NET Application

Today, most applications are developed keeping in mind the specific requirement of users with visual, mobility, and hearing disabilities. The applications developed in VC# can be made accessible by setting certain properties of controls used in them. These applications are created with various design guidelines.

Design Guideline for Accessible Applications

Applications can be made accessible by offering built-in options. These options enable a large number of users to use them more effectively. Some of the guidelines followed for designing applications are:

- **Flexibility**: A user interface should be flexible and customizable so that it accommodates a variety of user needs and preferences. For example, the user interface should allow users to select the input and output methods and color preferences. Users should also be allowed to modify text, graphics, and other objects on the screen to suit their requirements.

  Applications should be compatible with the high contrast option because this option improves readability and legibility for visually-impaired users.

  Users may even be allowed to adjust the timing of events according to their needs. For example, users who have difficulty in reading and reacting to briefly displayed information should be able to increase the time for which the information is displayed.
- **Simple and Intuitive**: The user interface should be simple and easy to understand, regardless of a user's experience, knowledge, language skills, or current concentration level. Any unnecessary complexity should be eliminated. Each object on the screen should be assigned a unique and descriptive label to assist users who cannot see its context on the screen.

  The application should interact with other applications in a standard manner. For example, the application should provide a standard mechanism for importing data from and exporting data to other applications.
- **Compatibility with Accessibility Aids**: Applications should be compatible with a variety of techniques or devices for users with sensory limitations. For example, users who are visually impaired may use screen readers. Users who cannot use devices, such as mouse or keyboard to give input may use voice input utilities. These devices and techniques used for making such applications are known as the *Accessibility Aids*.

  Applications need special provisions to convey information about the location and function of various interface elements to the accessibility aids. For example, an accessibility aid that narrates the content of a screen to the user needs to be provided

with information about the various screen elements. This can be done by setting certain properties associated with each element on the screen.

- **Redundant Input and Output Methods**: Applications should provide multiple input and output techniques. For example, each feature of the application should be accessible using a mouse or keyboard. You should also have multiple techniques for conveying the output to the user. Sound should not be used as the only means of conveying output. This will benefit users who have hearing impairments or those who work in noisy environments.

Standard Accessibility Properties in VC#

To make an application accessible, some properties need to be set for the various controls on a form. The following table describes some of the control properties used for making the application more accessible.

Control Property	***Accessibility Considerations***
AccessibleName	*Is used to specify a name for a control by which the accessibility aids identify the control.*
AccessibleDescription	*Specifies a description for a control. This description is reported to the accessibility aids.*
AccessibleRole	*Specifies the use of an element in the user interface and reports it to accessibility aids.*
TabIndex	*Must be set in such a way that it creates a logical navigational path through the form.*
Text	*Must be set in such a way that it provides keyboard access to all the controls on the form. This can be done by using the "&" character to create access keys.*
Font Size	*Should be set to 10 points or larger point, if the user interface does not allow the user to adjust the font size. All controls added to a form have the same font size as the form by default.*
Forecolor	*Is set to default to enable the form to use the user's default system settings for Forecolor.*
Backcolor	*Is set to default to enable the form to use the user's default system settings for Backcolor.*
BackgroundImage	*Should be left blank so as to make the text more readable.*

Control Properties Used for Making Accessible Application

Consider a scenario. You have to create an application considering the fact that it is to be used by partial sighted people. For this application, you have to use some of the standard

accessibility properties, such as Font Size, Forecolor, Backcolor, and BackgroundImage. These properties have to be set in such a manner that the partially sighted people can read the text with minimum effort.

Just a minute:

Which property is used to specify the use of an element in the user interface and reports it to the accessibility aids?

1. *AccessibleDescription*
2. *AccessibleRole*
3. *AccessibleName*

Answer:

2. *AccessibleRole*

Identifying Features of International Applications

The need for international applications could be understood from the following scenario.

A business group has a chain of hotels in different countries. As the geographical area is very vast, the business group is maintaining the information of the hotels through software. This software needs to implement globalization because it must be used by the users across the world and the users across the world use various languages and formatting standards for numbers, currency, and dates.

For this reason applications must be designed to accommodate users from various cultures. International applications should be customizable according to the preferences of various users.

An application refers to a locale to identify users' preferences. The *locale* represents the environment and language of a country. For example, the locale of a user speaking English and belonging to the United States is "en-US". The locale includes information about formats used for representing time and date, symbols and conventions used for representing currency, and the character encoding scheme being used.

The structure of an internationalized application is divided into two blocks:

- **Code block**: Contains the application code or the executable part of an application. This block will remain same for all locales.
- **Data block**: Contains all the resources, such as text and images, used by the user interface (UI). This block is locale-specific, and each locale will have one data block.

Consider a scenario. You want to develop an application that displays a welcome message. For an English speaking user, you can display a welcome message using `MessageBox.Show()` method and writing the text "Welcome", as an argument in the show method. In the case of an internationalized application, the text messages in the show method will vary according to the locale of the user. For this reason, the text message to be displayed is stored in another file called a resource file. For each locale, there is a corresponding resource file.

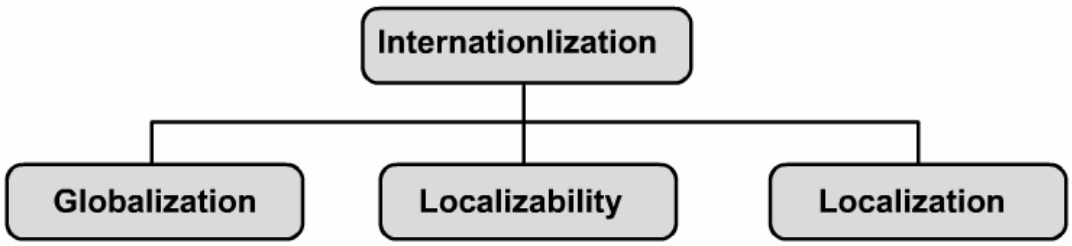
The process of making an application ready for international customers is called *internationalization*. This process includes three phases. These are:

- **Globalization**: Involves writing the executable code for an application in such a way that makes it culture-neutral and language-neutral. While incorporating globalization, you must ensure that the culture-specific details are not hard-coded into the executable code. The primary task in this phase is to identify the various locale-sensitive resources used by the application and to isolate these resources from the executable code.
- **Localizability**: Involves testing an application, which has been globalized to ensure that its executable code is independent of the culture and language specific data. This

is called *localizability testing*. The focus of localizability testing is to check how the application adapts itself to various locales.

- **Localization**: Involves the customization of an application for a specific locale. One major task in this phase is the translation of resources identified during the globalization phase. During this phase, various resources, such as images and text, for the designated locale are created. For example, to localize an application for French users, all text messages should be translated to French.

The following figure displays the three phases involved in developing an internationalized application:



The Phases of Internationalization

Factors Related to Creating International Applications

Application developers need to attend to locale considerations when they develop international applications because negligence of these considerations can lead to issues. The factors that need to be considered while designing an application are:

- Language Issues
- Formatting Issues
- String-related Issues
- User-interface Issues

Language Issues

Languages around the world differ in display, alphabets, grammar, and syntactical rules. For example, some languages such as Arabic is written from right-to-left, whereas other languages are written from left-to-right. Some languages include uppercase and lowercase characters, whereas others do not even have the concept of uppercase and lowercase. Languages differ in the number of characters, storage requirements, keyboard layouts, and code representations. This diversity makes it difficult to share data between cultures. It is even more difficult to create a multilingual user interface.

Formatting Issues

Formatting is the main source of differences in applications designed for multiple languages or cultures. Formatting differences may arise in the addresses, currency, dates, numerals, telephone numbers, time, and units of measure of various languages.

String-Related Issues

When developing international applications, programmers must consider issues related to strings. When strings are translated from one language to another, the translated strings may be longer than the original strings. In strings, the order of alphabets varies in different languages. This causes major problems in sorting and comparison of strings. Issues also arise when strings are concatenated because these strings may convey different meanings in different languages.

User-Interface Issues

Various user interfaces are associated with the design of an international application. The size of user interface elements should be larger than that required for accommodating English strings. This is because strings in other languages are usually longer than strings in the English language. When messages grow in size as a result of translation to another language, they should be allowed to wrap to subsequent lines. You should also ensure that all the access-key and shortcut-key combinations are available on international keyboards. This is because every language has a different keyboard layout, and some characters do not exist on all keyboards.

Creating International Applications

To create international applications, implement the following:

- Work with multiple culture codes
- Culture-specific formatting
- Character encoding
- Right-to-Left display
- Validate Non-Latin user input
- Culture-specific classes and methods

Working with Multiple Cultures

For developing an application that works for multiple cultures, you need to include support for culture-specific coding rules that can handle the application resources and culture-specific formatting rules.

Culture Code

Every locale is identified by its culture, which in turn is identified by culture code. The format of the culture code is as follows:

```
<Language code>-<Country/Region code>
```

- **Language code**: Represents a two-letter representation of a language. It is always written in lowercase. For example, the language code for Arabic is "ar" and that for Russian is "ru".
- **Country/Region code**: Represents a two-letter representation of a country or a region. It is always written in uppercase. For example, the country code for Russia is "RU" and that for France is "FR".

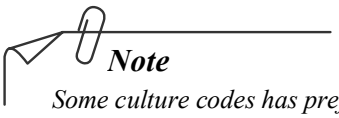
The following table lists some culture codes and their specifications.

Culture Code	***Specifications***
En	*English language, no region*
en-CA	*English language, Canada*
fr-FR	*French language, France*
De	*German language, no region*
zh-CN	*Chinese language, China region*
de-DE	*German language, Germany region*

List of the Culture Code

Cultures are of two types:

- **Neutral culture**: Signifies a culture that is linked only to a language and not to a country. For example, fr is a neutral code.
- **Specific culture**: Signifies a culture that is linked to both the country and its language. For example, fr-FR is a specific code.



Note

Some culture codes has prefixes associated with them. Prefixes represent the script of the culture. For example, the "Lt" prefix specifies that the script of the culture is Latin.

Retrieving and Setting the Current Culture and Current UI Culture

The Current Culture and Current UI Culture values determine the resources that are loaded for an application and how information such as currency, numbers, and dates is formatted. The resources loaded are determined by the UI culture setting, and the formatting options are determined by the culture setting.

In VC# applications, the `CultureInfo` class holds the information about a culture and how it should interact with your application. This class contains the information about the type of calendar, currency formatting, and date formatting.

The following table describes some of the commonly used properties of the `CultureInfo` class.

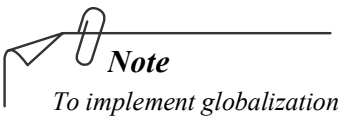
Property	***Description***
CurrentCulture	*Returns an instance of the CultureInfo class that represents the culture for the current thread.*
CurrentUICulture	*Returns an instance of the CultureInfo class that represents the culture for culture-specific resources.*
DisplayName	*Returns the full name of the culture in the <Language code>-<Country/Region code> format.*
EnglishName	*Returns the English name of the culture in the <Language code>-<Country/Region code> format.*
Name	*Returns the name of the culture in the <Language code>-<Country/Region code> format.*
LCID	*Returns the culture identifier for the instance of the CultureInfo class. Every culture has an identifier. For example, 0x0407 is the identifier for "de-DE" (German-Germany) culture.*

The Common Properties of CultureInfo Class

The following table describes some of the commonly used methods of the `CultureInfo` class.

Method	***Description***
GetCultures	*Returns a list of all the supported cultures.*
GetFormat	*Returns an object that represents the format of a specified type. For example, this method will return an object representing the format of the date when the specified type is date/time.*

The Common Methods of CultureInfo Class



Note

To implement globalization and localization in an application you need to include the Globalization namespace in your project. The Globalization namespace can be included by using the following code:

```
using System.Globalization;
```

The following code snippet shows how to use the `CultureInfo` class:

```
using System.Globalization;
CultureInfo my_CI = new CultureInfo("ru-RU");
```

In the preceding code snippet, a new `CultureInfo` class instance by the name, my_CI, is created that sets the culture of the instance to "ru-RU ".

An application automatically reads the system culture settings and implements them. You can set the culture of an application programmatically by setting the CurrentThread.CurrentCulture property to a new instance of the `CultureInfo` class. The `CultureInfo` class constructor takes a string as a parameter that represents the desired culture code.

The following is an example that sets the current culture to English-Canadian:

```
System.Threading.Thread.CurrentThread.CurrentCulture = new
System.Globalization.CultureInfo ("en-CA");
```

To get the `CultureInfo` class that represents the `CurrentCulture`, you can use the following code:

```
using System.Globalization;
CultureInfo myCulture;
myCulture = CultureInfo.CurrentCulture;
```

Set the current UI culture in the same way as you set the current culture. The following is an example that sets the current UI culture to French Canadian:

```
System.Threading.Thread.CurrentThread.CurrentUICulture = new
System.Globalization.CultureInfo ("fr-CA");
```

The CurrentUICulture can be retrieved by accessing the CultureInfo.CurrentUICulture property, as shown in the following code:

```
using System.Globalization;
CultureInfo myCulture;
myCulture = CultureInfo.CurrentUICulture;
```

Culture-Specific Formatting

Consider a scenario, where Nancy is writing an application for a client that collaborates with a group in China. According to the requirements given to her, formatting for most elements in the application must be China-specific, but the currency used should be U.S. dollars.

Nancy needs to use the members of the `CultureInfo` class. These members are DateTimeFormat, NumberFormat, and TextInfo and they provide specific formatting combinations.

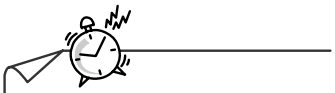
The following table describes the members of the `CultureInfo` class.

Member	***Description***
DateTimeFormat	*Contains information about how dates and times are formatted*
NumberFormat	*Contains information about how numbers and currency are formatted*
TextInfo	*Contains information about how text is formatted*

The Members of CultureInfo Class Used for Specific Formatting Task

The properties of the `CultureInfo` class members can be changed for specific culture formatting needs. For example, to create a culture setting that uses China-specific formatting for most of the elements, but uses the dollar symbol for currency, you need to create a CultureInfo object for the Chinese culture. You need to then modify it to change the currency symbol to "$", as shown in the following example:

```
using System.Globalization;
using System.Threading;
…
CultureInfo myCulture = new CultureInfo ("zh-CN");
myCulture.NumberFormat.CurrencySymbol = "$";
Thread.CurrentThread.CurrentCulture = myCulture;
```



Just a minute:

Which property of the `CultureInfo` *class will return an instance of the* `CultureInfo` *class that represents the culture for culture-specific resources?*

Answer:

CurrentUICulture

Character Encoding

All characters are internally represented in the numeric format. A unique number is assigned to every character. *Encoding* is the scheme for representing characters in a numeric format. The characters from various languages are grouped into a set called a *character set*.

