

# **10Tec hTooltip version 1.1 Manual**

## Contents

<b>Contents</b> .....	<b>1</b>
<b>Introduction</b> .....	<b>2</b>
<b>Adding hTooltip to Your Project</b> .....	<b>3</b>
Visual Basic and VBA.....	3
Other development environments .....	3
<b>The Basics</b> .....	<b>5</b>
Some definitions.....	5
Creating and displaying tooltips.....	5
<i>Sample #1: Creating a pop-up balloon tooltip for a command button</i> .....	5
<i>Sample #2: Using a display-on-demand tooltip for a text box</i> .....	6
<b>CTooltip's Members</b> .....	<b>8</b>
Tooltip properties .....	8
<i>Appearance properties</i> .....	8
<i>Behavior properties</i> .....	9
<i>Information properties</i> .....	10
Tooltip methods.....	10
<i>The Create* methods</i> .....	10
<i>The Destroy method</i> .....	11
<i>The Show method</i> .....	11
<i>The Hide method</i> .....	12
<i>The SetAllMargins method</i> .....	12
<i>The Supports function</i> .....	12
<i>The InitWithPattern method</i> .....	12
<i>The ShowAboutBox method</i> .....	13
Tooltip events.....	13
<i>The MouseEnter and MouseLeave events</i> .....	13
<b>Advanced Topics</b> .....	<b>15</b>
Multiline tooltips.....	15
Using tab stops .....	15
Setting tooltip font .....	15
Using the MouseEnter/MouseLeave events .....	16
Creating and displaying tooltips dynamically .....	17
<i>Scenario #1: A huge number of objects</i> .....	17
<i>Scenario #2: Tracking tooltips</i> .....	18
The ScaleMode property .....	20
Setting DelayTime and VisibleTime to system defaults .....	20
Tooltip patterns and the DefaultTooltip global object .....	20
The hTooltip ProgID and late binding .....	22
<b>Compatibility Info</b> .....	<b>23</b>
Windows visual styles support.....	23
hTooltip features and the Windows Common Controls library .....	23
Balloon tips availability .....	24
<b>Known Problems and Limitations</b> .....	<b>25</b>

## Introduction

The hTooltip component is an ActiveX DLL that provides you with a set of the objects you can use to create native Windows tooltips in your Win32 applications.

The name 'hTooltip' itself means '**h**andy **tool**tip', and the main goal of this component is to give you a handy tool you can use to create system-like tooltips with different options and features in your apps.

With hTooltip you can use the most important features of native Windows tooltips, such as:

- The standard rectangular and balloon styles.
- Tooltips can have a title and one of the predefined system icons.
- Displaying tooltips on demand at a specified point on the screen.
- The ability to adjust the colors of the tooltip window and its text.
- You can set the delay time and visible time of your tooltips.

In addition to these standard system features, hTooltip provides you with the following features which can lighten your work:

- The tooltip object raises the `MouseEnter` and `MouseLeave` events for the area it is attached to.
- hTooltip can be used to create tooltips for controls that do not have their own window handle (also known as 'windowless controls'). These are the Label, Shape and Image controls in Visual Basic 6.
- You can specify a rectangular area inside a control for which the tooltip should be displayed.
- hTooltip implements the `CreateForVBCtrl` method you can use to attach a tooltip to any Visual Basic 6 intrinsic control with one statement.

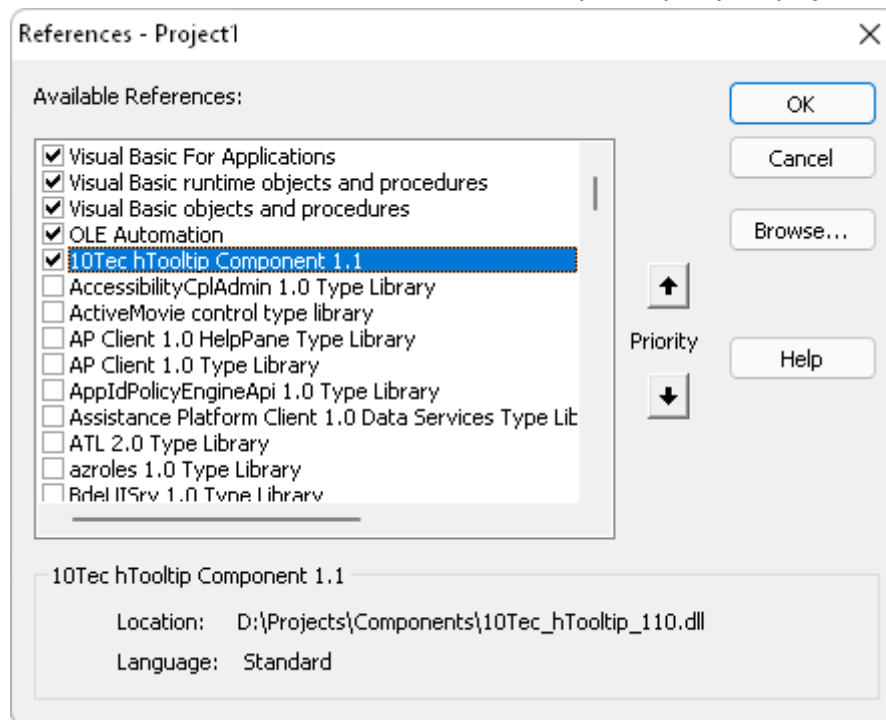
hTooltip can be used in any version of Windows, both client and server editions are supported. hTooltip automatically uses the visual effects available in latest versions of the OS.

The component was designed originally for Visual Basic 6, but it can be used in development environments that can work with COM objects, including Visual Basic for Applications in Microsoft Office, Microsoft Visual FoxPro, Borland Delphi, and the .NET Framework.

