Contents

Introduction

The ***CCRP High Performance Timer Objects*** provide much more robust options than Visual Basic's intrinsic Timer control ever has. For years, the only option available to VB developers was to add a form to their project if they wanted to use a timer. This had the effect of totally destroying any hope of encapsulation if, for example, a class module needed periodic timer events.

No more superfluous forms!

This library provides **ccrpTimer**, **ccrpStopWatch**, and **ccrpCountdown** objects that may be instantiated anywhere, at any time. All three objects use the multimedia timer, thus providing true 1ms resolution (if supported by the hardware). Taking advantage of the **WithEvents** method of declaring an object instance, **Timer** and **Tick** events may be sunk in either forms or class modules.

Secondary Interfaces are exposed for your implementation as another way to recieve "event" notification. Using Implements in this manner allows notifications in situations where normal events would be blocked, and also provides a method to simulate control arrays using class objects! See **Demo: Notify.vbp** and **Demo: ObjArray.vbp** for a taste of the possibilities!

Client applications may specify the internal rate at which the ccrpTimers own multimedia timer is firing, thus allowing control over the amount of CPU being consumed. Timer intervals may be any value from the minimum supported by the system (typically 1ms) up to the maximum positive value for a Long (nearly 25 days!).

The ccrpTimers library was created by Karl E. Peterson (karl@mvps.org), a member of the VB Common Controls Replacement Project, a group aiming to provide separate ActiveX controls to replace and extend the controls found in COMCTL32.OCX, COMCTL232.OCX, and VB's intrinsic controls.

About Multimedia Timers
Multimedia timer raise a number of interesting issues. Some of them are dealt with in more detail in the **About Multimedia Timers** section of this help file. The ccrpTimers library provides the most efficient access your application can have to these valuable Windows resources.

Installing and Registering the Library
Understanding that the great bulk of CCRP's users will be sophisticated developers themselves, it has been decided that you're quite capable of manual installation of these components. See the **Installation** topic for complete details.

Included Demos
One demo is included (see: **Demo: TimerTest.vbp**) with this object library, consisting of three files (TimerTest.vbp, FTimerTest.frm, and FTimerTest.frx) which you may place where you find convenient. **TimerTest** is a project that demonstrates most of the features of all the CCRP Timer objects.

Service Pack Issues
The ccrpTimers library requires that either Service Pack 2 or Service Pack 3 be installed. A non-service pack version of the ccrpTimers object library is not currently available.

Videosoft (http://www.videosoft.com) has kindly provided CCRP with a copy of their VSDOCX software for documenting ActiveX controls. This help file was created using VSDOCX. CCRP would like to extend their sincere thanks to Videosoft for their generousity.

About CCRP

As the interface to the Windows common control dynamic link library, COMCTL32.OCX gives developers access to most of Window's common controls including the Listview, Toolbar, Statusbar, Progress Bar and Imagelist. Powerful and versatile when implemented via the API, for the developer who relies on comctl32.ocx there arise many disadvantages.

The comctl32.ocx file itself is over 600kb in size in size, making distribution impractical for small applications or internet pages needing only a fraction of what the control encapsulates. The control is also restricted in the features that it delivers, features users often associate as indicative of a 'state of the art' application. COMCTL32.OCX can also be very sluggish; the ocx is but an interface to the real 450kb Windows COMCTL32.DLL file, which contains the actual code for the common controls. In addition, Microsoft regularly updates COMCTL32.DLL to provide new features but these new features are not available through COMCTL32.OCX directly.

The Common Controls Replacement Project is a new and unique project, combining the extensive talents of Windows developers who's aim is to provide smaller, faster, more fully-featured and free replacements to the provided Microsoft Windows Common Controls, Common Dialog controls, and 'Intrinsic Controls' - those provided as part of the standard VB toolbar. The project now has fourteen members who pool their knowledge to create the CCRP controls.

For more information visit our website at http://www.zeode-sd.com/ccrp. Scan the pages for each control to see its current features and development status. Registration for any control is free by filling out a short form from the control's download link. And if you want to help out, see the information at the Join CCRP link.
For a comprehensive listing of the CCRP members, their projects and other information, including how to become a CCRP developer, see the CCRP Membership page.

Note that some CCRP controls specify minimum design and runtime requirements to achieve full functionality. These are listed along with each control's enhanced functionality on the control's information page, as well as the download page. The requirements very from whether or not SP2 is required to use a control to the version of COMCTL32.DLL needed for some properties, methods and events to function correctly. See **COMCTL32.DLL versions** for more information on the latter and the **Introduction** for more information on SP2.

Installation

Understanding that the great bulk of CCRP's users will be sophisticated developers themselves, it has been decided that you're quite capable of manual installation of these components. It's really quite simple:

* Unzip the contents of ccrpTmr.zip into a working directory.

* Move the following files into your System directory:

ccrpTmr.dll	
ccrpTmr.hlp	
ccrpTmr.cnt	
ccrpTmr.dep	
ccrpTmr.tlb	(optional)

* Optionally, download the HtmlHelp file from http://www.mvps.org/vb/ccrptmr.

ccrpTmr.chm	(optional)

* The remaining files make up the samples, and may be placed whereever convenient.

Registering the Library
There are two methods you can use to register the DLL.

* Drop into a DOS box, change to the System directory, and issue the following command:

```
C:\WinNT\System32\>regsvr32 ccrpTmr.dll
```

* From the VB5 **Project** menu, select **References...**, then press the **Browse** button, find and highlight the library, press **OK**.

If you get an error while attempting to register ccrpTmr.dll (the SP2 version of the DLL), it is likely that you do not have SP2 installed and hence you will need to download and use ccrpTmrs.ocx (see **Introduction** about SP2 issues).

Distribution and Licensing Agreement

Support, Etc.

Support
The CCRP provides a public internet newsserver as a primary means of support. This should be the first place to go, as you will not only recieve the author's attention, but also that of other users of the control or library in question. This resource is located at **news://news.mvps.org**.