There are two very commonly used character sets. They are:

- **American National Standards Institute (ANSI)**: Includes 256 characters and punctuations. In ANSI, every character is represented using one byte (8-bit). The ANSI character set is not sufficient to support all languages.
- **Unicode**: Every character is represented using two bytes (16-bit). Unicode supports character sets of all the major international languages.

The advantage of Unicode is that you can easily convert data from one format to another. There are two major disadvantages of Unicode. They are:

- **File size**: When characters are represented in Unicode, the file size is doubled.
- **Compatibility**: Many computer systems around the world do not support Unicode.

You can use the *Unicode Transformation Format (UTF)* to make it compatible with systems that do not support Unicode. UTF encodes each Unicode as a byte value. For example, UTF-8 converts the value of a character into a byte value.

Convert Existing Encodings

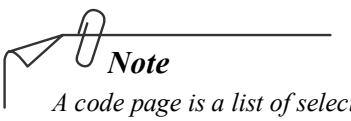
The .NET Framework allows you to convert data from other formats to Unicode data. It also allows you to convert Unicode data into other character encoding formats. This can be done using the `Encoding` class.

The `Encoding` class is found in the `System.Text` namespace and cannot be directly instantiated. An instance of this class can be obtained by using the static method, `Encoding.GetEncoding()`.

To create an instance of a particular encoding that specifies a particular character code page, you must use the `GetEncoding()` method, as shown in the following code snippet:

```
using System.Text;
. . .
Encoding greekEncoding;
greekEncoding = Encoding.GetEncoding(1253);
```

The preceding code snippet creates an instance for the encoding scheme represented by code page 1253.



Note

A code page is a list of selected character codes in a certain order. Languages across the world have different writing systems and therefore different code pages are used to represent these languages. For example, the code page 1250 provides the characters for Latin writing systems including English, German, and French. Code page 1253 provides character codes that are required in the Greek writing system.

When an instance of a particular encoding has been created, it can be used to convert characters in that encoding to the Unicode format. To convert an existing encoding to Unicode, you can use the `Encoding.Convert()` method. This method requires three parameters, namely, a source encoding, a target encoding, and an array of bytes, which represent the data to be converted.

In addition, this method returns an array of bytes that contains data in the target-encoding format. The target encoding in this case, can be specified as an instance of the `System.Text.UnicodeEncoding` class.

The following is an example of the process of converting data encoded in code page 1253 to data encoded in Unicode:

```
using System.Text;
...
byte[] unicodeData;
Encoding mySourceEncoding;
UnicodeEncoding myTargetEncoding = new UnicodeEncoding();
mySourceEncoding = Encoding.GetEncoding(1253);
// The data to be converted is represented as a byte array by
// the variable name mySourceData
unicodeData = Encoding.Convert(mySourceEncoding, myTargetEncoding,
mySourceData);
```

The `GetBytes()` method can be used to create a byte array from a char array or string, as shown in the following code:

```
using System.Text;
byte[] ByteData;
Encoding myEncoding;
myEncoding = Encoding.GetEncoding(1253);
//Data in the character format is contained in a variable
//called charData
ByteData = myEncoding.GetBytes(charData);
```

Similarly, the `GetChars()` method can be used to convert an array of bytes to an array of chars, as shown in the following example:

```
using System.Text;
byte[] CharData;
UnicodeEncoding myEncoding = new UnicodeEncoding();
//The byte data is contained in a variable called ByteData
CharData = myEncoding.GetChars(ByteData);
```

Implementing Right-to-Left Display

When scripts that read from right-to-left instead of left-to-right are to be displayed on a form, not only the script-flow, but also the visual alignment of form elements must be reversed. To enable implementation of a right-to-left user interface, the RightToLeft property of a form must be set to Yes. This property can also be set for the various controls on a form.

In addition, the RightToLeft property can be set to values, No or Inherit. No is the default value. If this property is set to Inherit for a control, its RightToLeft property is determined by the RightToLeft value of its parent control.

When a form's RightToLeft property is set to Yes, formatting of each control on the form becomes a mirror image of itself. The caption of the form is right aligned. vertical scroll bars are displayed on the left side, and horizontal scroll bars are initialized with the slider right aligned. The CheckAlign property of the check boxes, alignment of items in the list boxes and combo boxes, and the tab buttons are all reversed. The content contained in a RightToLeft control, however, is not reversed.

For example, if there is a TextBox control on the form with its Text property set to "Hello", the text is still displayed as "Hello" and not "olleH".

Validating Non-Latin User Input

When developing an application for different cultures, you might need to incorporate techniques for validating non-Latin input into your application. This can be difficult

particularly if you are not familiar with the character system used in a particular locale and, therefore, unable to determine the category to which the keystrokes belong.

The Char structure exposes some validation methods that can be used to determine the category of a non-Latin alphabet input in the same way as a Latin alphabet input. The `Char.IsDigit()`, `Char.IsLetter()`, and other methods return appropriate values no matter what input character set is used. If these methods are used for validation, the same validation code can be used to validate international input in various languages without any special modification.

Culture-Specific Classes and Methods

Different cultures use different conventions for displaying currency, dates, time, numbers and other information. Classes that automatically format information according to the culture setting are called *culture-specific* classes. The `System.Globalization` namespace contains classes that can be used to modify the display of culture-specific values, namely, DateTimeFormatInfo, Calendar, and NumberFormatInfo.

In addition to culture-specific classes, there are some culture-specific methods like, `IFormattable.ToString()`, `Console.Writeline()`, `String.Format()`, `MonthName()` and `WeekDayName()`. For example, the following code shows how you can use the `ToString()` method to format currency for the current culture:

```
using System.Threading;
using System.Globalization;
//Displays a number with the culture-specific currency //formatting
int MyInt = 100;
MessageBox.Show(MyInt.ToString("C",
Thread.CurrentThread.CurrentCulture));
```

Implementing Globalization and Localization in a .NET Application

In this section, implementing globalization and localization in Windows Forms application will be discussed. You will learn about the efficient approach for creating a globalized application. When creating a localized application, customizing the user interface is one of the primary tasks, which is a seemingly easy process in .NET Framework because it provides various methods for localizing an application. When creating a globalized or localized application, there is additional data that is used by the application. This data is stored in the form of resource files. The resource files of an application can be deployed in the form of packages. This section discusses such resource files and resource assemblies used while creating an international application.

Globalizing Windows Forms

When creating a globalized application, you could take the approach of writing completely different sets of source code for each culture. However, this would not be a very efficient approach. An efficient approach would be to write one set of source code and build ability to customize the application for different cultures into the solution. This is done in VC# using the concept of globalization.

Globalization is the process of identifying all localizable resources in the application and separating them from the executable code so that the localizable resources can be modified easily.

For example, consider a form with a Label control named DateLabel, whose Text property is set as:

```
DateLabel.Text = "5/23/1994";
```

In this case, the Text property of the DateLabel control is set to a constant value and therefore, will always be displayed as is, whatever be the current culture-settings.

Different cultures may use different formats for displaying dates. Therefore, you need to set the Text property of the DateLabel in such a way that it allows the application to automatically display the date in a format corresponding to the current culture settings. You can set the Text property of the label by using the Format function, as shown in the following example:

```
DateTime MyDateTime = DateTime.Parse("May 23, 1994");
DateLabel.Text = (MyDateTime.ToString("d"));
```

Localizing Windows Forms

Localization is the process of customizing an application for a given culture. Localization consists primarily of translating the user interface. Localizing forms with the .NET Framework is nearly an effortless process.

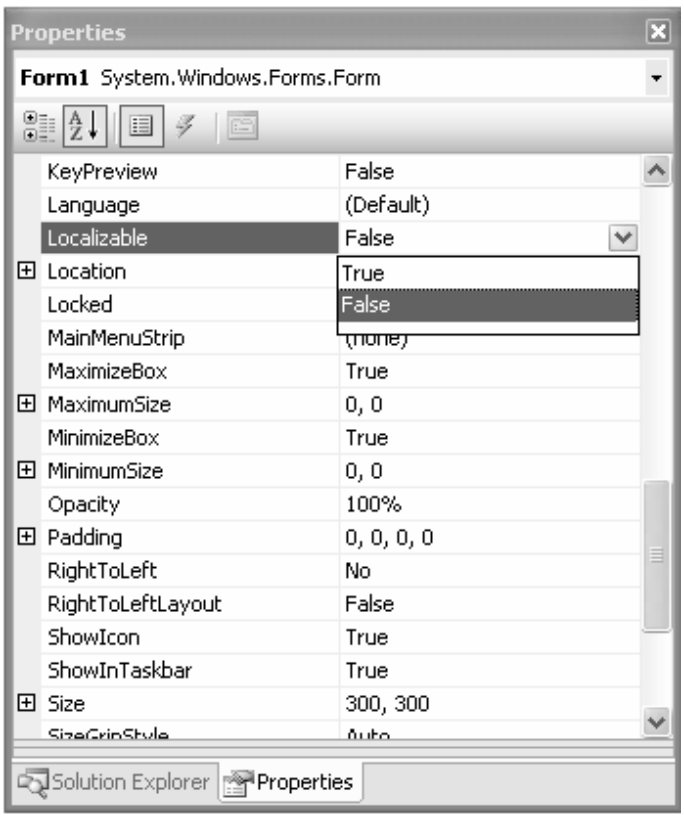
A form can be localized by setting its Localizable property to True. When this property is set to True, Visual Studio .NET automatically handles the creation of various resource files. The number of resource files created depends on the number of cultures or languages for which the form is localized.

For each language or culture the application needs to support, a user interface needs to be created. Different user interfaces can be created for various cultures by changing the Language property of the form. When this property is set to default, any of the form's properties or controls can be edited to create a default UI culture. For creating a localized version of the form, the Language property must be set to any value other than the default. Visual Studio .NET creates a resource file for the new language and stores the values that you set for the user-interface in that file.

Localized user interface elements are usually strings. However other elements, such as form layouts, display formats for dates and times, currency, numbers, and graphics with local content, can also be localized.

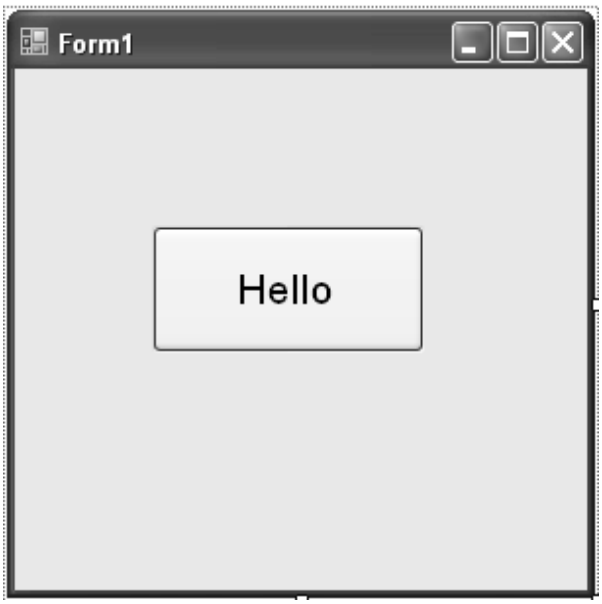
For example, you can design an application to localize a form for French (France region) and German (Germany region) cultures. To design the application, you need to perform the following steps:

1. Set the **Localizable** property of the form to **True**, as shown in the following figure.



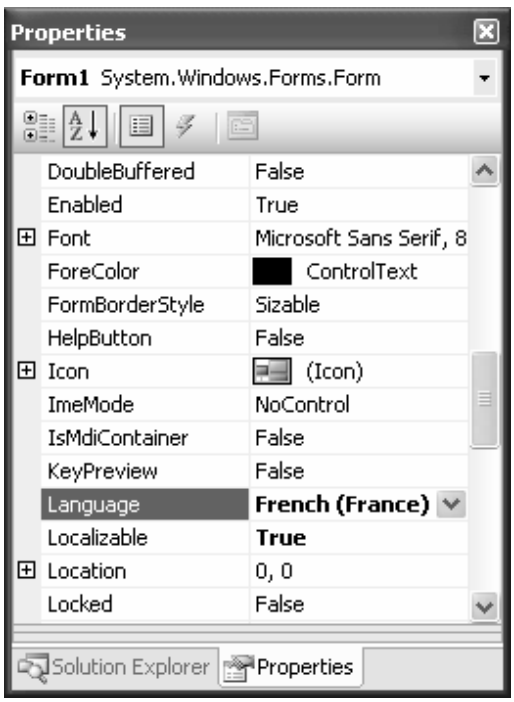
The Localization property

With the **Language** property of the form set to **Default**, design the default user interface. For example, when you add a button control to the form and set its **Text** property to "**Hello**", the form will appear, as shown in the following figure.



The Form for Default Culture

2. Set the **Language** property of the form to "**French (France)**", as shown in the following figure.



The Setting of the Language property to French (France)

Notice, when you change the **Language** property of the form, the form created earlier is not overwritten. Independent resource files are created for multiple language versions of the form.

3. Modify the user interface of your form for French by setting the **Text** property of the button to "**Bonjour!**". The form will appear, as shown in the following figure.

The Form for French Culture

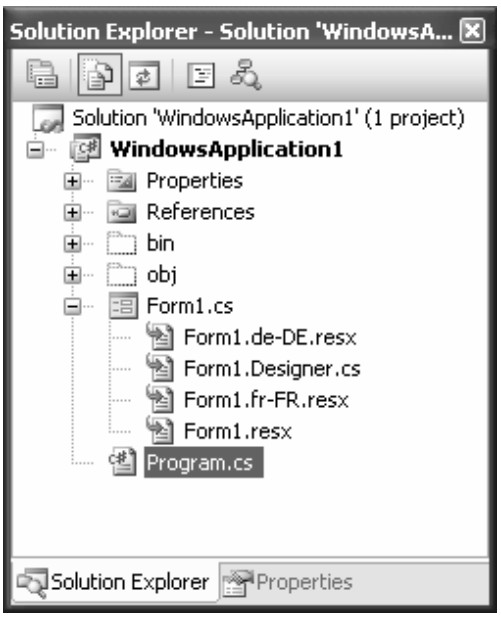
4. Set the **Language** property of the form to "**German (Germany)**".
5. Modify the user interface of your form for German by setting the **Text** property of the button to "**Hallo**".

The form will appear, as shown in the following figure.

The Form for German Culture

6. Build your application.

The resource files for each language/culture are automatically created by Visual Studio .NET. You can view the resource files for a form by clicking the Show All Files button in Solution Explorer, and then expanding the node next to your form. Under your form, you will see the resource files for each language that your form was localized for, as shown in the following figure.



The Culture-specific Resource Files

When the application is run, it will automatically load the appropriate version of the form. If no resource files exist for the current culture, the default user interface will be displayed.