## Adding hTooltip to Your Project

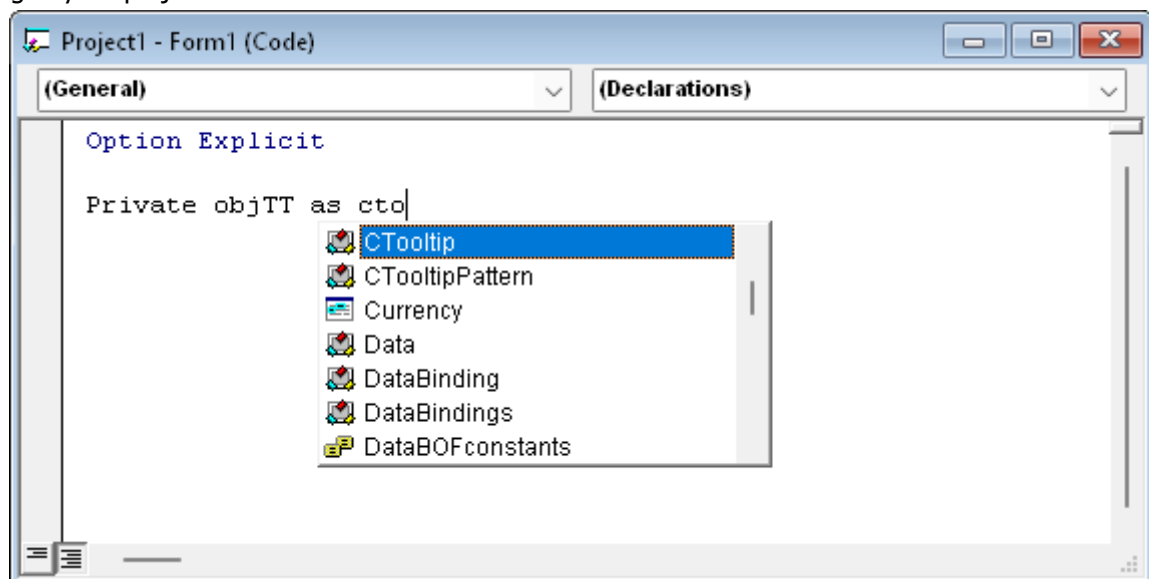
### Visual Basic and VBA

hTooltip is an ActiveX DLL and can be added to your project like any other ActiveX DLL in the Project References dialog in VB6 IDE or VBA IDE in Microsoft Office. To open this dialog, select the References... item in the Project menu in VB6 or in the Tools menu in VBA. In the Project References dialog, check the '10Tec hTooltip Component 1.1' item to add the reference to the hTooltip library to your project:



If the hTooltip library is not listed, click the Browse... button and select the hTooltip library DLL (10Tec\_hTooltip\_110.dll) in the Add Reference dialog.

Now you can use hTooltip's objects in your code. The IntelliSense feature proves that you are using early binding in your project:



The `CTooltip` and `CTooltipPattern` items are classes from the hTooltip library.

### Other development environments

Some development environments let you use COM objects but do not let you add references to ActiveX DLLs. In this case you can use hTooltip's objects via late binding. Each COM object has its own program identifier or

ProgID, and hTooltip's one is 'hTT110\_10Tec'. This lets you create hTooltip's objects by their ProgIDs. For instance, the core class in the hTooltip library has the name 'CTooltip', and in Visual FoxPro you can create an instance of this object with the following statement:

```
objHTT = CREATEOBJECT("hTT110_10Tec.CTooltip")
```

Note that `CreateObject` is also an intrinsic function in VB6 and VBA, and you can use it to create the reference to the instance of the `CTooltip` class using late binding. For more details, see the topic [The hTooltip ProgID and late binding](#) in this document.

## The Basics

### Some definitions

**The tooltip control** (or simply **'tooltip'** in the text below) is a pop-up window that displays text and optionally an icon. The text usually describes a **tool**, which is either a window, such as a child window or control, or an application-defined rectangular area within a window's client area. The pop-up window with tooltip text is called **the tooltip window**.

In this document we often use the term **tooltip object** meaning the object in your code the tooltip functionality is provided with.

### Creating and displaying tooltips

The `CTooltip` class is the core class of the hTooltip component. You deal with it when you create tooltips in your apps.

To create a tooltip in your app:

- 1) Create the instance of the `CTooltip` class.
- 2) Set the required properties of your tooltip.
- 3) Create and attach the tooltip window to a control or a rectangular area with one of the `Create*` methods.

The tooltip can then be displayed on the screen using one of the following 2 modes:

#### 1. The pop-up mode.

This is the default mode. The tooltip appears automatically, or pops up, when the mouse pointer hovers over the tool. The tooltip appears near the pointer and disappears when the user clicks a mouse button or moves the pointer away from the tool.

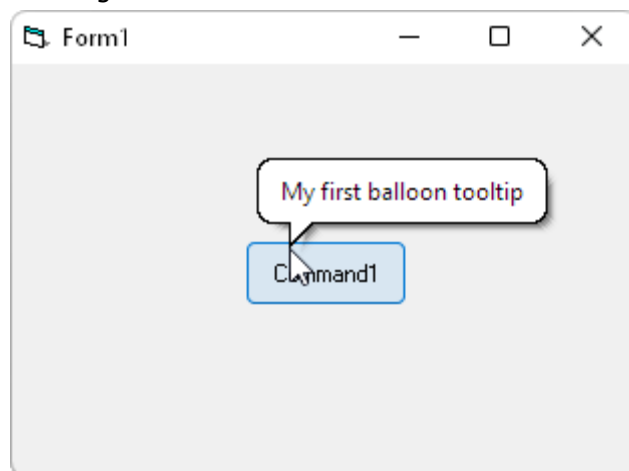
#### 2. The display-on-demand mode.

The tooltip is displayed on the screen when you invoke the `Show` method in code. This mode must be turned on with the `DisplayOnDemand` property; in this case the tooltip does not pop up automatically when the mouse pointer hovers over the tool. The coordinates the tooltip is displayed at are also specified with the `Show` method.

### Sample #1: Creating a pop-up balloon tooltip for a command button

[The source code for this section is stored in the Tutorial 1 - 1st balloon tooltip](#)

This sample demonstrates how to create a balloon tooltip and attach it to a command button in a VB6 form. The result will look like the following:



Create a new exe-application project with one form in VB6 and place the `CommandButton` control on the form. Name the button `Command1`, then declare the variable that will store the reference to the tooltip object in the form's module:

```
Private objTT As CTooltip
```

In the `Form_Load` event sub create the instance of the `CTooltip` class:

```
Set objTT = New CTooltip
```

Then initialize its properties. We will create a balloon tooltip with the text 'My first balloon tooltip':

```
objTT.Text = "My first balloon tooltip"  
objTT.Style = httStyleBalloon
```

And finally we need to create the real tooltip window for our button `Command1`:

```
objTT.CreateForVBCtrl Command1
```

That's all! When you launch a form and place the mouse pointer over the command button, the tooltip appears. The full source code looks like the following:

```
Private objTT As CTooltip  
  
Private Sub Form_Load()  
    Set objTT = New CTooltip  
  
    objTT.Text = "My first balloon tooltip"  
    objTT.Style = httStyleBalloon  
  
    objTT.CreateForVBCtrl Command1  
End Sub
```

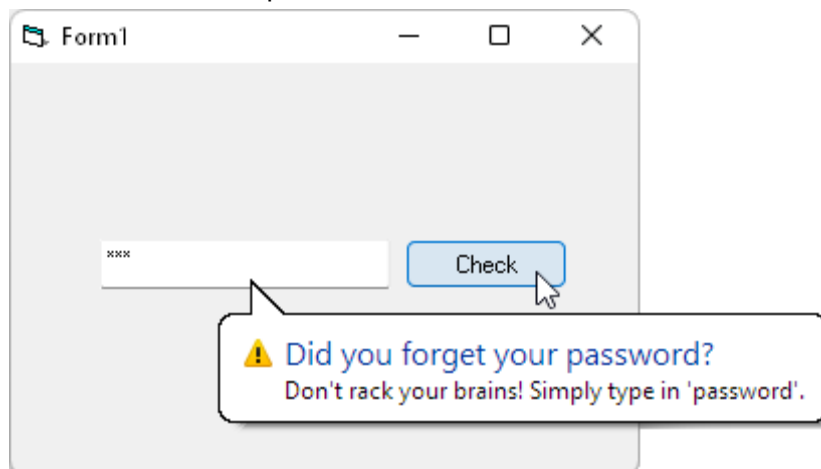
**Important!** You must declare an instance of the `CTooltip` class at module level. If you do it inside a procedure, the tooltip object is released automatically when the procedure finishes and therefore the tooltip will never appear.