For e-mail support, contact the author directly at **karl@mvps.org**.

Updates
Visit the official CCRP website at **http://www.mvps.org/ccrp** to download updates and other CCRP controls or get more information on this library.

Reporting Bugs / Submitting Feature Requests
To report bugs in the library or suggest features that you would like to see incorporated into a future version of the library send email to **karl@mvps.org**.

Further Resources
Swing by the author's web site at **http://www.mvps.org/vb** for numerous VB samples and other tools.

If you're interested in an HtmlHelp version of this helpfile, that's also available at the author's web site. This online (and downloadable) documentation will always be the most up-to-date, as it's the easiest to update and distribute. To view the documentation online, or download a compiled version, hit **http://www.mvps.org/vb/ccrptmr**.

About Multimedia Timers

Multimedia timer services allow applications to schedule timer events with the greatest resolution (or accuracy) possible for the hardware platform. These multimedia timer services allow you to schedule timer events at a higher resolution than other timer services. These timer services are useful for applications that demand high-resolution timing. For example, a MIDI sequencer requires a high-resolution timer because it must maintain the pace of MIDI events within a resolution of 1 millisecond.

Availability
The number of multimedia timers your application can create can vary depending on the operating system it is running under. In Windows NT timers are allocated on a per process basis, while in Windows 95/98 there is an absolute limit on the total number of timers system-wide.

Operating System	Timers
Windows 95/98	32 total for system
Windows NT	16 per process

User Interface Issues
If you attempt to update the screen at very high frequencies (more often than every 10ms, or so), prepare for fireworks. Windows 95 GDI is composed of much 16-bit code, and simply can't keep up. Hard-locks, and even system resets, can occur. If the **Interval** property is user configurable, test before updating the screen:

```
Private Sub Timer1_Timer()
   Static Ticks As Long
   Ticks = Ticks + 1
   '
   ' Updating the display more often than every 10ms can blow
   ' Win95's 16-bit GDI to shreds -- hardlock or system reset.
   '
   If Timer1.Interval >= 10 Then
      lblTimer1.Caption = " " & Format(Ticks, "#,##0")
   ElseIf (Ticks Mod 10) = 0 Then
      lblTimer1.Caption = " " & Format(Ticks, "#,##0")
   End If
End Sub
```

The Zen of Timers
The ccrpTimers library practices "safe timers." That is, internally, a single multimedia timer is used and each timer object is notified on each timer callback. The timer objects then, in turn, check to see if their specified Interval has elapsed, and if so raise a Timer event (or fire a notification method) to their client. This scheme insures the lowest possible impact on overall system performance.

Each timer object exposes a Stats property, which is actually a non-creatable class composed of data regarding the current state of the ccrpTimers library and the system multimedia timer. You may use the Stats.Frequency property set the internal interval used for the library timer. This is the rate at which all timer objects are notified of timer callbacks. It is not related to how often your client class or form will be notified, as long as this value is set to one equal to or lower than the Interval property of your timer object(s).

That last point is very important to keep in mind! The idea is to set the Stats.Frequency to a value that's as high as possible, without going higher than the timer Interval(s) you plan to use. In practical usage, there's really no need to go higher than 20 or 25 milliseconds, even if you only want a Timer event every few minutes or so. On most of today's hardware, there will be no measurable impact from such a setting.

However, if you set the Stats.Frequency to 20, then set a ccrpTimer.Interval to 10, you'll find that your events will only be half as regular as you desire. The whole point is to make it the highest common demoninator of all the timer object Intervals you plan to use. For example, setting Stats.Frequency to 30 and Interval to 100 doesn't make sense, as the timer callbacks will occur at 90 and 120 milliseconds. The 90 will not be enough to satisfy your Interval, so every notification will be at 120, or 20 milliseconds late. In this case, a setting of 20 or 25 would be far preferable.

Here's a little test you can run to observe how altering the Stats.Frequency setting will affect the accuracy with which your timer events are fired. Open a new project, set a Reference to the ccrpTimers library, and add the following code to the main form:

```
Option Explicit

Private WithEvents tmr As ccrpTimer

Private Sub Form_Load()
   Set tmr = New ccrpTimer
   With tmr
      ' Alter next value widely (1 to 400)
      ' to observe accuracy variations.
      .Stats.Frequency = 1
      .Interval = 200
      .Enabled = True
      Debug.Print "Resolution: "; .Stats.Resolution
   End With
End Sub

Private Sub tmr_Timer(ByVal Milliseconds As Long)
   Debug.Print Milliseconds
End Sub
```

You'll see that you'll come extremely close to hitting the desired Interval the lower the Stats.Frequency value goes, and that the target will get progressively sloppier the higher you set this value. What you need to ask yourself at that point is, just how much accuracy does your application require?

Perhaps one of the most important things to take from this little discussion is that timer events are something that need to be handled expeditiously. Get in, get out. If you try to do too much, you'll clog up the system in a hurry. If possible, it's always wisest to share system timers, thus reducing the overhead of having multiple timers firing.

Revision History

Version 1.20 (build 151), April 30, 1998

* Added the **ccrpTimer**.**NotifyEx** and **ccrpCountdown**.**NotifyEx** properties. These properties have been declared As Object, in anticipation of accomodating two new secondary interfaces, the two existing interfaces, and any and all new interfaces added in the future. Currently supported are two new additions -- ICcrpTimerNotifyEx and ICcrpCountdownNotifyEx -- and the two notification interfaces added in version 1.10. These new interfaces (See **Secondary Interfaces**) vary from the original in that they also pass references to the object firing their methods. With this identification available, it is easy to simulate control arrays with class objects.

* A new demo was added to illustrate simulated control array syntax via the new NotifyEx property and secondary interface event notification. See **Demo: ObjArray.vbp** for details.

* Fixed a limitation on the maximum value accepted for **Interval** properties. Previously, the Interval was restricted to the maximum supported by the operating system (typically 1,000,000ms in NT and 65,535ms in Win95/98). This restriction is no longer enforced, and you may set an Interval to any positive value within the range of a Long (1 to 2,147,483,647ms).