Apart from the executable files, an application also contains certain application resources such as text files. These resources are stored in the resource files.

Application Resources

A resource is any non-executable data that is logically deployed with the application. A resource might be a file containing string messages to be displayed in an application, either as error messages or as part of the user interface. Examples of resources include strings displayed in the user interface based on the culture settings of the system or a set of images. Packaging your localizable data in a resource file allows you to change the data without recompiling your entire application.

Information is stored in a resource file in the Key/Value format. For example, to store the welcome message you must assign the message to a key. The following code snippet shows how you can store "`Welcome to VC#`" message inside a resource file.

```
WelcomeText = "Welcome to VC#"
```

In the preceding code snippet `WelcomeText` is the key and "`Welcome to VC#`" is the value. There are specific naming conventions to follow while naming resource files. The resource file for the default culture should be filename.txt or filename.ResX.

Names of resource files for any other culture will have the following syntax:

```
filename.culture-name.ext
```

In the preceding syntax:

- `filename`: Represents the name of the file.
- `culture-name`: Represents the culture for which resource file is designated.
- `ext`: Represents the extension of the resource file, which can be either .txt or .ResX.

For example, myResources.fr-CA.resx is the resource file that corresponds to the culture "fr-CA". When resource files are created in the text form, they need to be converted into a binary format recognized by the .NET Framework.

Generating Resource File

You can use the `Resource File Generator (ResGen.exe)` command prompt utility to convert a resource file in the text format into the binary format. The syntax is as follows:

```
RESGEN [/compile] [inputfilename.extension]
[outputFilename.extension]
```

In the preceding syntax:

- `inputfilename.extension`: Represents the name of the resource file in the text format.
- `outputFilename.extension`: Represents the output file that is generated after conversion. If this parameter is missing, the name of the output file will be the same as the input file, except for the extension, which will be .resource.

Resource Assemblies

Resource assemblies are assemblies that contain only resources. These assemblies are useful when resources in a program need to be updated frequently, but you do not want to recompile the entire solution every time a resource is updated. By placing all of your

resources into a resource-only assembly, you can update the assembly as and when you need to, without having to recompile your application.

When creating an application for multiple cultures, different sets of resources are required to be created for various cultures. *Satellite assemblies* are resource assemblies that include separate resources for different cultures. These assemblies allow a different set of resources to be loaded automatically based on the CurrentUICulture setting of the thread.

Creating Resource-Only Assemblies

To create a resource-only assembly, you need to perform the following steps:

1. Create a new VC# project with the **Empty Project** template.
2. From the **Project** menu, choose **Add Existing Item** to add your resource files to your project.
3. In **Solution Explorer**, right-click your project and choose **Properties**. The **Property** pages open.
4. In the **Output Type** drop-down menu, change the output type of your project to **Class Library**. This will cause your assembly to be compiled as a .dll file.
5. From the **Build** menu, choose **Build <your project>** where <your project> is the name of your project. The resources you added to the project, are compiled into the assembly.

Creating Satellite Assemblies

Satellite assemblies can be created effortlessly in Visual Studio .NET by adding alternate sets of appropriately named resource files into your application. The rest is handled by Visual Studio .NET at the time of compilation.

A resource file must follow a specific naming culture to be incorporated into a satellite assembly. The name of a resource file for a specific culture is the same as the name of the resource file for the default culture, and the culture code is inserted between the base name and the extension.

Therefore, if you have a default resource file named MyResources.resx, a resource file containing the alternate resources for neutral German (i.e. German culture with no region specified; culture code “de”) user interface would be named MyResources.de.resx. And a version of the resource file containing French resources specific to Canada (culture code “fr-CA”) would be named MyResources.fr-CA.resx.

When these alternate versions of the resource file are added to your solution, Visual Studio .NET will compile them into satellite assemblies, and a directory structure for them

will be created. At runtime, the culture-specific resources contained in these files will be located automatically by the common language runtime.

To create satellite assemblies, you need to perform the following steps:

1. Create alternate versions of your resource files specific to locales where your application will run.
2. Name your resource files correctly for their specific culture. Names for resource files for satellite assemblies must contain the culture code separated by periods between the base name and the extension. The base name and extension must be the same as the resource file for the invariant culture.
3. From the **Project** menu, use **Add Existing Item** to add these files to your application.
4. From the **Build** menu, choose **Build Solution** to build your solution.
5. Culture-specific resource files are automatically compiled into satellite assemblies and placed in an appropriate directory structure.

Retrieving Resources at Run Time

At run time, the `ResourceManager` class can be used to retrieve resources from an assembly. Each instance of this class is associated with an assembly that contains the resources.

An instance of the `ResourceManager` class can be created by specifying two parameters in its constructor, the base name of the embedded resource file and the assembly in which the file is found.

The base name of a resource file is the name of the namespace that contains the file and the filename without any extensions. For example, a resource file named myResources.fr-CA.resx in a namespace myNamespace would have a base name of myNamespace.myResources.

The assembly parameter refers to the assembly in which the resource file is located. Before using the assembly name as a parameter to the `ResourceManager` constructor, the assembly must be loaded by using the Assembly object in the `System.Reflection` namespace, as shown in the following code:

```
using System.Resources;
. . .
System.Reflection.Assembly myAssembly;
myAssembly = System.Reflection.Assembly.Load ("ResourceAssembly");
ResourceManager myManager = new
ResourceManager("ResourceAssembly.myResources", myAssembly);
```

If however, the object that is creating the ResourceManager is also contained in the same assembly, the following code may be used to create the ResourceManager:

```
ResourceManager myManager = new ResourceManager
("myNamespace.myResources", this.GetType().Assembly);
```

When you have created an instance of the `ResourceManger` class, it can be used to retrieve strings and objects contained in the resource file. To retrieve a string, the `ResourceManager.GetString()` method can be used. This method requires you to specify the name of the string resource you want to retrieve as a parameter.

Images and other objects can also be retrieved from a resource file by using the `ResourceManager.GetObject()` method. This method returns an object corresponding to the name provided. The object returned must be explicitly converted to the correct type. The following example demonstrates how to retrieve an image using the ResourceManager:

```
System.Drawing.Image myImage;
myImage = (System.Drawing.Image)myManager.GetObject("ImageResource");
```

When your application has several satellite assemblies for various cultures, the ResourceManager class automatically loads the correct resources based on the CurrentUICulture setting. Therefore, if you have an embedded resource file named myResources.resx and a culture-specific resource file named myResources.fr.resx, the ResourceManager will load resources from myResources.resx under most conditions. However, the ResourceManager will load from myResources.fr.resx when the CurrentUICulture is set to "fr".

Just a minute:

Which command prompt utility is used to convert a resource file from the text format into the binary format?

Answer:

Resource File Generator (ResGen.exe) Utility

Creating a Help System Using HTML Workshop

While using an application, a user might need help on performing an operation or may require information about a button or a dialog box. To assist the user while using the application, it is a good practice to provide a help system within the application. You can create a help system by using HTML Help Workshop. In a VC# application you can incorporate a help system by using various methods, such as Context-sensitive help, pop-up help and tool tips.

Help Files

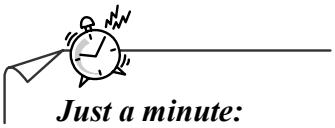
A *help system* comprises of several help files and components that are created using HTML Help Workshop. HTML Help Workshop (hhw.exe) is a help-authoring tool provided by Microsoft for creating help systems. Some files that make up a help system are:

- **HTML Files**: Contain the text that appears on each page in your help system or Web site
- **Graphics and Multimedia Files**: Contain links to graphic, sound, video, animation, and other multimedia files
- **Help Project Files**: Contain information about the location of help topics, graphics, and other files.
- **Contents Files**: Contain the information that appears in the table of contents for the help system or Web site
- **Index Files**: Contain the information that appears in the index for the help system or Web site

The HTML Help Workshop allows you create a help system with standard features of any help system. The features are:

- **Table of Contents**: Provides users with a hierarchical view of the various help topics
- **Index**: Helps users to quickly search for information they need

The files in the help system are managed by creating a help project file.



Just a minute:

What is the extension of a compiled help project file?

Answer:

.chm

Activity: Building a Help System

Problem Statement

You have created an application for a retail store. The application maintains the records of its customers. Before making the final delivery of the application, you want to implement the help system in the application. You need to create the help system using the HTML Help Workshop.

Solution

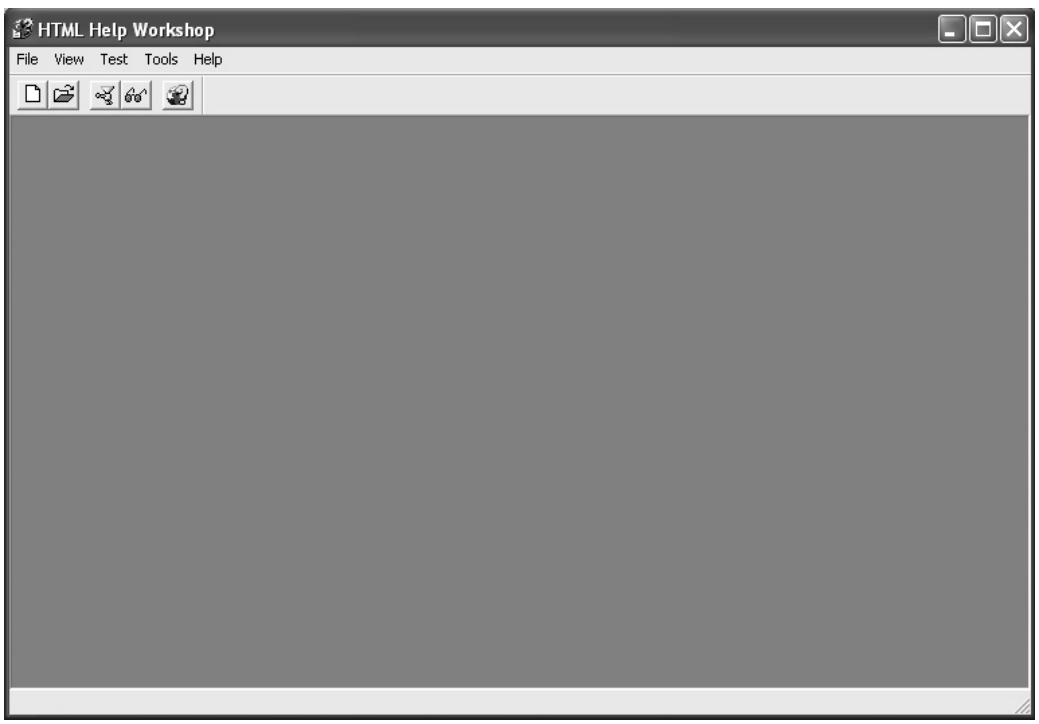
To build help system using the HTML Help Workshop, you need to perform the following tasks:

1. Create a project file.
2. Create the HTML topic pages.
3. Create an index.
4. Create a table of contents.
5. Compile and verify the project file.

Task 1: Creating a Project File

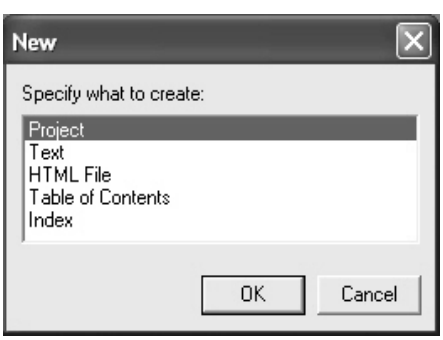
To create a project file, you need to perform the following steps:

1. Select **Start→All Programs→HTML Help Workshop→HTML Help Workshop** to open the **HTML Help Workshop** window, as shown in the following figure.



The HTML Help Workshop Window

2. Select **File→New**. The **New** dialog box appears, as shown in the following figure.



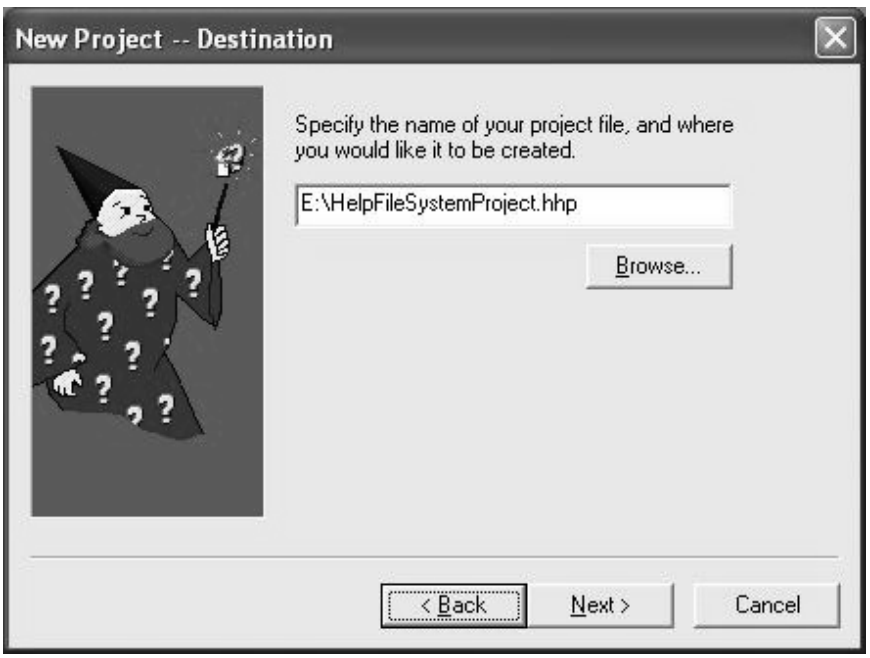
The New Dialog Box

3. Ensure that the **Project** option is selected. Click the **OK** button to open the **New Project** wizard for creating a new project, as shown in the following figure.