## Sample #2: Using a display-on-demand tooltip for a text box

[The source code for this section is stored in the Tutorial 2 - Tooltip on demand](#)

In this sample we demonstrate how to emulate a user message with a display-on-demand tooltip.

We will check the password the user entered into the text box, so place a text box and a command button with their default names `Text1` and `Command1` on a form. Then change the caption of the button to 'Check' and set the `PasswordChar` property of the text box to '\*'. We will compare the entered password with the word 'password' itself when the user presses the Check command button and will display a tooltip-based message if the user entered an incorrect password:



As always, let's declare the tooltip object at the form's module level:

```
Private objTT As CTooltip
```

Then create the tooltip with the message text and icon and attach it to the command button. We do it in the [Load](#) event of the form:

```
Private Sub Form_Load()  
    Set objTT = New CTooltip  
  
    With objTT  
        .Title = "Did you forget your password?"  
        .Text = "Don't rack your brains! Simply type in 'password'."  
        .Icon = httIconWarning  
        .Style = httStyleBalloon  
        .DisplayOnDemand = True  
    End With  
  
    objTT.CreateForVBCtrl Text1  
End Sub
```

Finally, add the required validation code to the [Click](#) event of the command button:

```
Private Sub Command1_Click()  
    If Text1.Text <> "password" Then  
        objTT.Show Text1.Width / 2, Text1.Height * 0.7  
    Else  
        MsgBox "The password is correct"  
    End If  
End Sub
```

When you display a tooltip on demand, you specify the coordinates in the [Show](#) method in the Left and Top properties of the control the tooltip is attached to. The stem of our balloon tooltip starts at that point.



## CTooltip's Members




### Tooltip properties

The `CTooltip` class provides you with a set of properties you can use to control your tooltips or get info about them. These properties can be organized into the following 3 groups:

- 1) Properties that define the appearance of a tooltip.
- 2) Properties that control the behavior of the tooltip.
- 3) Information properties to retrieve some info about a tooltip.

The tables below briefly describe properties in each group.

### Appearance properties

Property	Type	Description	Default value
<code>BackColor</code>	<code>OLE_COLOR</code>	Gets/sets the background color of the tooltip.	<code>vbInfoBackground (&amp;H80000018)</code>
<code>Font</code>	<code>StdFont</code>	Contains a font object used to display the tooltip text.	<code>Nothing</code> (the default system font is used)
<code>ForeColor</code>	<code>OLE_COLOR</code>	Gets/sets the color of the text in the tooltip window.	<code>vbInfoText (&amp;H80000017)</code>
<code>Icon</code>	<code>EIconType</code>	Gets or sets one of the predefined system icons displayed in the tooltip's title:  - <code>httIconInfo</code> (1)  - <code>httIconWarning</code> (2)  - <code>httIconError</code> (3) no icon - <code>httNoIcon</code> (0) Note that the tooltip should have a title in order to display one of these icons.	<code>httNoIcon</code> (0)
<code>MarginBottom</code>	<code>Long</code>	Specifies the bottom margin, in pixels, between the tooltip window border and the text contained within the tooltip window.	0
<code>MarginLeft</code>	<code>Long</code>	Specifies the left margin, in pixels, between the tooltip window border and the text contained within the tooltip window.	0
<code>MarginRight</code>	<code>Long</code>	Specifies the right margin, in pixels, between the tooltip window border and the text contained within the tooltip window.	0
<code>MarginTop</code>	<code>Long</code>	Specifies the top margin, in pixels, between the tooltip window border and the text contained within the tooltip window.	0

<code>MaxWidth</code>	Long	Specifies the maximum width, in pixels, of the tooltip window. If the tooltip string exceeds the maximum width, the control breaks the text into multiple lines, using spaces to determine line breaks.	2,147,483,647
<code>Style</code>	<code>ETooltipStyle</code>	Gets or sets the style of the tooltip. Accepts the following values from the <code>ETooltipStyle</code> enumeration: <code>httStyleStandard</code> (0, normal rectangular tooltip) or <code>httStyleBalloon</code> (1, new modern balloon style)	<code>httStyleStandard(0)</code>
<code>Text</code>	String	Gets/sets the text of the tooltip. The tooltip isn't displayed if this property is empty.	an empty string
<code>Title</code>	String	Gets/sets the title of the tooltip.	an empty string

### Behavior properties

Property	Type	Description	Default value
<code>AppearInInactiveForm</code>	Boolean	Specifies whether the tooltip is displayed when the form is inactive.	False
<code>Centered</code>	Boolean	Specifies whether the tooltip window is centered below the tool it is attached to.	False
<code>DelayTime</code>	Integer	Gets/sets the length of time, in milliseconds, the mouse pointer must remain stationary within the tool's bounding rectangle before the tooltip window appears. Assign -1 to set to the system default value.	the double-click time in the OS (by default 500 ms)
<code>DisplayOnDemand</code>	Boolean	Specifies whether the tooltip will work in the standard pop-up mode (False) or will be forcibly displayed with the Show method (True).	False
<code>MouseEnterOnCreation</code>	Boolean	Specifies whether the <code>MouseEnter</code> event will be raised automatically when the tooltip window is created. Set this property to True if the tooltip is created when the mouse pointer is inside the destination area.	False

<code>ScaleMode</code>	<code>EScaleMode</code>	Specifies the measurement unit for coordinate parameters. Accepts one of the following values from the <code>EScaleMode</code> enumeration: <code>httScaleTwips</code> (0) or <code>httScalePixels</code> (1).	<code>httScaleTwips</code> (0)
<code>SmartPositioning</code>	Boolean	Forces the tooltip window to adjust its coordinates automatically when it is displayed to reveal as much of the tool's bounding rectangle as possible.	True
<code>UseFadeEffect</code>	Boolean	Specifies whether the tooltip will use the fade system effect if it is possible in the current OS.	True
<code>UseSlideEffect</code>	Boolean	Specifies whether the tooltip will use the slide system effect if it is possible in the current OS.	True
<code>VisibleTime</code>	Integer	Gets/sets the length of time, in milliseconds, the tooltip window remains visible if the pointer is stationary within the tool's bounding rectangle. Assign -1 to set to the system default value.	the double-click time in the OS multiplied by 10 (by default 5000 ms)

### Information properties

These properties are read-only.

Property	Type	Description
<code>IsActive</code>	Boolean	Indicates whether the tooltip window is visible
<code>IsCreated</code>	Boolean	Returns a boolean value which indicates whether the WinAPI tooltip window is created.
<code>HwndParent</code>	Long	Returns the WinAPI handle of the window the tooltip is attached to.
<code>HwndTooltip</code>	Long	Returns the WinAPI handle of the tooltip window itself.

### Tooltip methods

#### The Create\* methods

To display a tooltip window, it is not enough to create an instance of the `CTooltip` class and set the required properties. You must create the tooltip window itself and attach it to a particular control or an entire window. One of the `Create*` methods is used for that.