* Adjusted the values passed in the Milliseconds parameter of **Timer** and **Tick** events, so they more accurately represent the true period elapsed since the last event.

Version 1.11 (build 130), April 11, 1998

* Updated documentation to include information about new properties and interfaces.

* Added the **Tag** property to each timer object.

Version 1.10 (build 113), March 29, 1998

* Added the **ccrpTimer**.**Notify** and **ccrpCountdown**.**Notify** properties. This addition provides signficant new functionality by exposing two new interefaces, ICcrpTimerNotify and ICcrpCountdownNotify. Client programs may choose to use Implements to add this secondary interface to their forms, usercontrols, or classes. The advantage of using the secondary interface over the classic event model is two-fold. Foremost, interface notification is significantly faster than raised events. Also, raised events may be blocked within the Visual Basic IDE by message boxes or modal dialogs, while interface notifications continue to come through, thus making debugging signficantly easier.

* A new demo, **Demo: Notify.vbp**, was added to illustrate usage of the Notify property and secondary interface event notification.

Version 1.02 (build 100), March 15, 1998

* Initial public release.

Error Codes

Various properties may raise an error 380 ("Invalid property value") if illegal values are assigned. This is generally only in cases where reasonable assumptions can't be made.   For example, passing a negative value to one of the Interval properties.

Valid property ranges are:

ccrpTimer.Interval	>= 0	(0 defaults to MinimumResolution)
ccrpTimer.EventType	0 or 1	(Enumerated)
ccrpCountdown.Interval	>= 0	(0 defaults to MinimumResolution)
ccrpCountdown.Duration	> 0	

It is also possible that the system may be depleted of available multimedia timers at the time the ccrpTimers library is initialized. See **About Multimedia Timers** for more details on this, and the **Zen of Timers**.

No other properties should raise an error when set. If any other errors are raised, please notify the author at once:

Karl E. Peterson
karl@mvps.org

Secondary Interfaces

The following interfaces are exposed to allow notification of timer "events" without threat of being blocked by modal dialog boxes or message boxes. Although such blocking typically only occurs within the Visual Basic IDE, it can be a nuisance during the development process. In addition, event notification via an interface can be many times faster than through the inherently late-bound WithEvents model.

ICcrpTimerNotify
Implement the ICcrpTimerNotify interface within your class or form to be notified of all ccrpTimer events via a method call. This interface is designed for use in situations where a single timer object is being used. Inform a ccrpTimer object that your client is implementing this interface via the **Notify** property.

```
Option Explicit

Public Sub Timer(ByVal Milliseconds As Long)
End Sub
```

ICcrpTimerNotifyEx
Implement the ICcrpTimerNotify interface within your class or form to be notified of all ccrpTimer events via a method call. This interface is designed for use in situations where a multiple timer objects is being used. Provides a method to simulate control arrays. Inform a ccrpTimer object that your client is implementing this interface via the **NotifyEx** property.

```
Option Explicit

Public Sub Timer(ByVal Milliseconds As Long, ByVal Tmr As ccrpTimer)
End Sub
```

ICcrpCountdownNotify
Implement the ICcrpCountdownNotify interface within your class or form to be notified of all ccrpCountdown events via method calls. This interface is designed for use in situations where a single timer object is being used. Inform a ccrpCountdown object that your client is implementing this interface via the **Notify** property.

```
Option Explicit

Public Sub Tick(ByVal TimeRemaining As Long)
End Sub

Public Sub Timer()
End Sub
```

ICcrpCountdownNotifyEx
Implement the ICcrpCountdownNotify interface within your class or form to be notified of all ccrpCountdown events via method calls. This interface is designed for use in situations where a single timer objects are being used. Provides a method to simulate control arrays. Inform a ccrpCountdown object that your client is implementing this interface via the **NotifyEx** property.

```
Option Explicit

Public Sub Tick(ByVal TimeRemaining As Long, ByVal Tmr As ccrpCountdown)
End Sub

Public Sub Timer(ByVal Tmr As ccrpCountdown)
End Sub
```

Demo: Notify.vbp

The Notify sample demonstrates the use of Implemented interfaces as opposed to more traditional Events. The FNotify.frm file uses Implements ICcrpTimerNotify and Implements ICcrpCountdownNotify to sink events generated from two distinct instances of their related objects.

There are two main advantages in using implemented secondary interfaces rather than events. The first is performance -- potentially reaching 10x or better. The second is that events can be blocked by a MsgBox or modal dialog while running within the IDE. Secondary interfaces allow calls into your objects, forms or classes, even when events might otherwise be blocked.

To use the more efficient secondary interfaces, add code such as the following to the Declarations section of any class or form:

```
Implements ICcrpTimerNotify
Private tmrNotify As ccrpTimer
```

Note that WithEvents wasn't used in the declaration for the ccrpTimer object. There's no longer a need for that. After entering the Implements directive, a new entry will appear in the left-hand dropdown of your code window. Selecting "ICcrpTimerNotify" from this list will add a new procedure, and populate the right-hand dropdown list with all the procedures you must implement:

```
Private Sub ICcrpTimerNotify_Timer(ByVal Milliseconds As Long)
   ' code to handle Timer "events"
End Sub
```

You're now prepared to accept "events" as calls into your class or form. To actually tell the ccrpTimers library that this is what you want, simply pass a reference to your class or form into the Notify property:

```
Private Sub Form_Load()
   Set tmrNotify.Notify = Me
End Sub
```

Similarly, if you choose to implement the ICcrpCountdownNotify interface, you would use the following code:

```
Implements ICcrpCountdownNotify
Private cntNotify As ccrpCountdown

Private Sub Form_Load()
   Set cntNotify.Notify = Me
End Sub

Private Sub ICcrpCountdownNotify_Tick(ByVal TimeRemaining As Long)
   ' code to handle Tick "events"
End Sub

Private Sub ICcrpCountdownNotify_Timer()
   ' code to handle Timer "events"
End Sub
```

You can, of course, implement both secondary interfaces, as the Notify project demonstrates.