The New Project Wizard

4. Click the **Next** button to move to the **Destination** screen of the **New Project** wizard.

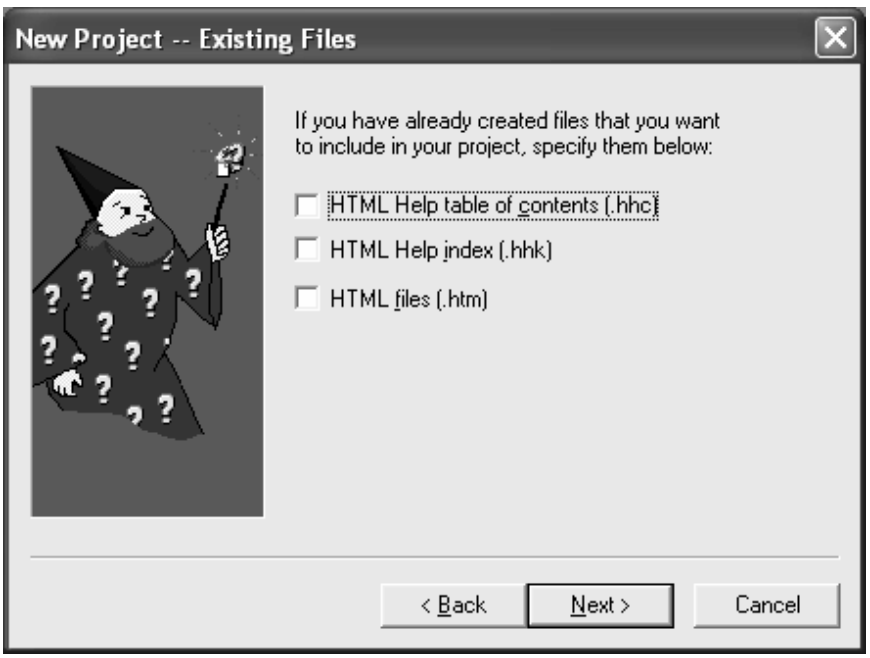
5. Enter the name of the project and the path where you want to create the project, as shown in the following figure.



The Destination Screen

Note that the file has an extension .hhp.

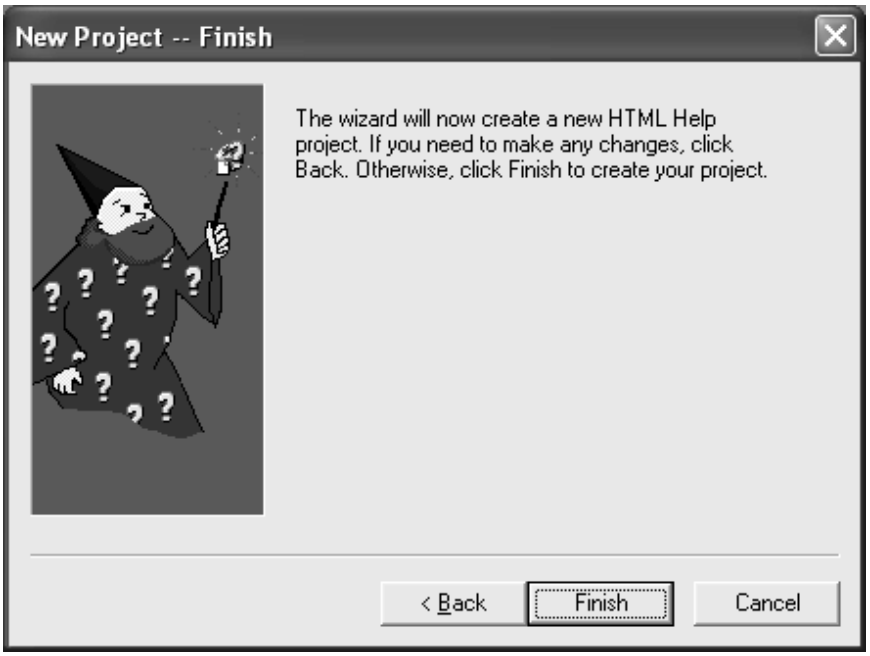
6. Click the **Next** button. The **Existing Files** screen is displayed, as shown in the following figure.



The Existing Files Screen

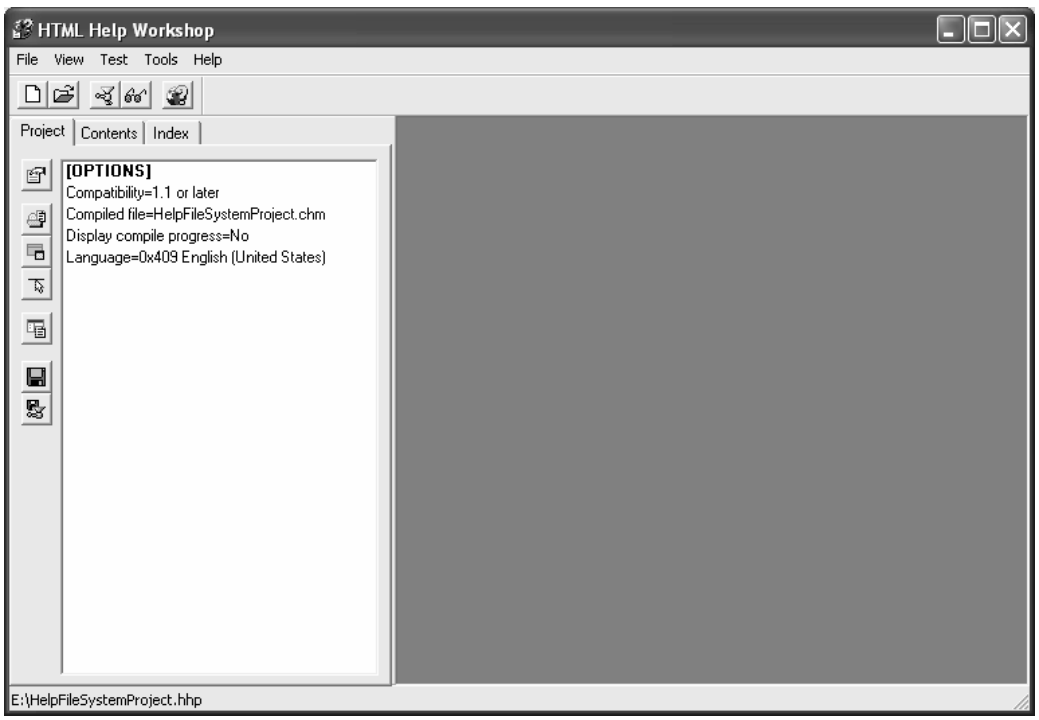
If you have some existing files that you want to include in your help project, select the types of files and then, click the **Next** button to move to the next screen.

7. Click the **Next** button to display the **Finish** screen, as shown in the following figure.



The Finish Screen window

8. Click the **Finish** button to create a new project file. The following window is displayed.



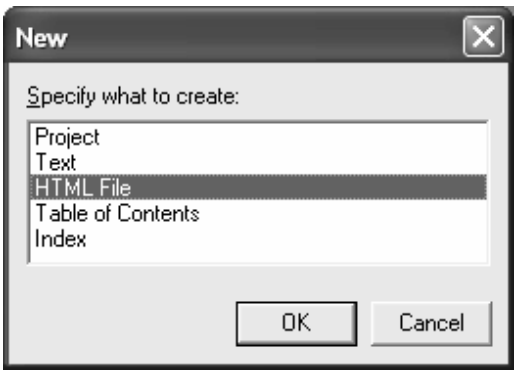
The Project File with the Active Project Tab

Notice, the **Project** tab is active by default.

Task 2: Creating the HTML Topic Pages

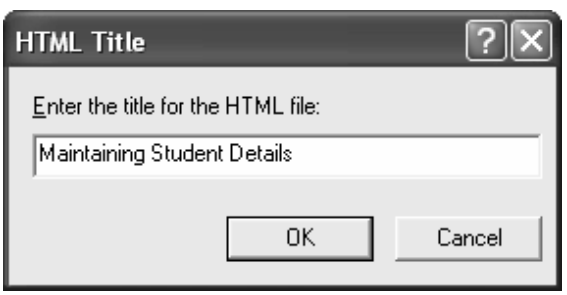
When the project file has been created, the HTML topic pages need to be created for each help topic. Topic pages are created in the HTML format. To create topic pages, you need to perform the following steps:

1. Select **File→New**. The **New** dialog box appears, as shown in the following figure.



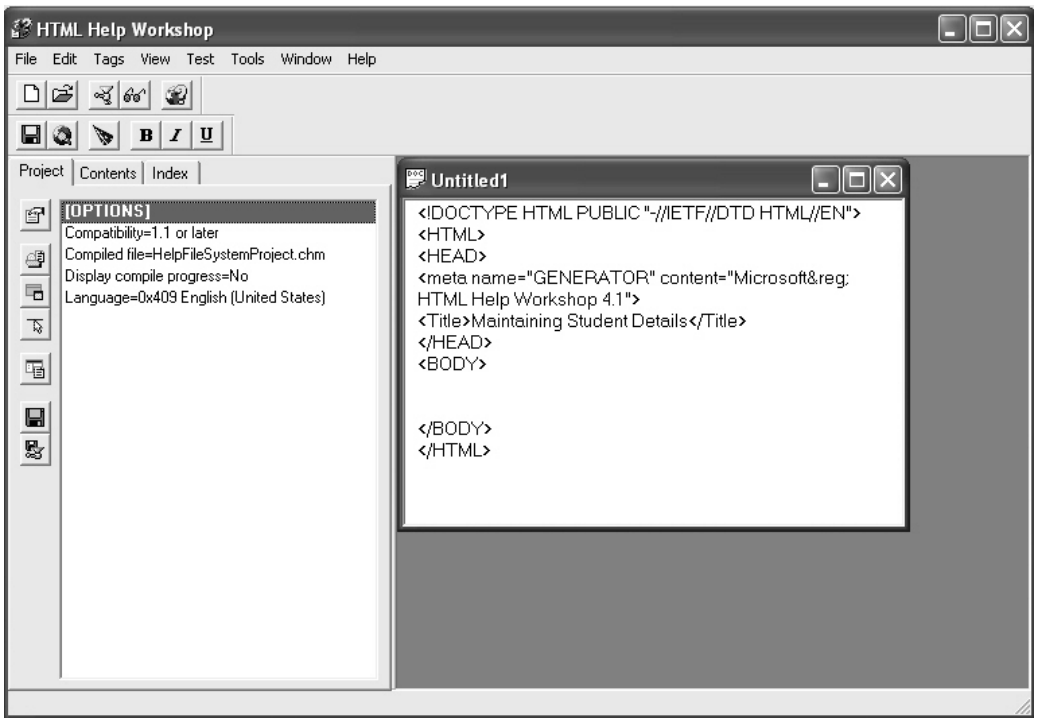
The New Dialog Box for Creating an HTML File

2. Ensure that the **HTML File** option is selected and click the **OK** button.
3. Type **Maintaining Student Details** in the **HTML Title** dialog box, as shown in the following figure.



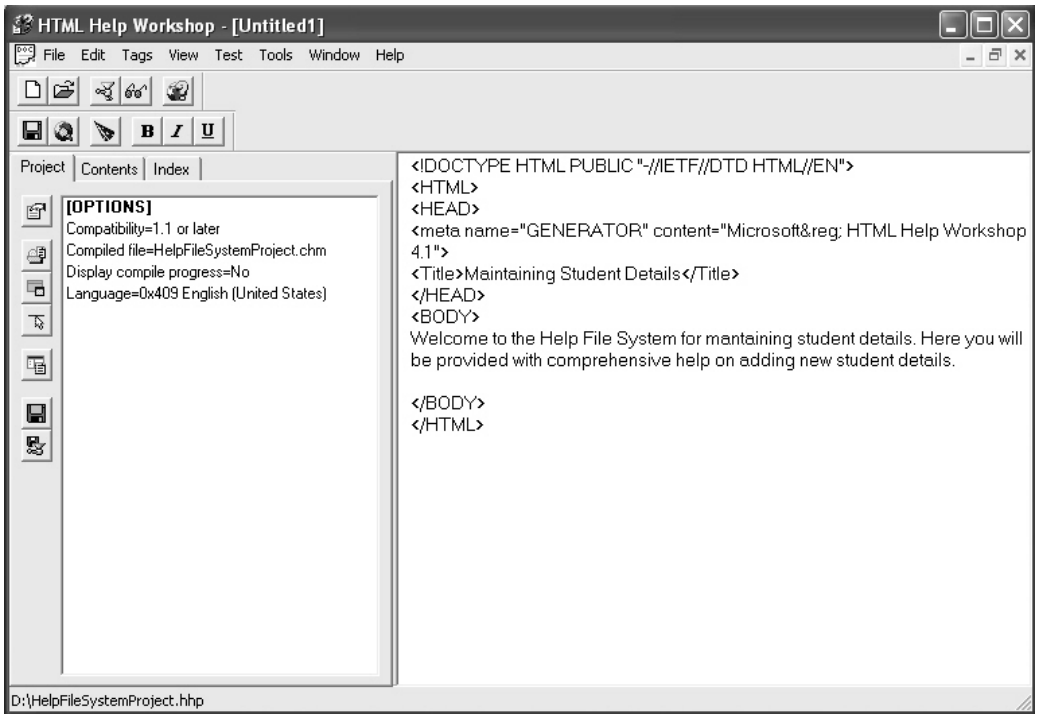
The HTML Title Dialog Box with Title

4. Click the **OK** button.
   The outline of the HTML page is displayed, as shown in the following figure.



The HTML Page

5. Write the help text within the **<BODY> </BODY>** tag in the HTML source page in the right pane of the window. The HTML page that appears will look similar to the following figure.



The Created HTML Page

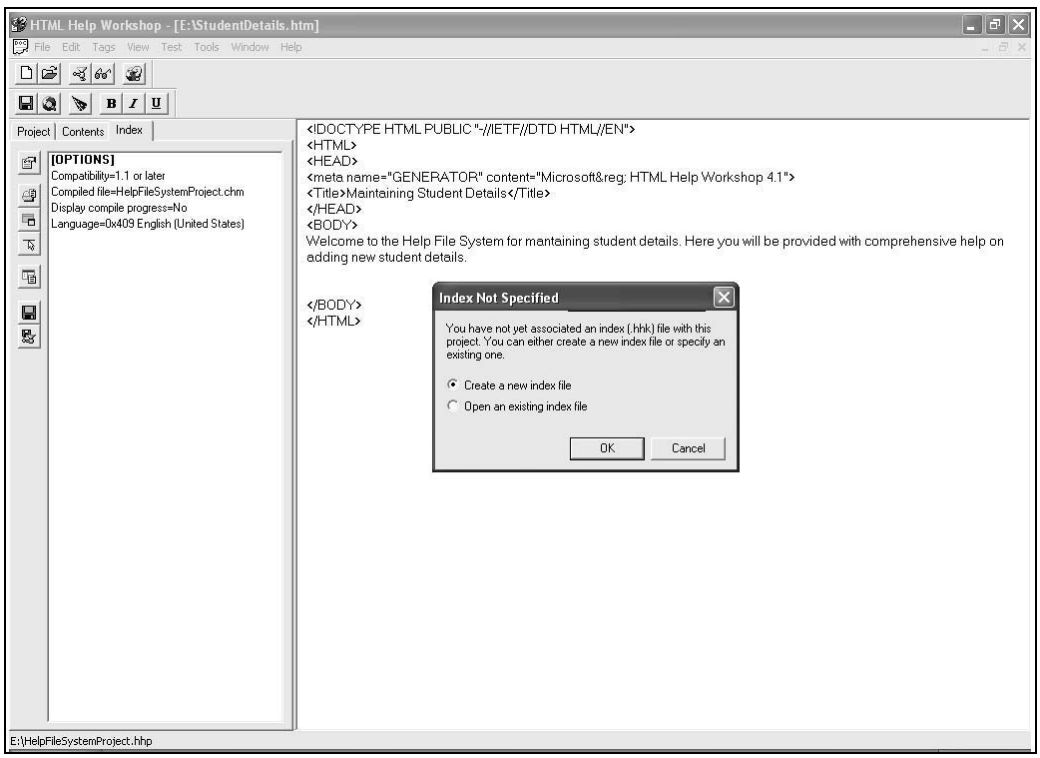
6. Click the **Save** icon on the Toolbar to save the HTML file.
7. Type **StudentDetails** in the **File name** text box of the **Save As** dialog box and click the **Save** button.