The first method from the `Create*` family you generally use for intrinsic VB6 controls is `CreateForVBCtrl`:

```
Sub CreateForVBCtrl ( _
    ByVal Ctrl As Object _
)
```

The only parameter it accepts is a reference to the target control.

Pay attention to the fact that this method can be used only for Visual Basic 6 controls, but not for UserForm controls in VBA or controls in other development environments.

Another method you can use to create a tooltip window is `CreateForHwnd`:

```
Sub CreateForHwnd( _  
    ByVal Hwnd As Long _  
)
```

It accepts the WinAPI handle of the control or a whole window for which you create the tooltip window. This approach based on `Hwnd` is widely used if you create a tooltip window not from VB6 or VBA. It can be also used in VB6/VBA if you deal with a non-standard control that can return its WinAPI handle.

The last method from this triad is `CreateForRect`. It is helpful if you need to create a tooltip window not for an entire window or a control but only for its rectangular part. In this case you specify the coordinates of that rectangle inside the tool with the specified WinAPI handle:

```
Sub CreateForRect( _  
    ByVal Hwnd As Long, _  
    ByVal Left As Long, _  
    ByVal Top As Long, _  
    ByVal Width As Long, _  
    ByVal Height As Long _  
)
```

This feature can be helpful if you want to display a tooltip only when the mouse pointer hovers over a particular part of the control. For instance, the following statement is used to create a tooltip for a small square area inside the picture box named `Picture1`:

```
objTT.CreateForRect Picture1.hWnd, 0, 0, 90, 90
```

When you invoke the `CreateForVBCtrl` method, internally `hTooltip` invokes the `CreateForHwnd` or `CreateForRect` method. If the specified control has the WinAPI handle (the `Hwnd` property), `CreateForHwnd` is used. Otherwise the method determines the coordinates of the specified control on the form (or another parent control with WinAPI handle) and invokes the `CreateForRect` method with the corresponding parameters.

**Caution:** If you created the tooltip for a control with the `CreateForRect` method and the control can subsequently be resized or moved (for instance, when the form is resized), you need to recreate the tooltip accordingly to the new coordinates whenever the control is moved or resized. The same applies to the `CreateForVBCtrl` method if it was invoked for a windowless control (without the `Hwnd` property – such as the `Label`, `Shape` or `Image` controls in Visual Basic).

If a tooltip window was created with the `Create*` method and is still exists, the `IsCreated` property returns `True`.

## The Destroy method

You destroy the tooltip window created with the `CTooltip` class with this `Destroy` method:

```
Sub Destroy()
```

This method destroys only the tooltip window but not the instance of the `CTooltip` class you invoke it from. You can create and attach the same tooltip to another tool with one of the `Create*` methods again after you've destroyed the tooltip window with `Destroy`.

The `IsCreated` property returns `False` for the tooltip object after you have issued its `Destroy` method.

## The Show method

You can display a display-on-demand tooltip with the `Show` method:

```
Sub Show( _  
    ByVal x As Long, _  
    ByVal y As Long, _  
    Optional ByVal NeverHide As Boolean = False _  
)
```

The tooltip is displayed at the point specified with the `x` and `y` coordinates. The tooltip is automatically hidden after the period of time specified in the `VisibleTime` property unless you set the `NeverHide` optional parameter to True (or simply omit it).

If a tooltip window is already visible when you call the `Show` method, it remains visible and may be repositioned to the new location specified by the `x` and `y` arguments. This feature is often used to move the tooltip window when you implement tracking tooltips that trace the mouse pointer movement.

Note that this method works only for display-on-demand tooltips (the `DisplayOnDemand` property is set to True).

### The Hide method

In contrast to the `Destroy` method, `Hide` is used to hide the tooltip window if it's visible:

```
Sub Hide()
```

The method can be used to hide both a display-on-demand tooltip and a normal pop-up tooltip.

### The SetAllMargins method

You can specify additional indents inside the tooltip window with the `MarginLeft`, `MarginRight`, `MarginTop` and `MarginBottom` properties. In most cases these indents are equal, and the `SetAllMargins` method is used to set all these 4 properties to the same value:

```
Sub SetAllMargins( _  
    ByVal MarginSize As Long _  
)
```

### The Supports function

hTooltip is a handy wrapper for the native Windows tooltip functionality, and the provided functionality may depend on the version of Windows. For example, some features, such as the balloon tooltip style, may be unavailable in early versions of Windows like Windows 95 or 98. Fortunately, all hTooltip features are available starting from Windows XP. The [Compatibility](#) section in this document specifies the hTooltip features that were not available in the older version of Windows.

If your app with hTooltip can be used in Windows 2000 or an earlier version of the OS, you can determine whether specific tooltip features are available with the `Supports` function:

```
Function Supports( _  
    ByVal SupportedFeature As ESupportedFeatures _  
) As Boolean
```

This function accepts an item from the `ESupportedFeatures` enumeration and returns a Boolean value indicating whether the corresponding feature is available. The `ESupportedFeatures` enumeration has two items, `httSupportBalloonStyle` (0) and `httSupportTitleAndIcon` (1), which correspond to the abilities to display balloon tooltips and to provide title and/or icon respectively. For instance, the following expression returns True if you can use balloon tooltips in your app:

```
objTT.Supports(httSupportBalloonStyle)
```

### The InitWithPattern method

[The source code for this section is stored in the Tutorial 7 – Tooltip patterns](#)

If you create a large number of tooltips with similar properties, you can simplify your code and automate the process of initialization of your tooltips with tooltip patterns. Tooltip pattern is an object that stores the

properties of the tooltip but does not do any real work. In the hTooltip component a tooltip pattern is represented with the instance of the `CTooltipPattern` class, and the following `InitWithPattern` method is used to initialize the tooltip object with the values of the properties of the specified pattern:

```
Sub InitWithPattern( _  
    ByVal TooltipPattern As CTooltipPattern _  
)
```

For instance, you want to provide the same tooltip view and functionality for all controls in your form. Say you need to display white text on blue background in your tooltips, each tooltip must appear practically at the same time when the user places the mouse pointer inside a control (the `DelayTime` parameter should be 50ms) and the tooltip window should remain visible for 7 seconds. And in addition, each tooltip should have the balloon style and must be centered under the tool it is attached to.

To achieve this, we create the corresponding instance of the `CTooltipPattern` class in our code:

```
Dim objTTP As New CTooltipPattern  
With objTTP  
    .DelayTime = 50  
    .VisibleTime = 7000  
    .BackColor = vbBlue  
    .ForeColor = vbWhite  
    .Centered = True  
    .Style = httStyleBalloon  
End With
```

Let's assume we have two command buttons (Command1, Command2) and one text box (Text1) on our form, and we need to attach tooltips which are different in tooltip text to these controls. Then we simply use this pattern to initialize our tooltips:

```
Set objTT1 = New CTooltip  
objTT1.InitWithPattern objTTP  
objTT1.Text = "Command1"  
objTT1.CreateForVBCtrl Command1  
  
Set objTT2 = New CTooltip  
objTT2.InitWithPattern objTTP  
objTT2.Text = "Command2"  
objTT2.CreateForVBCtrl Command2  
  
Set objTT3 = New CTooltip  
objTT3.InitWithPattern objTTP  
objTT3.Text = "Text1"  
objTT3.CreateForVBCtrl Text1
```

This code will be supported easily in the future too. See, if you need to change for example the `DelayTime` parameter for all your tips, you can simply do it in the pattern with one line of code.