Demo: TimerTest.vbp

The TimerTest sample VB5 application displays all the functionality provided by ccrpTimers. In it you will see how easy it is, using WithEvents, to create drop-in **ccrpTimer** replacements for VB's intrinsic Timer control.

Also shown is how to measure elapsed type using the **ccrpStopWatch** object, and set off a countdown with intermittent notifications using the **ccrpCountdown** object. Dig into this sample, and see just how simple it is to bump your resolution up to true millisecond accuracy!

The three files that compose **TimerTest** (TimerTest.vbp, FTimerTest.frm, FTimerTest.frx) may be placed anywhere on your hard disk. To load the project, first ensure that you have properly **registered** (see **Introduction**) the **ccrpTimers** object library.

Demo: ObjArray.vbp

The ObjArray sample demonstrates the use of Implemented interfaces as opposed to more traditional Events. The FTmrArray.frm file uses Implements ICcrpTimerNotifyEx and Implements ICcrpCountdownNotifyEx to sink events generated from two simulated control arrays of five distinct instances of their related objects.

There are two main advantages in using implemented secondary interfaces rather than events. The first is performance -- potentially reaching 10x or better. The second is that events can be blocked by a MsgBox or modal dialog while running within the IDE. Secondary interfaces allow calls into your objects, forms or classes, even when events might otherwise be blocked.

A third advantage is demonstrated by ObjArray -- the ability to work with an array of objects that are capable of firing events. Since the "events" are actually interface methods being fired into the client, there is no need to use WithEvents in the array declaration. Simply add code such as the following to the Declarations section of any class or form:

```
Implements ICcrpTimerNotifyEx
Private m_Tmr(0 To 4) As ccrpTimer
```

Note that WithEvents wasn't used in the declaration for the ccrpTimer object array. There's no longer a need for that. After entering the Implements directive, a new entry will appear in the left-hand dropdown of your code window. Selecting "ICcrpTimerNotifyEx" from this list will add a new procedure, and populate the right-hand dropdown list with all the procedures you must implement:

```
Private Sub ICcrpTimerNotifyEx_Timer( _
   ByVal Milliseconds As Long, _
   ByVal Tmr As ccrpTimers.ccrpTimer)
      ' code to handle Timer "events"
End Sub
```

You're now prepared to accept "events" as calls into your class or form. The reference to the timer library object provides your "Index" into the array. Several methods are used in the ObjArray demo to show the variety of approaches that can be used in interpreting the passed object reference. Perhaps the simplest is to place an Index value into the object's Tag property.

To tell the ccrpTimers library what object to notify of timer events, simply pass a reference to your class or form into the NotifyEx property:

```
Private Sub Form_Load()
   Dim i As Long
   For i = LBound(m_Tmr) To UBound(m_Tmr)
      Set m_Tmr(i) = New ccrpTimer
      Set m_Tmr(i).NotifyEx = Me
   Next i
End Sub
```

Similarly, if you choose to implement the ICcrpCountdownNotifyEx interface, you would use code similar to the following:

```
Implements ICcrpCountdownNotifyEx
Private m_Cnt(0 To 4) As ccrpCountdown

Private Sub Form_Load()
   Dim i As Long
   For i = LBound(m_Cnt) To UBound(m_Cnt)
```

```
         Set m_Cnt(i) = New ccrpCountdown
         Set m_Cnt(i).NotifyEx = Me
      Next i
   End Sub

   Private Sub ICcrpCountdownNotify_Tick( _
      ByVal TimeRemaining As Long, _
      ByVal Tmr As ccrpTimers.ccrpCountdown)
         ' code to handle Tick "events"
   End Sub

   Private Sub ICcrpCountdownNotify_Timer( _
      ByVal Tmr As ccrpTimers.ccrpCountdown)
         ' code to handle Timer "events"
   End Sub
```

You can, of course, implement both secondary interfaces, as the ObjArray project demonstrates.

Editorial: No More Lame Benchmarks!

Warning: This might make sense
If you're like me, you're sick and tired of seeing really lame methods to time VB code. The most common mistakes include:

* using VB's own Timer function, with its clock-tick resolution of 55ms,
* not timing enough iterations to make for meaningful comparisons,
* not eliminating code that shouldn't be timed,
* not subtracting the time of code that couldn't be eliminated, and
* testing in the IDE rather than from an EXE.

Obviously, any results published that didn't account for the above are worse than worthless. So, I'm providing my ccrpTimers library as freeware, and encouraging folks to use it. The **ccrpStopWatch** object provides millisecond resolution on most PCs, by employing the multimedia timer call timeGetTime. Using it is almost too easy! Check out this example:

A complete project, benchmarking the difference between 100,000 calls to Unicode and ANSI versions of GetWindowsDirectory:

```
Option Explicit

Private Declare Function GetWindowsDirectoryA Lib "kernel32" _
   (lpBuffer As Any, ByVal nSize As Long) As Long
Private Declare Function GetWindowsDirectoryW Lib "kernel32" _
   (lpBuffer As Any, ByVal nSize As Long) As Long

Private Sub Form_Click()
   Dim tmr As ccrpStopWatch
   Dim tL As Long
   Dim t1 As Long, t2 As Long
   Dim i As Long
   Dim BufferA As String
   Dim BufferW() As Byte
   Const Loops As Long = 100000
   Const MAX_PATH = 260
   '
   ' Create instance of stopwatch class
   '
   Set tmr = New ccrpStopWatch
   '
   ' Determine overhead of looping
   '
   tmr.Reset
   For i = 1 To Loops
   Next i
   tL = tmr.Elapsed
   '
   ' See how long it takes to make (Loops) calls
   ' to an ANSI function
   '
   BufferA = Space(MAX_PATH)
   tmr.Reset
   For i = 1 To Loops
      Call GetWindowsDirectoryA(ByVal BufferA, MAX_PATH)
   Next i
   t1 = tmr.Elapsed
   '
   ' See how long it takes to make (Loops) calls
   ' to Unicode version of same function
   '
   ReDim BufferW(0 To (MAX_PATH * 2) - 1) As Byte
   tmr.Reset
   For i = 1 To Loops
      Call GetWindowsDirectoryW(BufferW(0), MAX_PATH)
   Next i
   t2 = tmr.Elapsed
   '
   ' Output results, subtracting loop overhead
   '
   Me.Cls
   Me.Print "Loop Overhead:", tL; " ms"
   Me.Print "ANSI Calls:", , t1 - tL; " ms"
   Me.Print "Unicode Calls:", t2 - tL; " ms"
   Debug.Print "Loop Overhead:", tL; " ms"
   Debug.Print "ANSI Calls:", , t1 - tL; " ms"
   Debug.Print "Unicode Calls:", t2 - tL; " ms"
End Sub
```