Repeat steps 1-7 for each help topic page you want to add to the project.

Task 3: Creating an Index

To create an Index, you need to perform the following steps:

1. Click the **Index** tab. The **Index Not Specified** dialog box appears, as shown in the following figure.

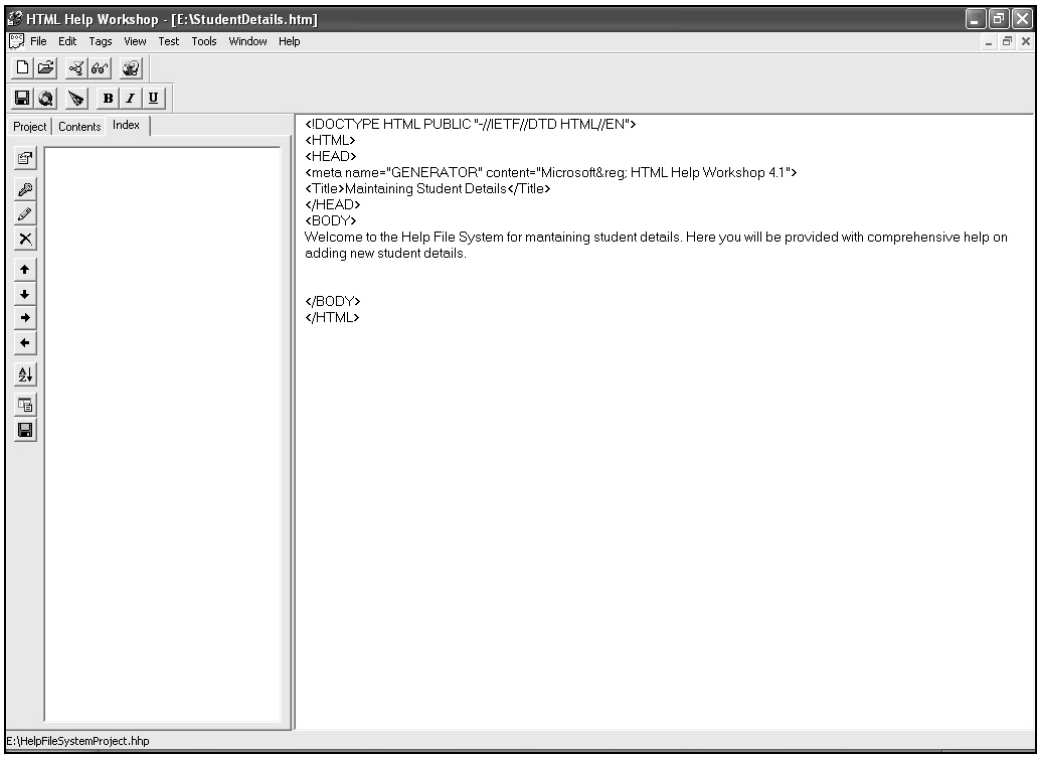


The Index Not Specified Dialog Box

The **Index Not Specified** dialog box prompts the user to select an option, either to create a new index file or open an existing one.

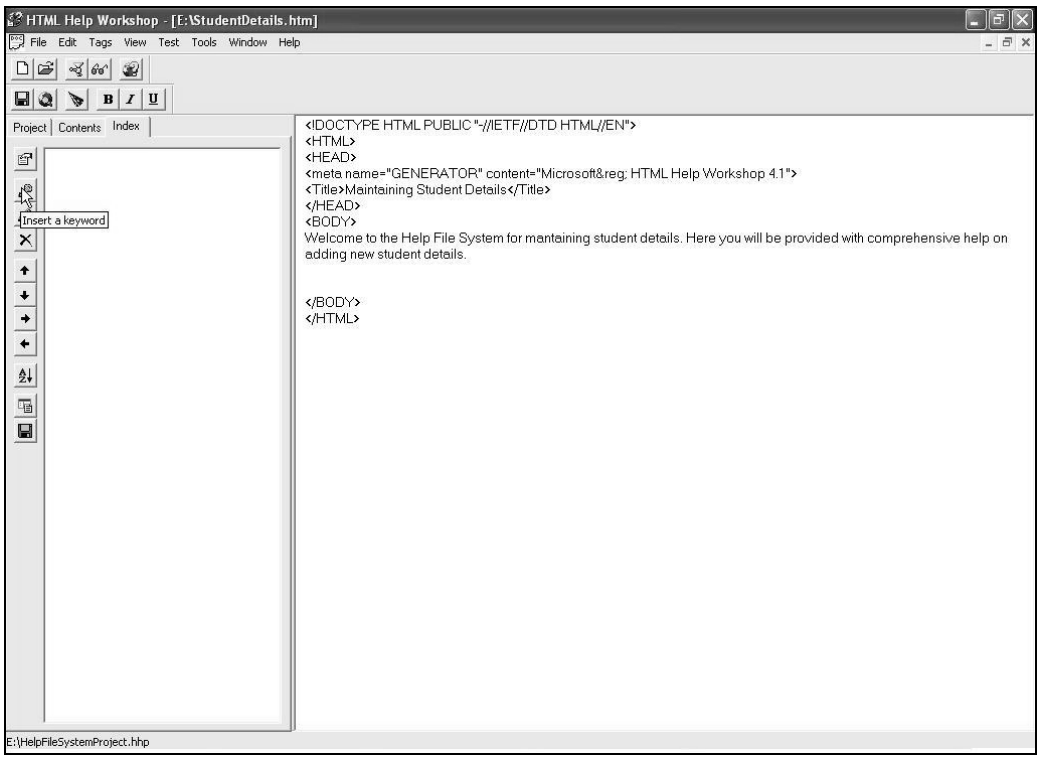
2. Select the **Create a new index file** option and click the **OK** button.

3. Type the name as **Fields** in the **File name** text box, select the **E:** in the **Save in** drop-down in the **Save As** dialog box, and click the **Save** button. The index view is displayed, as shown in the following figure.



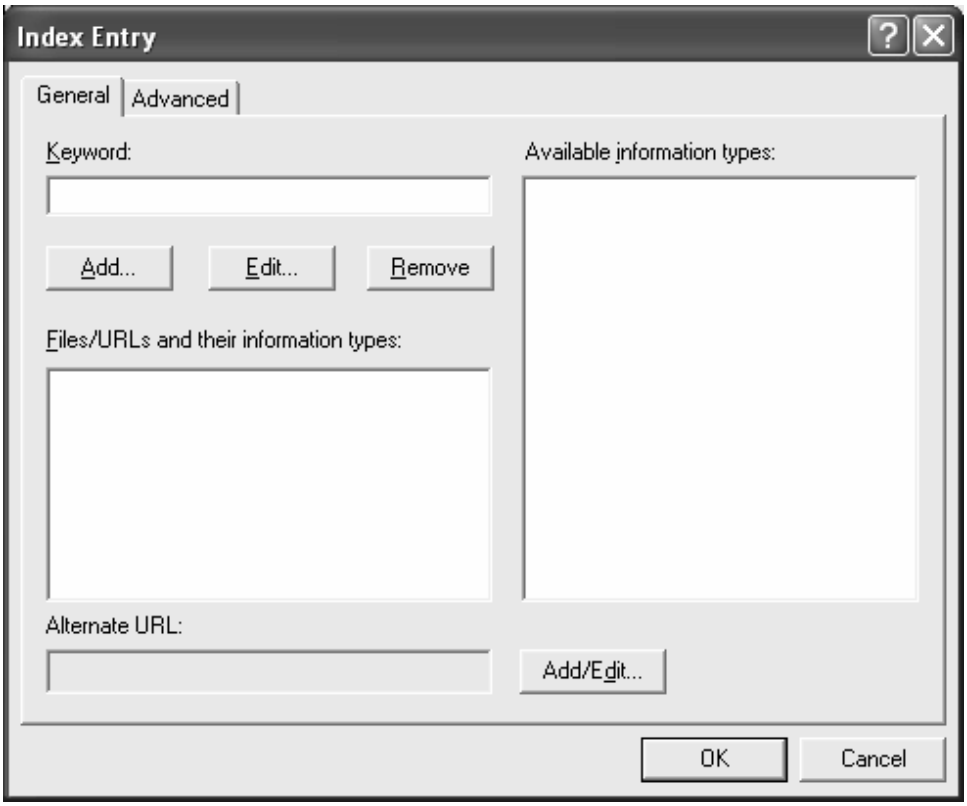
The Index View

4. Click the **Insert a keyword** icon on the toolbar to insert a keyword, as shown in the following figure.



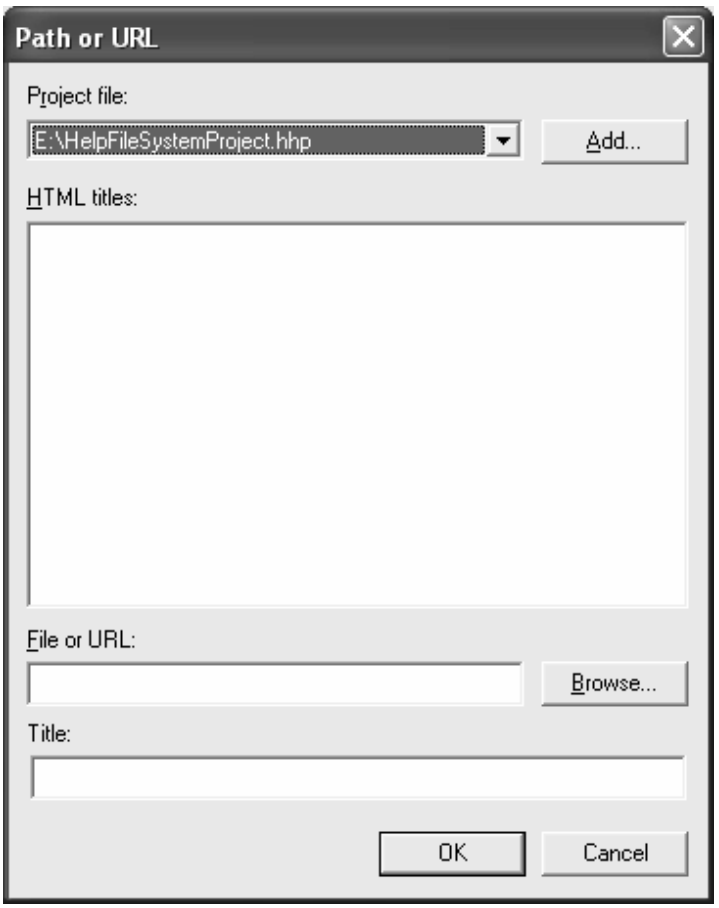
Window Displaying Insert the Keyword Icon

The **Index Entry** dialog box is displayed, as shown in the following figure.



The Index Entry Dialog Box

5. Type **Details** as the search keyword in the **Keyword** text box and click the **Add** button. The **Path or URL** dialog box is displayed, as shown in the following figure.



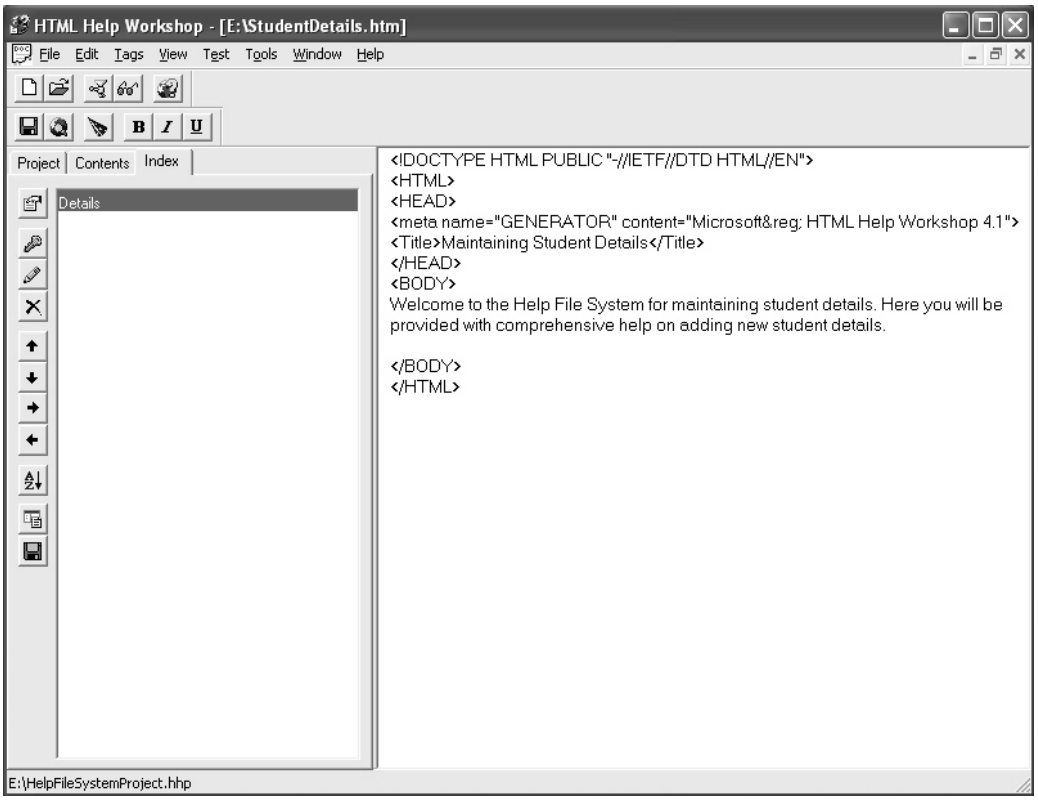
The Path or URL Dialog Box

By default, the currently open Help project is displayed in the Project file drop-down.

6. Type the HTML file name to be associated with the keyword either by entering its name in the **File or URL** text box or by selecting the file by using the **Browse** button to close the **Path or URL** dialog box.
7. Click the **OK** button in the **Index Entry** dialog box.

Repeat the steps 4-7 for assigning keywords to each help topic.