### The ShowAboutBox method

In fact, this method does not do any tooltip-related work, it simply displays the About dialog that contains the information about the current version of the hTooltip component (such as the author, the version, etc.):

```
Sub ShowAboutBox()
```

### Tooltip events

#### The MouseEnter and MouseLeave events

These are the only events raised by the `CTooltip` class. The `MouseEnter` event occurs when the mouse pointer enters the tool the tooltip is attached to, the `MouseLeave` one is triggered when the mouse pointer leaves the area occupied by the tool. Note that these events are fired properly even if the tool is a rectangle

inside a real control or an entire window (when the tooltip window was created with the `CreateForRect` method).

In some cases you need to set the `MouseEnterOnCreation` property of the `CTooltip` object to `True` to properly raise these events because of the internal window organization in Windows. It is required if (1) you create tooltips from the `MouseMove` event in your controls and (2) your form is cluttered with a large amount of controls which are placed next to each other. When you move the mouse pointer very fast over these controls, the `MouseEnter` and `MouseLeave` events may be missed for some controls in this scenario, and setting the `MouseEnterOnCreation` to `True` fixes this issue. In fact, the `MouseEnter` event is raised automatically when you invoke one of the `Create*` methods, and after that the tooltip object 'knows' that it should check whether the mouse pointer leaves the tool's area in order to raise the `MouseLeave` event properly.

## Advanced Topics

### Multiline tooltips

[The source code for this section is stored in the Tutorial 4 - Multiline tooltip](#)

You can create a multiline tooltip if you do the following:

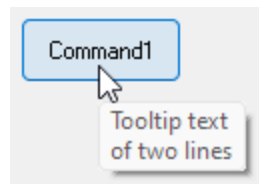
- 1) Insert the CR (carriage return, the ASCII code is 13) and LF (line feed, the code is 10) characters into the tooltip text in the places where you need a new line of text.
- 2) Assign a value to the `MaxWidth` property. In this case the component breaks the tooltip text by itself trying to keep up the specified maximum width.

If required, both ways can be used for the same tooltip.

Below you see an example of a multiline tooltip created with the first way:

```
objTT.Text = "Tooltip text" & vbCrLf & "of two lines"
```

This setting creates the following tooltip with two lines of text:



NB: you can only create multiple lines in the tooltip's text, not in its title.

### Using tab stops

hTooltip lets you create tabular formatted text using system tab stops. You specify tabulation in your text simply by inserting the TAB character (the ASCII code is 9).

Look at the following screenshot:



This tooltip can be created with the following code snippet:

```
objTT.Icon = httIconInfo
objTT.Style = httStyleBalloon
objTT.Title = "Louisiana"
objTT.Text = "Capital: " & vbTab & "Baton Rouge" & vbCrLf & _
    "Population: " & vbTab & "4 468 976"
```

### Setting tooltip font

By default, hTooltip uses the default system tooltip font. Its `Font` property is set to `Nothing` by default to indicate this fact.

If you need to assign your own font to a tooltip, you first need to create an instance of the `StdFont` object and assign a reference to it to the `Font` property. After that you will be able to set the properties of your tooltip font.

Here is an example of how you can assign a new Tahoma bold 12pt font to your `objTT` tooltip object:



```
Set objTT.Font = New StdFont
objTT.Font.Name = "Tahoma"
objTT.Font.Size = 12
objTT.Font.Bold = True
```

If you want to revert to the default system tooltip font, assign *Nothing* to the *Font* property:

```
Set objTT.Font = Nothing
```

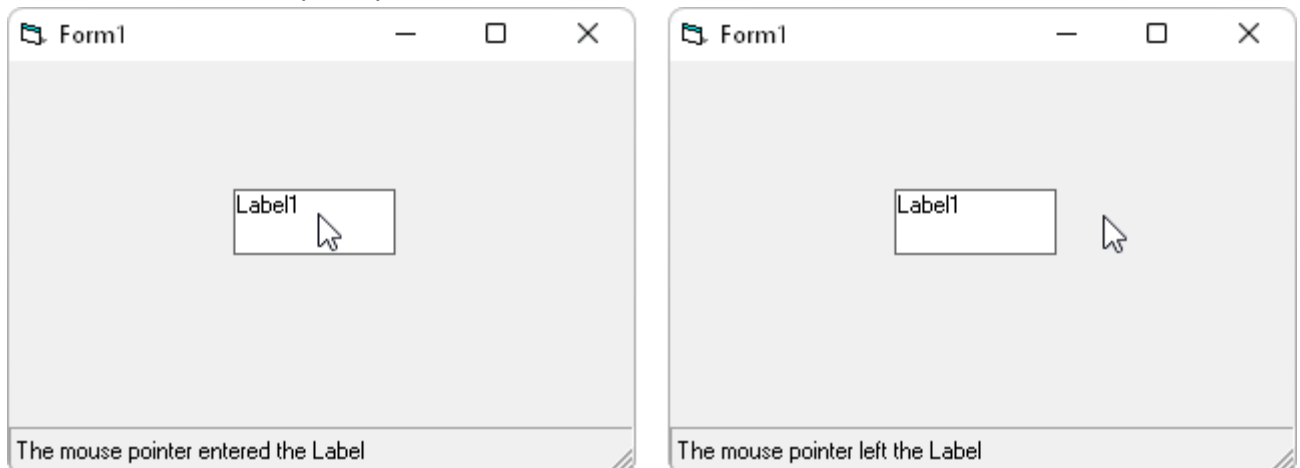
### Using the *MouseEnter*/*MouseLeave* events

[The source code for this section is stored in the Tutorial 3 - MouseEnter & MouseLeave](#)

VB6 and VBA lack the *MouseEnter*/*MouseLeave* events for their controls. You can emulate these events in your code by processing standard mouse events like *MouseMove*. You must do it for the control, the form it is placed on, and other controls on the form (if any). You must also write a lot of additional code that detects whether the control is overlapped by another control or even another app, but it is a hard and tedious work in the general case.

The *CTooltip* class provides you with a simplest way to get these events. When you create a tooltip for a control, the corresponding instance of the *CTooltip* class already fires these events – you simply need to handle them if you need it.

Let's consider an example in which we display a status bar message in a form when the mouse pointer enters and leaves the area occupied by a Label control:



As always, let's declare a tooltip object in the form module. Notice that now we should do it with the *WithEvents* keyword as we are going to process events:

```
Private WithEvents objTT As CTooltip
```

The next step is to create the corresponding instance of the *CTooltip* class and set its properties:

```
Private Sub Form_Load()
    Set objTT = New CTooltip

    With objTT
        .Title = "Tooltip title"
        .Text = "Tooltip text."
    End With

    objTT.CreateForVBCtrl Label1
End Sub
```

And finally we write the event handlers for the *MouseEnter* and *MouseLeave* events:

```
Private Sub objTT_MouseEnter()  
    StatusBar1.SimpleText = "The mouse pointer entered the Label"  
End Sub  
  
Private Sub objTT_MouseLeave()  
    StatusBar1.SimpleText = "The mouse pointer left the Label"  
End Sub
```

That's all. Launch the project and see – it works.