Tests conducted on a dual-P6/200, with 128Mb RAM, running NT4/SP3 and VB5/SP2.

IDE Results

Loop Overhead:	16 ms
ANSI Calls:	1672 ms
Unicode Calls:	109 ms

EXE Results

Loop Overhead:	0 ms
ANSI Calls:	1422 ms
Unicode Calls:	78 ms

Conclusions

Amazing what a difference there is between making Unicode and ANSI calls, isn't there? That aside, how much simpler could it be to accurately benchmark your VB code? Hopefully, you'll find this methodology not only useful, but much more accurate than any others out there!

COMCTL32.DLL versions

Microsoft frequently provides updates to COMCTL32.DLL. These updates add new features to the controls found in COMCTL32.DLL. At the time of writing, there are four versions of COMCTL32.DLL available:

version 4.00 - Installed by Windows 95
version 4.70 - Installed by Internet Explorer 3
version 4.71 - Installed by Internet Explorer 4
version 4.72 - A version released after Internet Explorer 4 which for the first time is redistributable.

The ComCtlVer will identify which of these is installed on the System:

2 - Win95
3 - IE3
4 - IE4

Currently the control does not differentiate between 4.71 and 4.72.

Version-specific features
The CCRP High Performance Timer Objects library will function without regard to what version of the common controls you have installed, since it uses system multimedia timer services and does not use any features of COMCTL32.DLL.

Obtaining the COMCTL32.DLL redistributable update
In order to redistribute version 4.72 of COMCTL32.DLL, you must download a self-extracting EXE from Microsoft's web site. You must then call the EXE from within your setup program. *You may not simply copy COMCTL32.DLL in your setup program.* CCRP highly recommend that you distribute this update to your clients so that this and other CCRP controls will function fully. The self-extracting EXE can be downloaded from:

http://www.microsoft.com/msdn/downloads/files/40Comupd.htm

The self-extracting EXE is called 40comupd.exe and can be invoked silently from your setup program.

ccrpTimers Library

Controls

Description:	CCRP: High Performance Timers Objects
Library:	ccrpTimers
File Name:	ccrpTmr.dll
Help File:	ccrpTmr.hlp
GUID:	{d1866ac5-bec8-11d1-bbac-0055003b26de}
Objects:	ccrpStopWatch, ccrpTimer, ccrpCountdown

About this Help File
As noted elsewhere, this help file was generated using the VSDOCX tool from Videosoft. It's a *very slick* method to automate much of the process behind documenting OCX and DLL files. However, like most automation tools, it falls down at times too. It would seem Videosoft didn't spend much time testing against DLLs, as the output is definitely geared towards OCX documentation. Please overlook "control" or "controls" when you see those words (for example, the link at the top of this page <sigh>), and instead read them as "object" or "objects." Similarly, ignore references to forms in the **Syntax** listings for each property and method. There appears to be no way to keep those out of the helpfile without a lot of tedious manual editing. <deep sigh> Nonetheless, VSDOCX is very cool, and getting close to the point where I'd unhesitatingly recommend it.

ccrpTimer Object

Properties Methods Events

Object Name:	ccrpTimer
Description:	Extremely accurate replacement for VB's standard Timer control
File Name:	ccrpTmr.dll
Help File:	ccrpTmr.hlp
GUID:	{d1866abd-bec8-11d1-bbac-0055003b26de}
Properties:	7
Events:	1
Methods:	1

Before you can use **ccrpTimer** object in your application, you must add the **ccrpTmr.dll** file to your project. If you use the object in most of your VB projects, you may want to add it to VB's Autoload file.

To distribute applications you create with the **ccrpTimer** object, you must install and register it on the user's computer. The Setup Wizard provided with Visual Basic provides tools to help you do that. Please refer to the Visual Basic manual for details. A dependency file (ccrpTmr.DEP) has been included for your convenience.

General Usage
When declaring a new instance of **ccrpTimer**, use the special syntax first made available with VB5 to do so with full event support. Place a declaration similar to the following in the Declarations section of either a form or class module:

```
Private WithEvents Timer1 As ccrpTimer
```

At this point, you'll note that **Timer1** appears in the objects dropdown within that module's code window. By selecting Timer1 from this list, a new event will be placed into your code:

```
Private Sub Timer1_Timer()

End Sub
```

Typically, you'll want to initialize the ccrpTimer object in either the Form_Load or Class_Initialize event:

```
Set Timer1 = New ccrpTimer
```

At any point after initializing the object, you may start setting its various property values. Use **Interval** to set how often the **Timer** event should occur. The **EventType** property specifies whether to notify you continually (just like VB's Timer control) or to set up a single occurance event.   As with VB's Timer control, **Enabled** turns the ccrpTimer object on and off.

As with the other objects in the ccrpTimers library, properties are exposed via the **Stats** object to reveal the **MinimumResolution** and **MaximumResolution** settings supported on the current hardware. Additionally, a **Resolution** property is exposed to inform you what resolution the library is currently using. You may also set or read the **Frequency** of the timer being used by ccrpTimers through **Stats**.