A sample index screen with a keyword is displayed in the following figure.



A Sample Index

Task 4: Creating a Table of Contents

To create a table of contents, you need to perform the following tasks:

1. Create a new contents file.
2. Add a heading to the table of Contents.
3. Insert related topic pages.

Task 4.1: Creating a New Contents File

To create a new contents file, you need to perform the following steps:

1. Click the **Contents** tab. The **Table of Contents Not Specified** dialog box, prompts you to confirm either to create a new contents file or to open an existing one, as shown in the following figure.

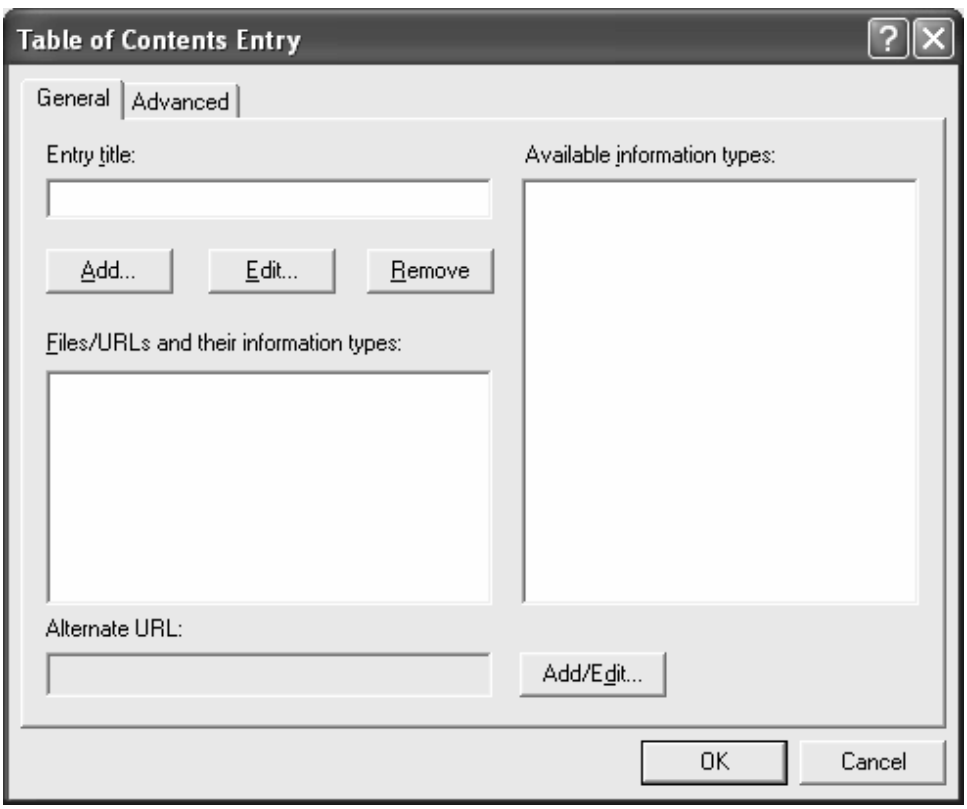
The Table of Contents Not Specified Dialog Box

2. Click the **OK** button. The **Save As** dialog box is displayed.
3. Ensure that the path is same as the path of the project and type the name of the file in the **File name** combo box. By default the name of the file is **Table of Contents**.
4. Click the **Save** button to close the **Save As** dialog box.

Task 4.2: Adding a Heading to the Table of Contents

After creating a new contents file, you need to add a heading to the table of contents. A heading groups various related help topics under it. To create a heading, perform the following steps:

1. Click the **Insert a heading** ( ) icon in the Toolbar of the **Contents** tab. The **Table of Contents Entry** dialog box is displayed, as shown in the following figure.

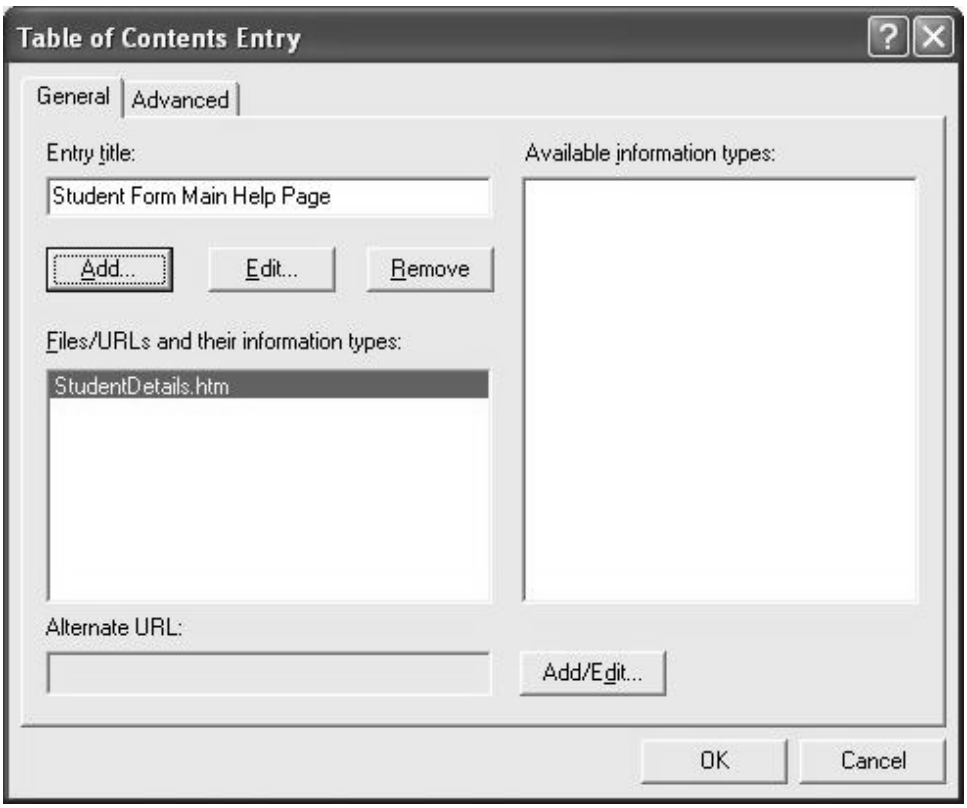


The Table of Contents Entry Dialog Box

2. Type **Student Form Main Help Page** in the **Entry title** text box and click the **Add** button. The **Path or URL** dialog box opens.
3. Type the filename in the **File or URL** text box and click the **OK** button to go back to the **Table of Contents Entry** dialog box.

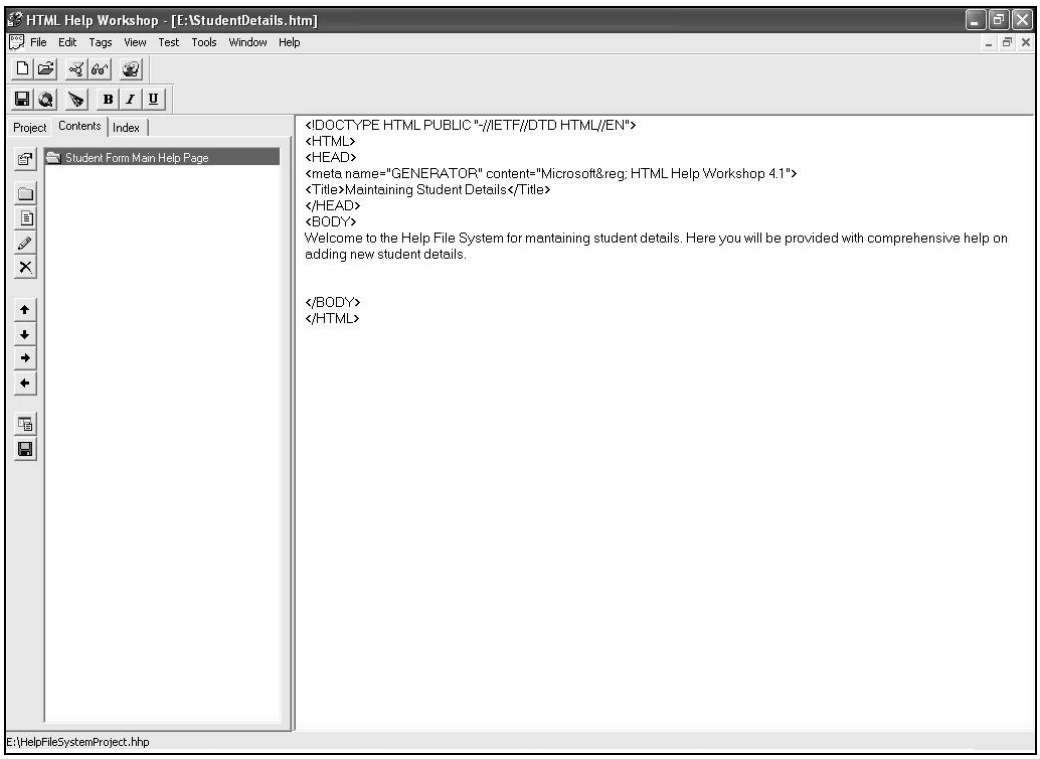
   Repeat steps 1 to 3 to add more headings in the table of contents.

A sample **Table of Contents Entry** dialog box is displayed in the following figure.



The Table of Contents Entry Dialog Box

4. Click the **OK** button. The heading that you have created is displayed preceding the folder icon, as shown in the following figure.

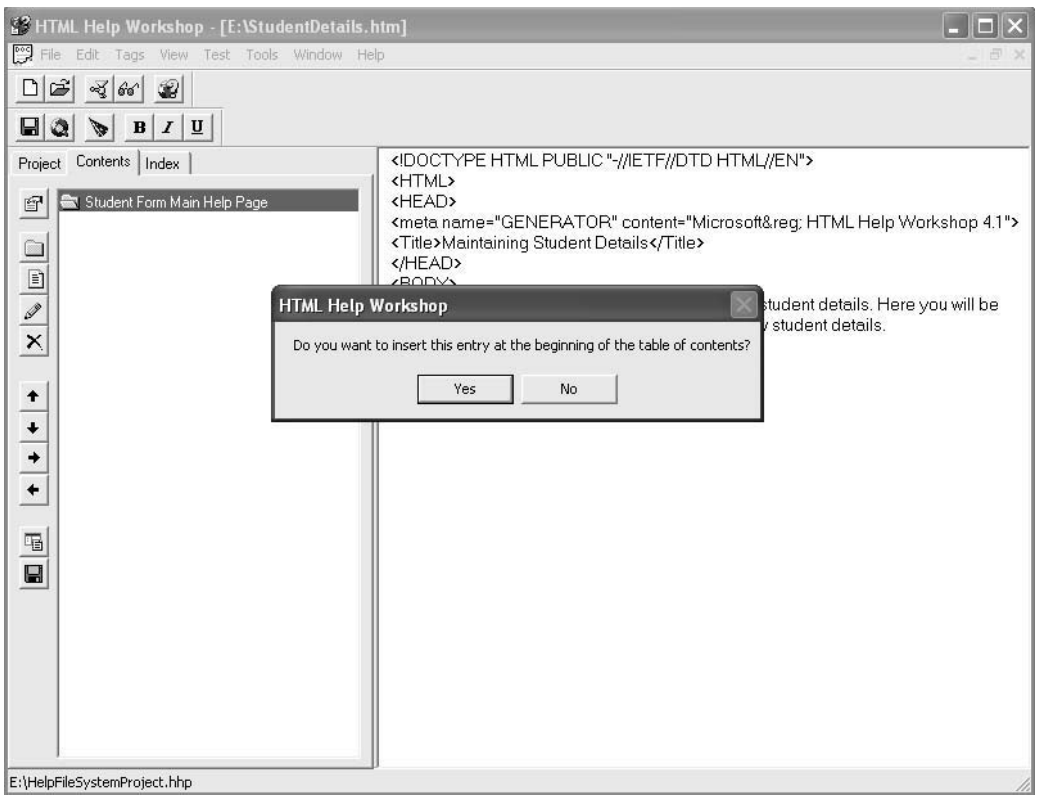


The Sample Heading

Task 4.3: Inserting Related Topic Pages

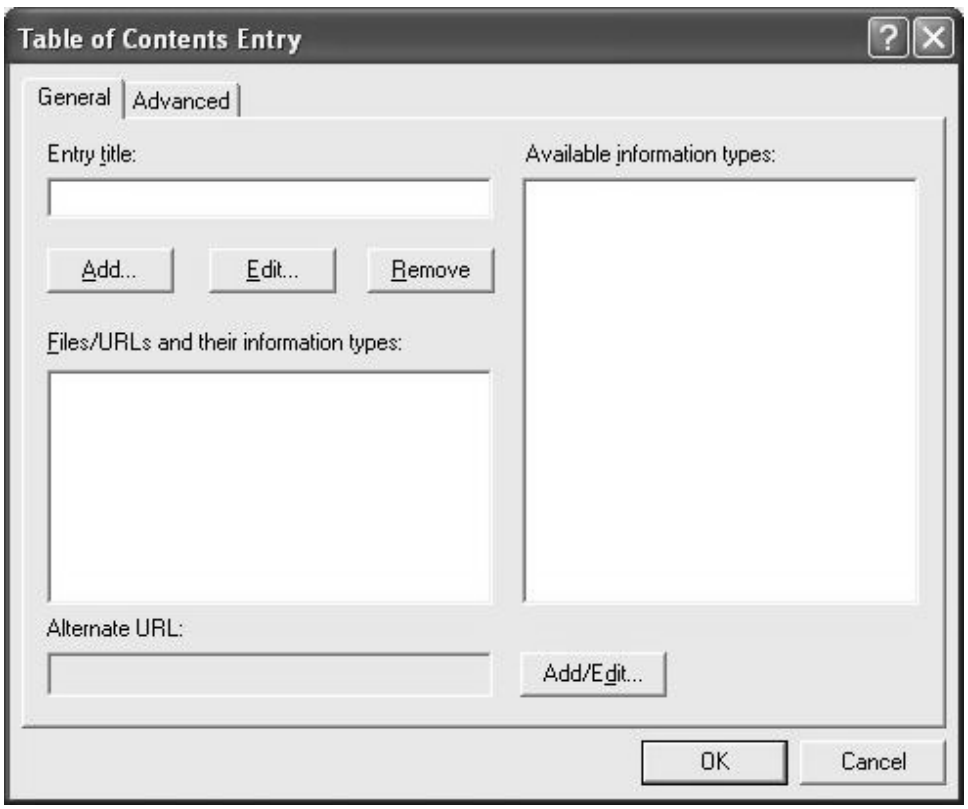
After the heading has been created, the related topic pages need to be inserted. To create a topic page, you need to perform the following steps:

1. Click the **Insert a page** ( ) icon in the toolbar. The **HTML Help Workshop** dialog box is displayed, as shown in the following figure.