With the hTooltip component, you can even use the `MouseEnter` and `MouseLeave` events without displaying a tooltip. Remove the `With` statement in the code above in which we set up the tooltip (if you leave the `Text` property empty, the tooltip never appears):

```
Private Sub Form_Load()  
    Set objTT = New CTooltip  
    objTT.CreateForVBCtrl Label1  
End Sub
```

And you will see that the `MouseEnter`/`MouseLeave` events still works!

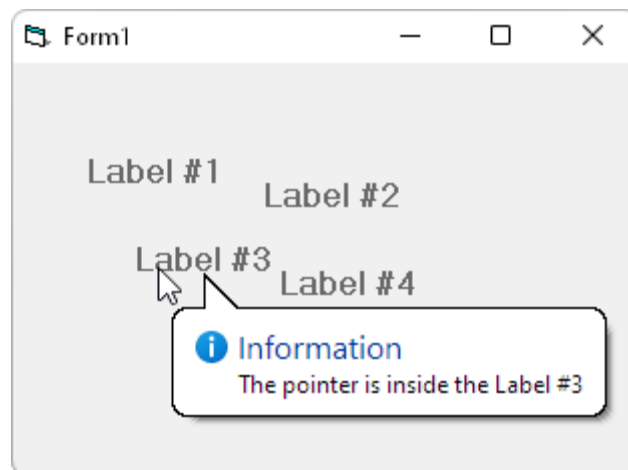
## ***Creating and displaying tooltips dynamically***

### **Scenario #1: A huge number of objects**

[The source code for this section is stored in the Tutorial 5 – Dynamic tooltips](#)

In this scenario you need to display a lot of typical tooltips for a large amount of objects in your code. Sometimes, the total number of these objects is even not known beforehand because they are created dynamically. The hTooltip component can also greatly simplify this task. You can create one instance of the `CTooltip` class in your code and dynamically assign it to each object when the mouse pointer hovers over the object.

Let's place several labels on a form and display a tooltip with the name of the label under the mouse pointer when the pointer hovers over one of our labels:



First, create a control array from our labels with the name `LabelArray`. Then declare our tooltip object and initialize it:

```
Private WithEvents objTT As CTooltip

Private Sub Form_Load()
    Set objTT = New CTooltip

    objTT.Title = "Information"
    objTT.Style = httStyleBalloon
    objTT.Icon = httIconInfo
    objTT.DisplayOnDemand = True
    objTT.MouseEnterOnCreation = True
End Sub
```

Pay attention to the fact that we do not create a real tooltip window and attach it to any control. We do it later in the `MouseMove` event for our labels:

```
Private Sub LabelArray_MouseMove(Index As Integer, Button As Integer, _
    Shift As Integer, X As Single, Y As Single)
    If Not objTT.IsCreated Then
        objTT.Text = "The pointer is inside the " & LabelArray(Index)
        objTT.CreateForVBCtrl LabelArray(Index)
        objTT.Show LabelArray(Index).Width / 2, LabelArray(Index).Height * 0.8, True
    End If
End Sub
```

And we also need to hide the displayed tooltip when the mouse pointer left the tool. The best place to do it is the `MouseLeave` event of the `CTooltip` class:

```
Private Sub objTT_MouseLeave()
    objTT.Destroy
End Sub
```

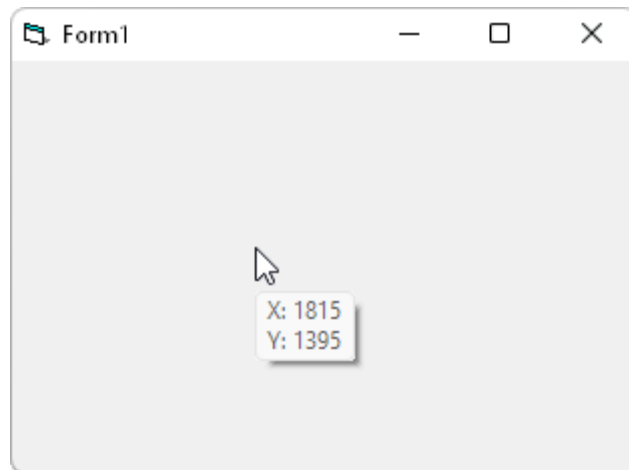
Some remarks to the code.

1. The `MouseMove` event can be raised several times for a Label, but we display the tooltip only once when the pointer enters the Label's area. We check whether the tooltip window is created in the `MouseMove` event with the `IsCreated` property. When the pointer leaves the Label's area, we destroy the tooltip window with the `Destroy` method. The method destroys only the tooltip window but not the entire tooltip object, and the `IsCreated` property returns False after that.
2. The tooltip remains visible until you move the mouse pointer outside the Label's area. This is possible due to the `NeverHide` parameter of the `Show` method we set to True. If we did not do this, this parameter would be set to False by default, and the tooltip would have disappeared after the period of time specified in the `VisibleTime` property.
3. When we initialize the tooltip, we set the `MouseEnterOnCreation` property to True. It is not required in our sample when we have several objects, but if you deal with a large number of objects, the `MouseEnter/MouseLeave` events may not be fired properly if you move the mouse very fast over your objects (due to the internal organization of the operating system). Setting the `MouseEnterOnCreation` property to True causes the `CTooltip` class to raise the `MouseEnter` event automatically when you create the tooltip window with the `Create*` method and the `MouseLeave` event is also fired properly in this case in the future.

## Scenario #2: Tracking tooltips

[The source code for this section is stored in the Tutorial 6 - Tracking tooltips](#)

With hTooltip you can implement so-called tracking tooltips that 'track' the mouse pointer (the tooltip follows it). The tooltip text can also be changed while the tooltip is moving. As an example, let's implement with hTooltip the application in which we display the coordinates of the mouse inside a form using our own multiline tooltip:



First, we create the tooltip object and initialize it:

```
Private WithEvents objTT As CTooltip

Private Sub Form_Load()
    Set objTT = New CTooltip
    objTT.DisplayOnDemand = True
    objTT.SmartPositioning = False
    objTT.CreateForHwnd Me.hWnd
End Sub
```

Note that we create the tooltip for the entire form using the form's API handle and the `CreateForHwnd` method of `CTooltip`.

Then we simply display our tooltip just below the mouse pointer and hide the tooltip when the pointer is out of the form (notice the default measurement unit for a form in VB6 – twips):

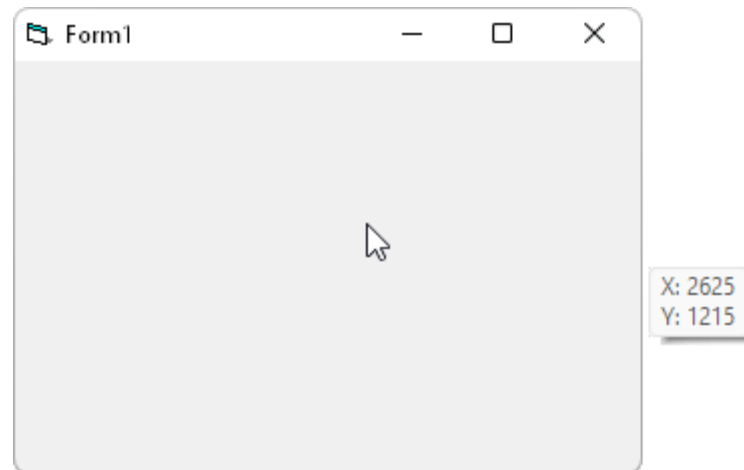
```
Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    objTT.Text = "X: " & X & vbCrLf & "Y: " & Y
    objTT.Show X, Y + 330, True
End Sub

Private Sub objTT_MouseLeave()
    objTT.Hide
End Sub
```

The same technique can be applied to any intrinsic control, not just to the whole form.