In general, your application should not set a frequency higher than absolutely necessary, to avoid forcing the CPU to process extraneous hardware interrupts. See **About Multimedia Timers** for more details.

About Method (ccrpTimer Object)

See Also   Examples   Applies to

Shows About box for ccrpTimers.

Syntax

[*form*!]*ccrpTimer*.**About**

Remarks

This method is provided solely to provide you, the user, with a method of determining the origin of this DLL.

Enabled Property (ccrpTimer Object)

See Also   Examples   Applies to

Returns/sets a value that determines whether Timer events will occur as specified.

Syntax

[*form*!]*ccrpTimer*.**Enabled**[= {**True | False**}]

Remarks

Controls whether or not **Timer** events will fire every [**Interval**] milliseconds.

Disabling a Timer object by setting **Enabled** to **False** kills the object's internal timer, and prevents any further **Timer** events from occuring. **Timer** events only occur when a Timer object's **Enabled** property is set to **True**.

Settings	**Description**
True	Allows object to trigger to events.
False	(Default) Prevents object from triggering events.

Data Type

Boolean

Default Value

False

EventType Property

See Also   Examples   Applies to

Returns/sets a value that determines whether timer events will be periodic (recurring) or once-only.

Syntax

[*form*!]*ccrpTimer*.**EventType**[= *TimerEventTypes*]

Settings

Valid settings for the **EventType** property are:

Value	**Constant**
0	TimerOneShot
1	TimerPeriodic

Remarks

The **EventType** property controls whether **Timer** events fire continuously, or only once.

Settings	**Description**
TimerOneShot	Event occurs once, after [**Interval**] milliseconds.
TimerPeriodic	(Default) Event occurs every [**Interval**] milliseconds.

Data Type

TimerEventTypes (Enumeration)

Default Value

tmrPeriodic

Interval Property (ccrpTimer Object)

See Also   Examples   Applies to

Returns/sets the number of milliseconds between calls to the object's Timer events.

Syntax
[*form*!]*ccrpTimer*.**Interval**[= *value As Long*]

Remarks
The **Interval** property of the **Timer** object determines how often, in milliseconds, **Timer** events will fire.

Set the **Enabled** property to **True** to enable, or false to disable, **Timer** events.

Important Note
If you attempt to update the screen at very high frequencies (more often than every 10ms, or so), prepare for fireworks. Windows 95 GDI is composed of much 16-bit code, and simply can't keep up. Hard-locks, and even system resets, can occur. If the **Interval** is user configurable, test before updating the screen:

```
Private Sub Timer1_Timer()
   Static Ticks As Long
   Ticks = Ticks + 1
   '
   ' Updating the display more often than every 10ms can blow
   ' Win95's 16-bit GDI to shreds -- hardlock or system reset.
   '
   If Timer1.Interval >= 10 Then
      lblTimer1.Caption = " " & Format(Ticks, "#,##0")
   ElseIf (Ticks Mod 10) = 0 Then
      lblTimer1.Caption = " " & Format(Ticks, "#,##0")
   End If
End Sub
```

Data Type
Long

Default Value
100   (1/10 second)

Notify Property (ccrpTimer Object)

See Also Examples Applies to

Secondary interface provided for notification via method calls rather than events.

Syntax
[*form*!]*ccrpTimer*.**Notify**[= *ICcrpTimerNotify*]

Remarks
The **ccrpTimer** object will call methods within an object (form or class) that implements the *ICcrpTimerNotify* interface, and registers itself using this property. There are several advantages to choosing this approach, as opposed to the more classic WithEvents declaration. Principally, interfaces are early-bound, and hence much faster in execution. Secondly, events may be blocked (within the IDE) by such things as message boxes or modal dialogs. Therefore, debugging with Implements is much easier than debugging with WithEvents.

Data Type
ICcrpTimerNotify

Default Value
Nothing

NotifyEx Property (ccrpTimer Object)

See Also   Examples   Applies to

Secondary interface provided for notification via method calls rather than events. Supports identification of originating object.

Syntax
[*form*!]*ccrpTimer*.**NotifyEx**[= *object*]

Remarks
The **ccrpTimer** object will call methods within an object (form or class) that implements the ICcrpTimerNotifyEx interface, and registers itself using this property. There are several advantages to choosing this approach, as opposed to the more classic WithEvents declaration. Principally, interfaces are early-bound, and hence much faster in execution. Secondly, events may be blocked (within the IDE) by such things as message boxes or modal dialogs. Therefore, debugging with Implements is much easier than debugging with WithEvents.

This property has been declared As Object so that it can accomodate a variety of interfaces. The intention is that all supported interfaces may be passed here. Presently, the **ICcrpTimerNotify** and **ICcrpTimerNotifyEx** interfaces are supported. ICcrpTimerNotifyEx is identical to ICcrpTimerNotify with one very important exception. With each method, a reference to the ccrpTimer object firing that method is passed (see **Secondary Interfaces** for details). This allows simulation of control arrays, as well as the ability to sink "events" from multiple timer objects via a single interface without the need to resort to use of WithEvents. See **Demo: ObjArray.vbp** for complete details on implementation.

Valid Interfaces
ICcrpTimerNotify, ICcrpTimerNotifyEx

Data Type
Object

Default Value
Nothing

Stats Property (ccrpTimer Object)

See Also   Examples   Applies to

Provides a class of information relevant to the current system's multimedia timer.

Syntax
[*form*!]*ccrpTimer*.**Stats**[= *ccrpTimerStats*]

Remarks
The **Stats** property of all ccrpTimers objects provides information regarding the current state of the system's multimedia timer and how the ccrpTimers library is using it.

Frequency
This property may be used to set or return the frequency of the multimedia timer being used by the ccrpTimers library. Internally, ccrpTimers runs off a single multimedia timer, and services all classes based on that timer. At initialization, this timer is set to fire once every 1 millisecond, to offer users the most frequent events possible. However, if you find that you need events fired at a lower frequency, you can conserve CPU usage by setting ccrpTimers to use a higher frequency.