The Message Box Confirming the Position of the Related Topic Page

2. Click the **No** button to insert the entry under the **Student Form Main Help Page** heading. The **Table of Contents Entry** dialog box appears, as shown in the following figure.

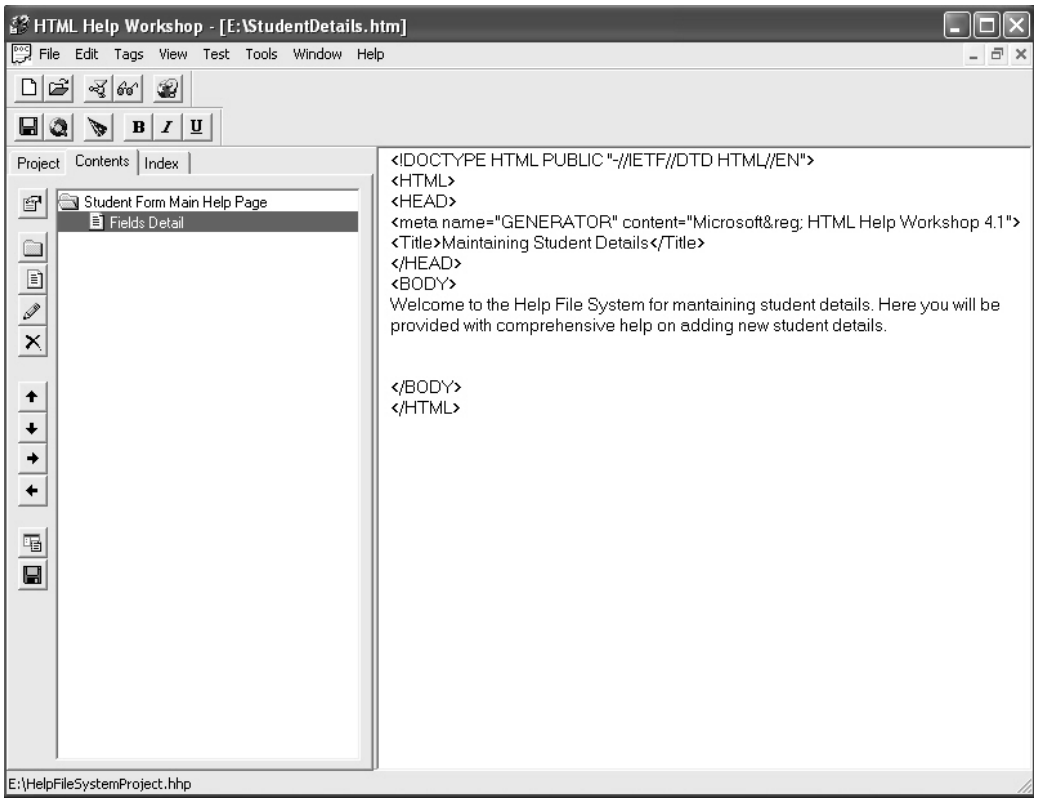


The Table of Contents EntryDialog Box

3. Type **Fields Detail** as the title for the topic page in the Entry title text box and click the **Add** button. The **Path or URL** dialog box opens.
4. Type the file name to be linked to this topic in the **File or URL** text box and click the **OK** button to go back to the **Table of Contents Entry** dialog box.
5. Click the **OK** button to close the dialog box.

Repeat the steps from 1 to 5 for adding all topics related to the heading.

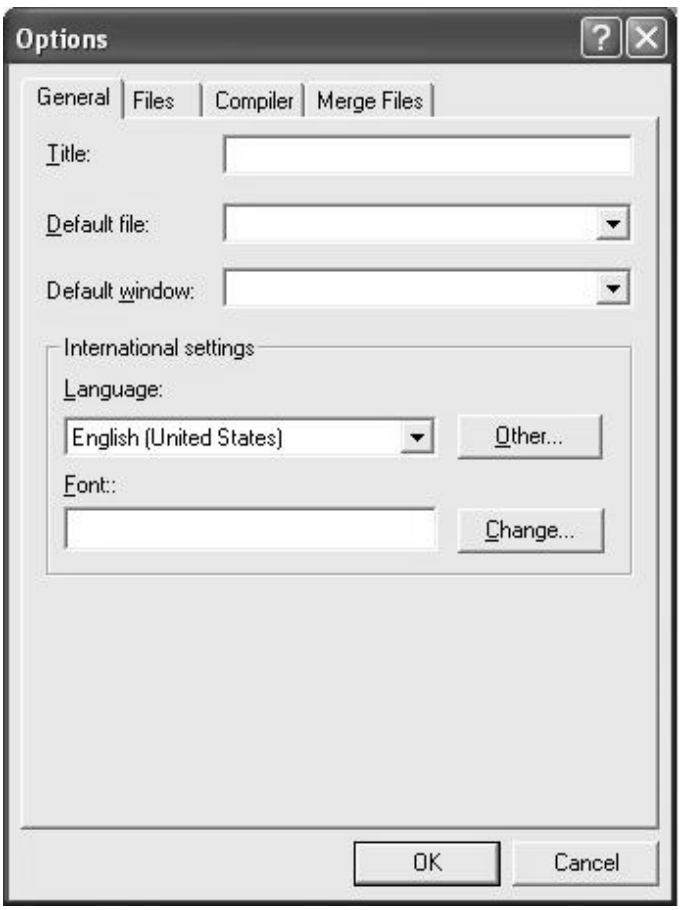
A sample Table of contents with various help topics is shown in the following figure.



The Content View with Content Pages

You can specify a default HTML file for your project. When the Help file is displayed to the user, the HTML file, which is set as the default file for the project, opens automatically. To set a default file for the project, click the Project tab and then, click the Change project options icon in the toolbar.

The Options dialog box is displayed, as shown in the following figure.



The Options Dialog Box

Type the name and path of the file, which you want to set as the default file, in the Default file combo box and click the OK button.

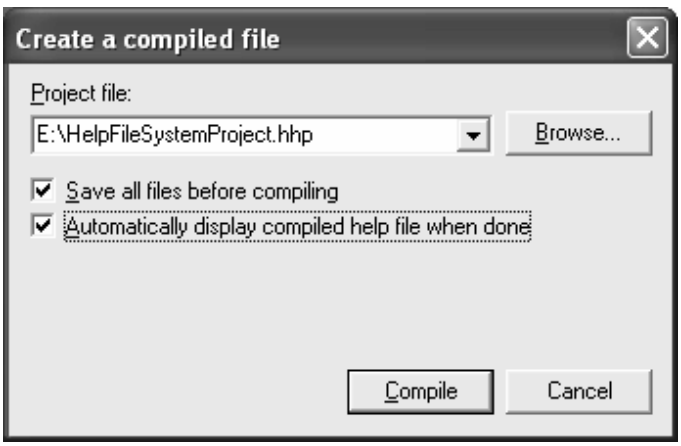
Note

In this case, we are using E:\StudentDetails.htm as the default file.

Task 5: Compiling and Verifying the Project File

To compile a project file, you need to perform the following steps:

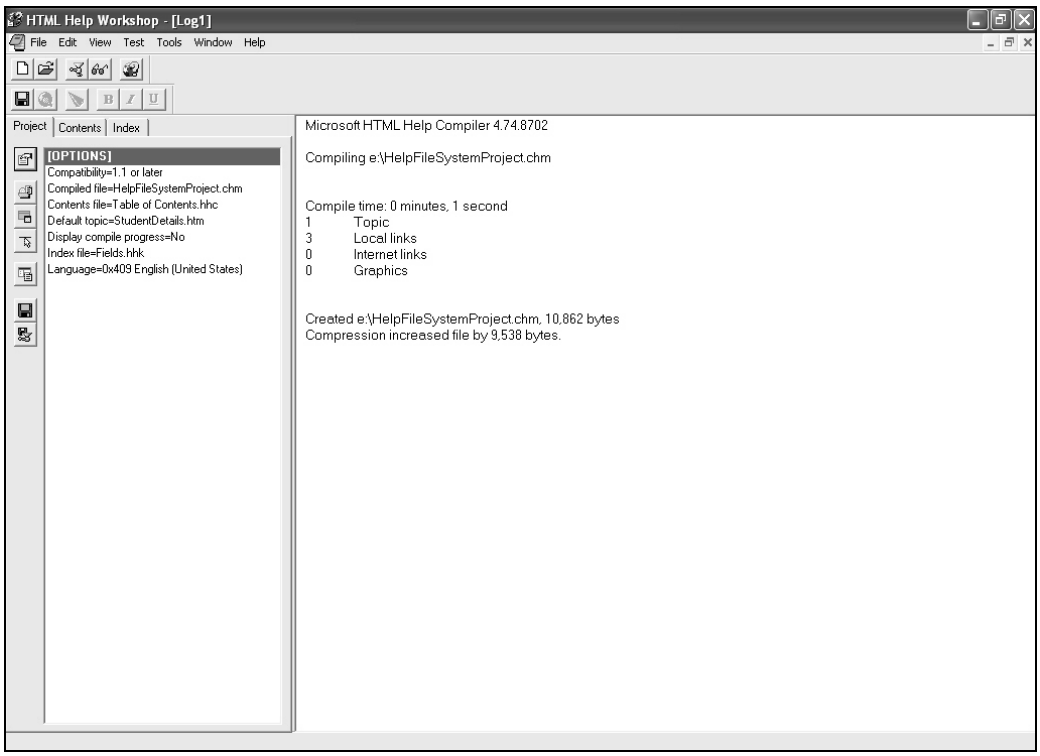
1. Select **File→Compile**. The **Create a compiled file** dialog box is displayed. By default, the path of the current project file is displayed in the **Project file** combo box.
2. Select the **Save all files before compiling** and the **Automatically display compiled help file when done** check boxes, as shown in the following figure.



The Create a compiled file Dialog Box

3. Click the **Compile** button to start compiling the project file. The **Save As** dialog box with the default name **Log1** appears.
4. Click the **Save** button.

The Project file is compiled as .chm file. After compilation is complete, the compiler information is displayed, as shown in the following figure.



The Compiler Information Displayed

After compiling the project, you can verify it by going to the location where you have saved the file and clicking the HelpFileSystemProject.chm file.

5. Close the HTML Help Workshop application.

Implementing Help System in .NET Applications

When running an application, a user might need help on the function of a user interface. The user should be able to access the help system by pressing the F1 key.

To provide help for an application, you need to link it with a help system. The help system can be associated with your Windows application with the help of the HelpProvider control. To do so add a HelpProvider control to the form and then link the control with a help file.

This can be done by setting the HelpNamespace property of the HelpProvider control to the help file with which the form is to be linked. The HelpProvider control can be used to provide context-sensitive help and pop-up help in an application.

Providing Context-Sensitive Help

Context-sensitive help assists users by providing help based on a specific dialog box or a control in a program. This enables users to get specific information about whatever part of the program they are using at any given moment.

To add context-sensitivity to the application, the various controls in the form need to be linked with the Help file.

To add context-sensitive help to a control, you need to set the following properties for controls:

- ShowHelp on <HelpProvider control> to True.
- HelpKeyword on <HelpProvider control> to an appropriate keyword for retrieving help.
- HelpNavigator property to any value, as listed in the following table.

Value	***Description***
Find	*Displays the search page*
Index	*Displays the index*
AssociateIndex	*Displays the index for a specified topic*
Topic	*Displays a help topic*
KeywordIndex	*Displays a keyword search result*
TableOfContents	*Displays the table of contents*

The HelpNavigator Property Values

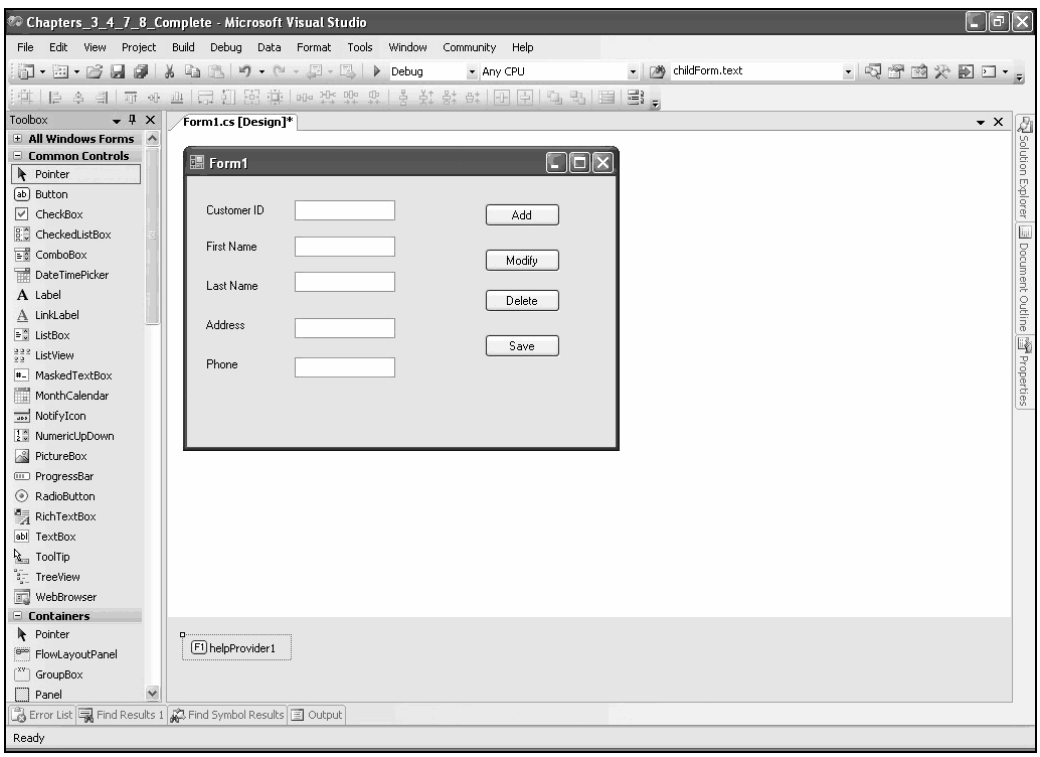
To implement context-sensitive help, you need to perform the following tasks:

1. Link your application with a help file.
2. Add context sensitive help to various form elements.

Link the Help File with the Application

To link the help file with the application, you need to perform the following steps:

1. Open your form in the Design mode and drag a **HelpProvider** control from the **Windows Forms** tab of the **Toolbox** into the form. The **HelpProvider** control is added to the component tray, as shown in the following figure.