Note that we hide the tooltip with the `Hide` method, not `Destroy`. In the latter case the tooltip window would be destroyed, and we would need to recreate it again each time in the `MouseMove` event with the `CreateForHwnd` method.

We also turn off smart positioning for our tooltip with the `SmartPositioning` property. Otherwise the tooltip would be displayed near the form as smart positioning tries to position the tooltip window to not overlap the window it is attached to:



### **The ScaleMode property**

To simplify your programming in VB6, hTooltip uses twips as the default measurement unit because this is the default measurement unit in VB6. 'Twip' is a screen-independent unit used to ensure that placement and proportion of screen elements in your screen application are the same on all display systems. A twip is a unit of screen measurement equal to 1/20 of a printer's point. There are approximately 1440 twips to a logical inch or 567 twips to a logical centimetre (the length of a screen item measuring one inch or one centimetre when printed). In most cases, when you use the standard screen resolution in Windows (96 DPI, dots per inch), 1 pixel equals 15 twips.

But if you use hTooltip in another development environment, the measurement unit will in all likelihood be the pixel (FYI: you can also select this measurement unit for your forms in VB6). You must factor this in if you deal with coordinates in some hTooltip methods, such as [Show](#) or [CreateForRect](#). In this case the [CTooltip](#) and [CTooltipPattern](#) classes provide you with the [ScaleMode](#) property you should set to 'pixels' ([EScaleMode.httScalePixels](#) in VB6/VBA or simply 1 in other development environments).

For instance, the following Visual FoxPro code creates a balloon tooltip for the command button COMMAND1. Note that we set the [ScaleMode](#) property to 'pixels':

```
PUBLIC objHTT
objHTT = CREATEOBJECT("hTT110_10Tec.CTooltip")
objHTT.Text = "It is a real balloon tooltip"
objHTT.Title = "Balloon tooltip"
objHTT.Icon = 1
objHTT.Style = 1
objHTT.ScaleMode = 1
objHTT.CreateForRect(THISFORM.HWnd, THISFORM.COMMAND1.Left, THISFORM.COMMAND1.Top,
THISFORM.COMMAND1.Width, THISFORM.COMMAND1.Height)
```

### **Setting DelayTime and VisibleTime to system defaults**

You can assign -1 or any other negative value to the [DelayTime](#) or [VisibleTime](#) property to set the value of the property to the corresponding system default value. Note that after such an assignment the property returns a positive value in milliseconds that is in effect.

These properties of a tooltip pattern object (the instance of the [CTooltipPattern](#) class) are set to -1 by default. This causes any real tooltip object (the [CTooltip](#) class) to use the system default values as well.

### **Tooltip patterns and the DefaultTooltip global object**

[The first sample for this section is stored in the Tutorial 7 – Tooltip patterns](#)

[The second sample code for this section is stored in the Tutorial 8 - The DefaultTooltip object](#)

We have already described how you can simplify your work with the [InitWithPattern](#) method and tooltip patterns when you create a bunch of tooltips of the same kind (see the section devoted to this method). In VB6 and VBA in Microsoft Office you can even simplify coding even more using the [DefaultTooltip](#) global object.

The `DefaultTooltip` object is implemented by the hTooltip library and can be used anywhere in your VB6/VBA code without preliminary declaration as it is a global object. When you set the reference to the hTooltip DLL, the `DefaultTooltip` object appears in the IntelliSense list and is used in your code as if it were an intrinsic object.

The `DefaultTooltip` object has the `CTooltipPattern` type (like tooltip patterns) and in fact it stores the default values for the properties of the tooltip objects you create in your code (the instances of the `CTooltip` class).

Let us demonstrate how the code from the first sample (in which we demonstrate the `InitWithPattern` method) is simplified with the `DefaultTooltip` object. Here is the part of the original code:

```
Dim objTTP As New CTooltipPattern
With objTTP
    .DelayTime = 50
    .VisibleTime = 7000
    .BackColor = vbBlue
    .ForeColor = vbWhite
    .Centered = True
    .Style = httStyleBalloon
End With

Set objTT1 = New CTooltip
objTT1.InitWithPattern objTTP
objTT1.Text = "Command1"
objTT1.CreateForVBCtrl Command1

Set objTT2 = New CTooltip
objTT2.InitWithPattern objTTP
objTT2.Text = "Command2"
objTT2.CreateForVBCtrl Command2
```

We can avoid creating the tooltip pattern and issuing the `InitWithPattern` method if we set the required global tooltip properties through the `DefaultTooltip` object:

```
With DefaultTooltip
    .DelayTime = 50
    .VisibleTime = 7000
    .BackColor = vbBlue
    .ForeColor = vbWhite
    .Centered = True
    .Style = httStyleBalloon
End With

Set objTT1 = New CTooltip
objTT1.Text = "Command1"
objTT1.CreateForVBCtrl Command1

Set objTT2 = New CTooltip
objTT2.Text = "Command2"
objTT2.CreateForVBCtrl Command2
```

Be careful when changing the properties of `DefaultTooltip` as it will affect all the tooltips you subsequently create in your code. The second sample for this section demonstrates how the settings made in the `DefaultTooltip` object in one form have an effect in the other form displayed from the first one.

Notice also that the `DefaultTooltip` object stores the settings which are COPIED each time to a new tooltip object when you create one. These are not the global settings you can change in one statement for all the tooltips you currently have, `DefaultTooltip` is only a pattern, and its properties are copied by hTooltip internally to each new instance of `CTooltip` when you create it.

The `DefaultTooltip` object implements only one method – `Reset`:

```
Sub Reset()
```

This method is used to reset the properties of this object to their default values. These default values correspond to the default values of tooltip parameters in the operating system.

### ***The hTooltip ProgID and late binding***

hTooltip like any other COM object has its own ProgID (program identifier), and you can use it in your code to create the instances of the hTooltip classes. This approach is especially useful if your development environment supports only late binding for COM objects or you want to use late binding to work with the hTooltip component for some reasons. Late binding can also be used to work with different versions of the hTooltip component simultaneously in the same app.