Important Note: All objects exposed by ccrpTimers library use a single timer, thus a single frequency! Optimization of your CPU usage is achieved by selecting the Highest Common Denominator of all timer objects to use as your desired Frequency.

Resolution
This property returns the functional resolution, or accuracy, of the multimedia timers requested by ccrpTimers library. Resolution is considered the accuracy with which timer events occur. For example, if you request a timer event every 100 milliseconds, and the system's minimum resolution is 5 milliseconds, the timer events will occur within a range of 95 to 105 milliseconds apart. The ccrpTimers library requests a resolution of 1 millisecond, although the Frequency may be varied. It is important to recognize that resolution *requests* are just that -- a request for a desired setting -- and are entirely dependent on the hardware being used.

MinimumResolution and **MaximumResolution**
These properties are provided for informational purposes only. They is a measure of the minimum and maximum resolutions, or accuracy, of the multimedia timer on the hardware being tested. Resolution is considered the accuracy with which timer events occur. For example, if you request a timer event every 100 milliseconds, and the system's minimum resolution is 5 milliseconds, the timer events will occur within a range of 95 to 105 milliseconds apart. Most modern PCs are capable of true 1 millisecond minimum resolution, and generally return 1,000,000 under NT and 65,535 under Windows 95 as their maximum resolution.

Data Type
ccrpTimerStats

Tag Property (ccrpTimer Object)

See Also Examples Applies to

Stores any extra data needed by your program.

Syntax
[*form*!]*ccrpTimer*.**Tag**[= *value As Variant*]

Remarks
A place for you to store any information required to uniquely identify, or related to, each instance of this object. The setting of this property in no way affects the behavior of the object.

Data Type
Variant

Default Value
Empty

Timer Event (ccrpTimer Object)

See Also   Examples   Applies to

Event that fires whenever [Interval] milliseconds elapses.

Syntax
Private Sub *ccrpTimer*_**Timer**(ByVal *Milliseconds* As Long)

Remarks
Timer events occur at the first opportunity following [**Interval**] milliseconds when the Timer object is **Enabled**. In situations where the processor is busy, multimedia timer events take low precedence and hence a **Timer** event may actually be several milliseconds late.

The **Milliseconds** parameter to this event reports how long it's been since the last time the event has been called or since the object has been enabled.

Use the **Interval** property to set how far apart **Timer** events should occur.

Set the **Enabled** property to **True** to enable, or false to disable, **Timer** events.

Based on the setting for the **EventType** property, **Timer** events will either be ongoing or "one-shot."

ccrpStopWatch Object

Properties   Methods   Events

Object Name:	ccrpStopWatch
Description:	Extremely accurate method to determine elapsed time.
File Name:	ccrpTmr.dll
Help File:	ccrpTmr.hlp
GUID:	{d1866abb-bec8-11d1-bbac-0055003b26de}
Properties:	3
Events:	0
Methods:	2

Before you can use **ccrpStopWatch** object in your application, you must add the **ccrpTmr.dll** file to your project. If you use the object in most of your VB projects, you may want to add it to VB's Autoload file.

To distribute applications you create with the **ccrpStopWatch** object, you must install and register it on the user's computer. The Setup Wizard provided with Visual Basic provides tools to help you do that. Please refer to the Visual Basic manual for details. A dependency file (ccrpTmr.DEP) has been included for your convenience.

General Usage
When declaring a new instance of **ccrpStopWatch**, do so as you would any other object:

```
Dim sw As ccrpStopWatch
Set sw = New ccrpStopWatch
```

At the moment the object is initialized (with the Set statement above), the time is noted. At any point after that, you may query the **Elapsed** property of the object to obtain the number of milliseconds that have gone by. Call the **Reset** method to set the elapsed count back to zero.

As with the other objects in the ccrpTimers library, properties are exposed via the **Stats** object to reveal the **MinimumResolution** and **MaximumResolution** settings supported on the current hardware. Additionally, a **Resolution** property is exposed to inform you what resolution the library is currently using. You may also set or read the **Frequency** of the timer being used by ccrpTimers through **Stats**.

The StopWatch object is extremely useful for benchmarking your code. See **Editorial: No More Lame Benchmarks!** for one technique.

About Method (ccrpStopWatch Object)

See Also   Examples   Applies to

Shows About box for ccrpTimers

Syntax
[*form*!]*ccrpStopWatch*.**About**

Remarks
This method is provided solely to provide you, the user, with a method of determining the origin of this DLL.

Elapsed Property

See Also   Examples   Applies to

Returns the number of milliseconds elapsed since creation of the Stopwatch object or invocation of its Reset method.

Syntax
value As Long = [*form*!]*ccrpStopWatch*.**Elapsed**

Remarks
The StopWatch object works by calling the **timeGetTime** API at initialization and whenever its **Reset** method is called, and then storing this time for future use. When you query the **Elapsed** property of a StopWatch object, timeGetTime is called again and the initialization time is subtracted from this new value, thus returning the number of milliseconds since the first time was stored.

Query the Elapsed property whenever you need to know how long it's been since the StopWatch was initialized. The StopWatch object is extremely useful for benchmarking your code. (See **Editorial: No More Lame Benchmarks!** for one technique.)

Data Type
Long

Reset Method

See Also   Examples   Applies to

Resets the Stopwatch object's start time.

Syntax
[*form*!]*ccrpStopWatch*.**Reset**

Remarks
The StopWatch object works by calling the **timeGetTime** API at initialization, and storing this time for future use. When you query the **Elapsed** property of a StopWatch object, timeGetTime is called again and the initialization time is subtracted from this new value, thus returning the number of milliseconds since the first time was stored.

When you call the **Reset** method, a new value is stored representing the current time, and against which future measurements provided by **Elapsed** will be taken.

Stats Property (ccrpStopWatch Object)

See Also   Examples   Applies to

Provides a class of information relevant to the current system's multimedia timer.