The HelpProvider Control in the Component Tray

2. Right-click the **helpProvider1** control from the component tray and select **Properties** from the shortcut menu.

3. The **Properties** window is displayed, as shown in the following figure.

The Properties Window for the HelpProvider Control

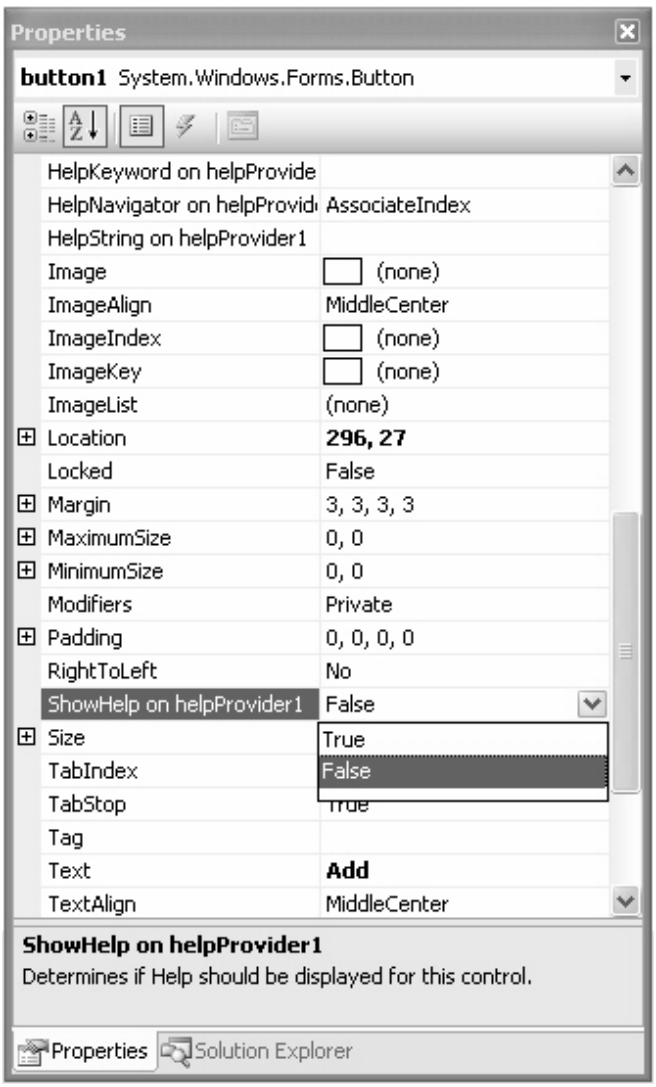
4. Click the **HelpNamespace** property and then click the ellipse button to open the **Open Help File** dialog box.
5. Select a compiled help file in the **Open Help File** dialog box and click the **Open** button.

Add Context-Sensitive Help to Various Form Elements

To Add Context-Sensitive Help to Various Form Elements, you need to perform the following steps:

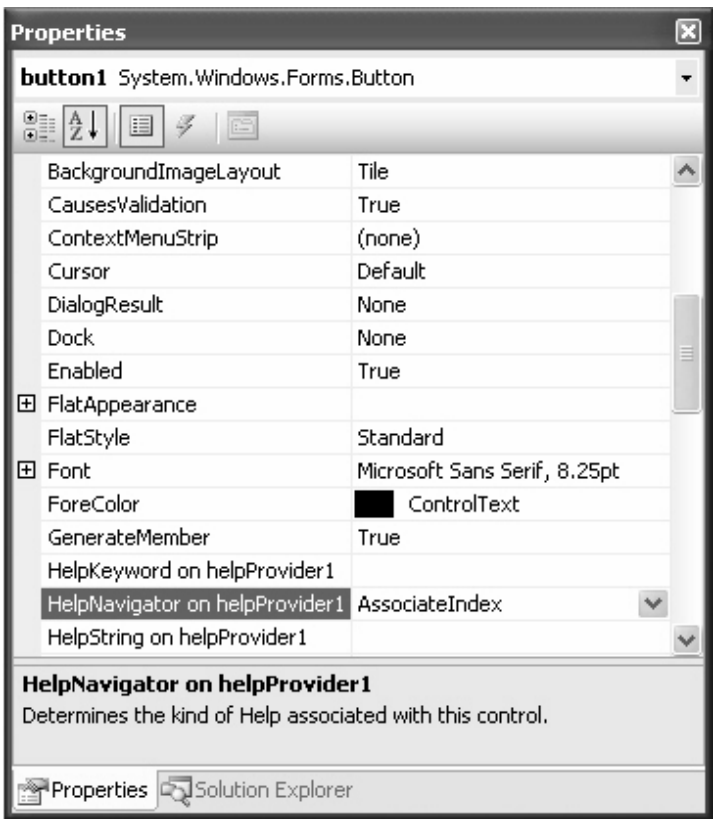
1. Open the **Properties** window for any control present on the form.

2. Set the **ShowHelp on helpProvider1** property to **True**, to display help for the control at run time, as shown in the following figure.

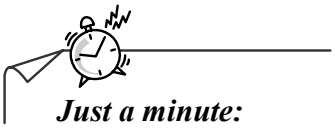


The Properties Windows Displaying the ShowHelp on helpProvider1 Property

3. Type a search keyword for the **HelpKeyword on helpProvider1** property.
4. Select an appropriate value for the **HelpNavigator on helpProvider1** property, as shown in the following figure.



The Property Windows for the Add Button



Just a minute:

Which property value of the HelpNavigator property will display the index for a specified topic?

1. *Index*
2. *AssociateIndex*
3. *KeywordIndex*
4. *Topic*

Answer:

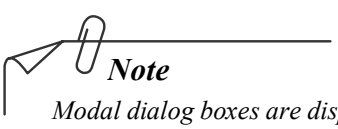
2. *AssociateIndex*

Working with Pop-up Help

Pop-up help is a common method for displaying tips for a control in an application. Pop-up help can be provided in an application through the Help button, also called the What's This button. This button is present in the application on the right of the title bar. In a VC# application, the pop-up help can be displayed by setting a few properties of the form and its controls.

Displaying Pop-up Help

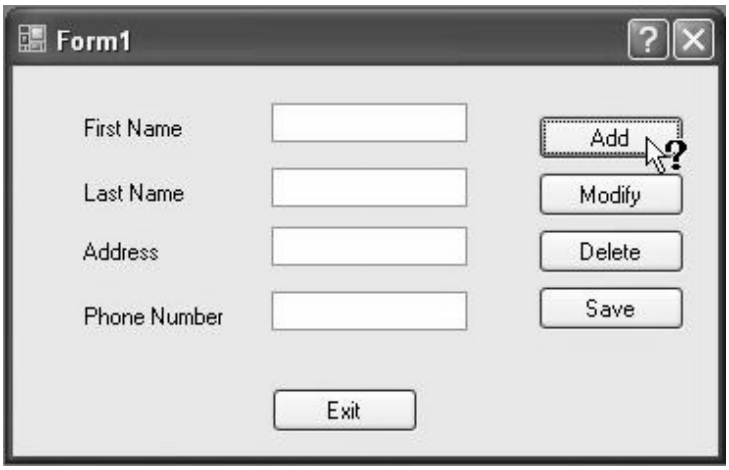
Pop-up help is most effective in modal dialog boxes. The Help button is displayed when the Maximize and Minimize buttons are not there on the title bar. Modal dialog boxes do not have the Maximize and Minimize buttons unlike forms that have these buttons by default on the title bar, therefore using pop-up help is most suitable for modal dialog boxes.



Note

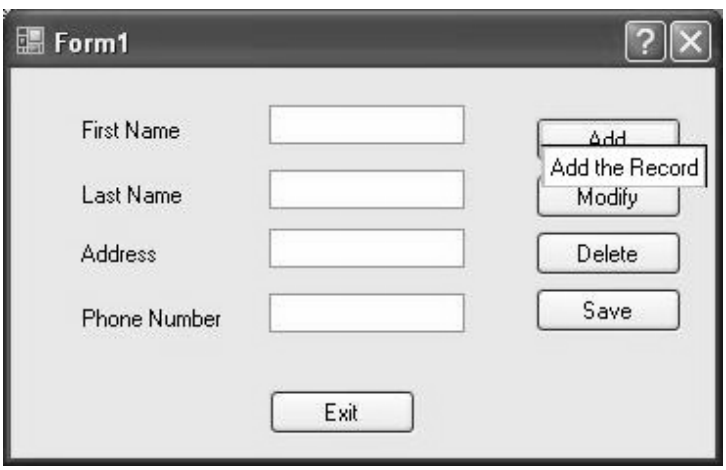
Modal dialog boxes are displayed using the `ShowDialog()` *method.*

A sample application displaying popup help is shown in the following figure.



The Question Mark Icon Attached to the Cursor

When the What's This button on the title bar is clicked, a question mark gets attached with the cursor. If the user clicks a control on the form, pop-up help for that control is displayed. The sample pop-up help is displayed in the following figure.



The Sample Pop-up Help

To display Help button on the title bar of a form, you need to perform the following steps:

1. Set the **MinimizeBox** and **MaximizeBox** properties of the form to **False**.
2. Set the **HelpButton** property of the form to **True**.
3. Add a **HelpProvider** control to the form.

To associate pop-up help with various controls on the form, set the HelpString on <HelpProvider Control> property of the controls to a string that specifies the help text for the control.

Working with ToolTip Control

A tool tip is a brief message for individual controls on a form. Tool tips are generally used for displaying help for toolbar icons. They can be displayed by using the ToolTip control present in the Toolbox. You can access a tool tip by pointing to the control for which the tool tip has been created.

A sample application displaying tool tip for a control is displayed as shown in the following figure.

The Sample Tooltip Help

Displaying Tool Tips

To display a tool tip, you need to add a ToolTip control to the form and enter the text in the ToolTip on <Tooltip Control Name> property of the control for which the tool tip is to be displayed.

This property can be used to set the tool tip for a StatusStrip control, a ToolStrip control, and a selected tab of TabControl. You do not need to insert a ToolTip control in the form to view tool tips for these controls.

Consider the following code to set a tool tip text for a ToolStripButton control programmatically:

```
toolStripButton1.ToolTipText = "Add a record";
```

Controls such as the Button control and the TextBox control require the ToolTip control to display tool tips. The following code sets a tool tip text for the Button control programmatically:

```
toolTip1.SetToolTip(button1, "Add a record");
```

The `SetToolTip()` method of the ToolTip control is used to set the tool tip text for a control. The `SetToolTip()` method takes two parameters, the control for which the tool tip text is to be set and the tool tip text to be displayed for that control.

Some of the important properties of the ToolTip control are:

- **Active**: Is used to set the current status of the ToolTip control. The property can be set to either True or False. By default, the Active property is set to True.
- **AutomaticDelay**: Is used to set time in milliseconds after which the tool tip would appear. By default, the AutomaticDelay property is set to 500.

There are three properties of the ToolTip control that have values based on the value of the AutomaticDelay property. The following table describes the properties of the ToolTip control.

Property	***Description***
AutoPopDelay	*Is used to set the time period for which the tool tip is displayed on the screen when the mouse pointer is stationary in a tool tip region. When the AutomaticDelay property is set for a ToolTip control, the AutoPopDelay property is automatically set to 10 times the value of the AutomaticDelay property. You can also set the value for the AutoPopDelay property independently and override the default value.*
InitialDelay	*Is used to set the time period for which the mouse pointer should be stationary in the tool tip region before the tool tip appears. When the AutomaticDelay property is set for a ToolTip control, the InitialDelay property is automatically set to the value of the AutomaticDelay property. You can also set the value for the InitialDelay property independently, thus overriding the default value.*

Property	*Description*
ReshowDelay	*Is used to set the time taken by the subsequent instances of the tool tip to appear when the mouse pointer is moved from one tool tip region to another. When the AutomaticDelay property is set for a ToolTip control, the ReshowDelay property is automatically set to one fifth of the value of the AutomaticDelay property. You can also set the value for the ReshowDelay property independently, thus overriding the default value.*

The ToolTip Control Properties

Practice Questions

1. The _____ property of a control specifies a name for the control by which the accessibility aids identify the control.
   a. Name
   b. AccessibleName
   c. AccessibleAidName
   d. AccessibleIdentity

2. Which of the following option lists the culture code for Chinese language, China region?
   a. en-CA
   b. fr-FR
   c. zh-CN
   d. de-DE

3. Which of the following involves customizing an application for specific cultures?
   a. Globalization
   b. Localization
   c. Culture-specific formatting
   d. Deployment

4. Which property of a form needs to be set to implement right-to-left display?
   a. RightToLeft
   b. Direction
   c. Display
   d. TextAlign

5. Which method is used to convert an existing encoding to Unicode?
   a. `Encoding.GetEncoding`
   b. `Encoding.ToUnicode`
   c. `Encoding.Convert`
   d. `Encoding.Translate`

Summary

In this chapter, you learned that:

- Applications need to be made accessible so that they can be used by a wide range of audience, including those with special needs.
- You need to follow a set of guidelines for developing accessible applications.
- The .NET Framework provides support for the development of world-ready applications. It helps you create applications that adapt to various languages, currency formats, date/time formats, and other culture-specific information.
- When developing a world-ready application, the process should be divided into three phases: globalization, localizability, and localization.
- While globalizing an application, you should strive to code the application in such a way that allows it to work equally well for all cultures your application supports.
- During the localizability phase, you test whether the application's executable code has been properly separated from its resources.
- During localization an application is customized for specific cultures or regions.
- Unicode provides a standard encoding scheme for characters of various languages across the world. Therefore, it and hence helps in creating application for multiple cultures and having a multilingual user interface.
- A resource is any non-executable data that is logically deployed with an application.
- Visual Studio.NET allows the creation of three types of help systems:
  - HTML Help
  - Pop-up Help
  - Tool Tips
- The HelpProvider control is used to associate a Help file with an application.
- The Help file to be associated with the HelpProvider control is specified using the HelpNamespace property of the HelpProvider control.
- The HelpKeyword property of a control is used to set the keyword for retrieving help from the Help file specified in the HelpNamespace property.

Exercises

Exercise 1

Create an application to calculate the amount of tax that an individual has to pay. The tax is calculated as 20% of the individual's salary. A form has already been created for accepting the name and salary of a user. The form also contains a button labeled Calculate which when clicked displays the amount of tax to be paid by the user in a label.

The following code is added to the click event of the Calculate button:

```
private void CalcButton_Click(object sender, System.EventArgs e)
{
  float tax;
  //Calculate tax
  tax = (float)0.2 * Convert.ToSingle(textBox2.Text);
  //Display the tax amount
  TaxValueLabel.Text = "$" + tax.ToString();
}
```

The application is provided to you as a data file. You need to localize this application for the French culture. The following table lists the French translations for various strings used on the form.

English String	***French Translation***
Name	*Nom*
Salary	*Salaire*
Tax	*Impôts*
Calculate Tax	*Calculez L'Impôt*

Ensure that the culture-specific currency symbol is also displayed in the localized form.