The ProgID for the described version of hTooltip is 'hTT110\_10Tec'. Thus, the full ProgID for the `CTooltip` class is 'hTT110\_10Tec.CTooltip'. This ProgID is used, for instance, in Visual FoxPro code when you need to create a tooltip for the entire form:

```
PUBLIC objHTT
objHTT = CREATEOBJECT("hTT110_10Tec.CTooltip")
objHTT.Text = "It is a real balloon tooltip"
objHTT.Style = 1  && httStyleBalloon
objHTT.CreateForHwnd(THISFORM.HWnd)
```

Here is the equivalent code in Visual Basic:

```
Private objTT As Object

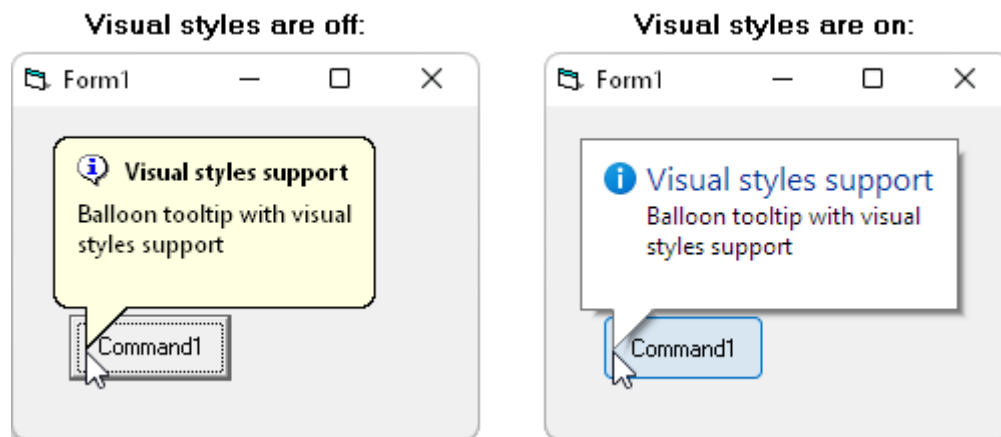
Private Sub Form_Load()
    Set objTT = CreateObject("hTT110_10Tec.CTooltip")
    objTT.Text = "It is a real balloon tooltip"
    objTT.Style = 1  ' httStyleBalloon
    objTT.CreateForHwnd Me.hWnd
End Sub
```

## Compatibility Info

### Windows visual styles support

[The source code for this section is stored in the Tutorial 9 - Windows visual styles](#)

The tooltips created with the hTooltip component are automatically rendered using the current Windows visual style if visual styles are enabled in your application. The main differences you can notice in tooltips when visual styles are turned on are the beautiful shadow behind the tooltip window and a more attractive, not denticulated, system icon. The tooltip window also appears using a fade-in effect. Compare the following two pictures of the same tooltip when visual styles are turned off and on:



Microsoft Office VBA provides built-in visual styles support in Microsoft Office since 2019. For other development environments, including Visual Basic 6, you must enable support for visual styles manually if the development environment does not provide this support out of the box.

There are several approaches how you can enable visual styles in your VB6 applications. One of them is implemented in the accompanying sample project. In a nutshell, we incorporated a special resource file into the VB6 project and enabled visual styles by initializing the Common Controls library with the `InitCommonControls` WinAPI function:

```
Private Declare Sub InitCommonControls Lib "comctl32.dll" ()
Private Declare Function LoadLibrary Lib "kernel32" Alias "LoadLibraryA" ( _
    ByVal lpLibFileName As String) As Long
Private Declare Function FreeLibrary Lib "kernel32" ( _
    ByVal hLibModule As Long) As Long
Private m_hMod As Long

Private Sub Form_Initialize()
    m_hMod = LoadLibrary("shell32.dll")
    InitCommonControls
End Sub

Private Sub Form_Unload(Cancel As Integer)
    ' unload shell32.dll - used only to turn on using visual styles
    FreeLibrary m_hMod
End Sub
```

Notice that visual styles work only if you launch a compiled exe but not in the VB6 IDE as this IDE does not support them.

### hTooltip features and the Windows Common Controls library

The tooltip features encapsulated in the hTooltip component rely on the Common Controls library (ComCtl32.dll) included in each version of Microsoft Windows. Depending on the version of this library, some features may not work.

Fortunately, practically all the tooltip features you can access through the hTooltip component are implemented by the version 5.80 of the Common Controls, and this version of the DLL is included in Windows 98 Second Edition and Windows 2000 or redistributed with Internet Explorer 5.0. If you have a later version



of Microsoft Windows or IE, the corresponding later version of ComCtl32.DLL is installed on your system, and of course, due to backward compatibility, all the tooltip features will work as well.

The following table lists the main hTooltip features and the versions of Windows and ComCtl32.dll they are rely on:

Feature	hTooltip Members	ComCtl	Comes with
ComCtl32.dll v5.80			
Balloon style	Property: <code>Style</code>	5.80	IE: 5 Win9X: 98SE WinNT: 2000 (but not originally in Windows 95,98/Windows NT 4.0)
Title	Property: <code>Title</code>		
Icon (only if has title)	Property: <code>Icon</code>		
ComCtl32.dll v5.80 and MS Windows 98/2000			
Slide effect	Property: <code>UseSlideEffect</code>	5.80	The sliding ToolTip animation is available on Windows 98 and Windows 2000 systems. This effect cannot be used on earlier systems. The effect works only for rectangular tooltips.
Fade effect	Property: <code>UseFadeEffect</code>	5.80	The fading ToolTip animation is available on Windows 98 and Windows 2000 systems. This effect cannot be used on earlier systems. The effect works only for rectangular tooltips.
ComCtl32.dll v4.70			
Display on demand	Property: <code>DisplayOnDemand</code> Method: <code>Show</code>	4.70	IE: 3 Win9x: Win95 OSR2 WinNT: 4.0
Margins changing	Properties: <code>Margin{Left Top Right Bottom}</code> Method: <code>SetAllMargins</code>		
Max width	Property: <code>MaxWidth</code>		
Smart positioning	Property: <code>SmartPositioning</code>		

The `Supports` function of the `CTooltip` class allows you to determine whether some of these features are available in the current environment.

### **Balloon tips availability**

Balloon tips can be disabled in the Windows OS even if the system supports them. Although this setting cannot be changed in the Windows OS's system settings, it can be switched using any registry editor app or with many 3rd-party system tweakers. Sometimes this option is also called 'system tray balloons' in those apps. Actually all balloon tooltips in the system are suppressed when you turn this option off.

If hTooltip balloon tips are not displayed at all, check the following registry key:

HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced

If it contains the `EnableBalloonTips` DWORD value set to 0, balloon tooltips are disabled in your system. To enable them, set `EnableBalloonTips` to 1 or even remove this value from the registry. You should restart your application to see the effect of this setting change.

## Known Problems and Limitations

1. The `DelayTime` and `VisibleTime` have the Integer data type in Visual Basic (2 bytes signed integer value). This means that the maximum value you can set each of these properties to is 32767 milliseconds. This is a restriction of the native tooltips in Windows.
2. If you create two tooltips for two rectangle regions with the `CreateForRect` methods and those two rectangles have an intersection, you may see the two tooltips when you pause the mouse pointer in the intersection region. Be careful if you have overlapped windowless controls (such as Label) in your VB6 form because `CreateForRect` is used internally to create tooltips for such controls.
3. A VB6 application crashes if you create tooltips for two or more Label controls twice. This should not be a problem in real-world apps because in the vast majority of cases a tooltip is created just once for a particular control when an application starts.