Syntax
[*form*!]*ccrpStopWatch*.**Stats**[= *ccrpTimerStats*]

Remarks
The **Stats** property of all ccrpTimers objects provides information regarding the current state of the system's multimedia timer and how the ccrpTimers library is using it.

Frequency
This property may be used to set or return the frequency of the multimedia timer being used by the ccrpTimers library. Internally, ccrpTimers runs off a single multimedia timer, and services all classes based on that timer. At initialization, this timer is set to fire once every 1 millisecond, to offer users the most frequent events possible. However, if you find that you need events fired at a lower frequency, you can conserve CPU usage by setting ccrpTimers to use a higher frequency.

Important Note: All objects exposed by ccrpTimers library use a single timer, thus a single frequency! Optimization of your CPU usage is achieved by selecting the Highest Common Denominator of all timer objects to use as your desired Frequency.

Resolution
This property returns the functional resolution, or accuracy, of the multimedia timers requested by ccrpTimers library. Resolution is considered the accuracy with which timer events occur. For example, if you request a timer event every 100 milliseconds, and the system's minimum resolution is 5 milliseconds, the timer events will occur within a range of 95 to 105 milliseconds apart. The ccrpTimers library requests a resolution of 1 millisecond, although the Frequency may be varied. It is important to recognize that resolution *requests* are just that -- a request for a desired setting -- and are entirely dependent on the hardware being used.

MinimumResolution and **MaximumResolution**
These properties are provided for informational purposes only. They is a measure of the minimum and maximum resolutions, or accuracy, of the multimedia timer on the hardware being tested. Resolution is considered the accuracy with which timer events occur. For example, if you request a timer event every 100 milliseconds, and the system's minimum resolution is 5 milliseconds, the timer events will occur within a range of 95 to 105 milliseconds apart. Most modern PCs are capable of true 1 millisecond minimum resolution, and generally return 1,000,000 under NT and 65,535 under Windows 95 as their maximum resolution.

Data Type
ccrpTimerStats

Tag Property (ccrpStopWatch Object)

See Also Examples Applies to

Stores any extra data needed by your program.

Syntax
[*form*!]*ccrpStopWatch*.**Tag**[= *value As Variant*]

Remarks
A place for you to store any information required to uniquely identify, or related to, each instance of this object. The setting of this property in no way affects the behavior of the object.

Data Type
Variant

Default Value
Empty

ccrpCountdown Object

Properties Methods Events

Object Name:	ccrpCountdown
Description:	Object which notifies you after a specified period of time as elapsed.
File Name:	ccrpTmr.dll
Help File:	ccrpTmr.hlp
GUID:	{d1866ac3-bec8-11d1-bbac-0055003b26de}
Properties:	8
Events:	2
Methods:	1

Before you can use **ccrpCountdown** object in your application, you must add the **ccrpTmr.dll** file to your project. If you use the object in most of your VB projects, you may want to add it to VB's Autoload file.

To distribute applications you create with the **ccrpCountdown** object, you must install and register it on the user's computer. The Setup Wizard provided with Visual Basic provides tools to help you do that. Please refer to the Visual Basic manual for details. A dependency file (ccrpTmr.DEP) has been included for your convenience.

General Usage
When declaring a new instance of **ccrpCountdown**, use the special syntax first made available with VB5 to do so with full event support. Place a declaration similar to the following in the Declarations section of either a form or class module:

```
Private WithEvents Countdown1 As ccrpTimer
```

At this point, you'll note that **Timer1** appears in the objects dropdown within that module's code window. By selecting Timer1 from this list, a new **Timer** event will be placed into your code:

```
Private Sub Countdown1_Timer()

End Sub
```

You will also be able to add a **Tick** event by dropping the events list in the code window:

```
Private Sub Countdown1_Tick(ByVal TimeRemaining As Long)

End Sub
```

Typically, you'll want to initialize the ccrpCountdown object in either the Form_Load or Class_Initialize event:

```
Set Countdown1 = New ccrpCountdown
```

At any point after initializing the object, you may start setting its various property values. Use the **Duration** property to set how long the countdown should last before the **Timer** event occurs, and the **Interval** property to set how often **Tick** events should occur *during* the countdown. As with VB's Timer control, **Enabled** turns the ccrpTimer object on and off.

As with the other objects in the ccrpTimers library, properties are exposed via the **Stats** object to reveal the **MinimumResolution** and **MaximumResolution** settings supported on the current hardware. Additionally, a **Resolution** property is exposed to inform you what resolution the library is currently using. You may also set or read the **Frequency** of the timer being used by ccrpTimers through **Stats**.

In general, your application should not set a frequency higher than absolutely necessary, to avoid forcing the CPU to process extraneous hardware interrupts. See **About Multimedia Timers** for more details.

About Method (ccrpCountdown Object)

See Also   Examples   Applies to

Shows About box for ccrpTimers

Syntax

[*form*!]*ccrpCountdown*.**About**

Remarks

This method is provided solely to provide you, the user, with a method of determining the origin of this DLL.

Duration Property

See Also Examples Applies to

Returns/sets a value that determines how long the countdown will last. Measured in milliseconds.

Syntax

[*form*!]*ccrpCountdown*.**Duration**[= *value As Long*]

Remarks

Use the **Duration** property to set how long you want the countdown to last. The **Timer** event will fire when this duration has transpired.

The time remaining in a countdown may be determined at any time by querying the **TimeRemaining** property.

The **Interval** property of the Countdown object determines how often **Tick** events will fire *during* the countdown.

Set the **Enabled** property to **True** to begin a countdown, or to **False** to interrupt a countdown.

Data Type

Long

Default Value

1000 (1 second)

Enabled Property (ccrpCountdown Object)

See Also   Examples   Applies to

Returns/sets a value that determines whether the countdown should commence or continue.

Syntax

[*form*!]*ccrpCountdown*.**Enabled**[= {**True | False**}]

Remarks

Controls whether or not a countdown is underway.

Disabling a Countdown object by setting **Enabled** to **False** cancels the countdown set up by the control's **Duration** property. **Tick** and **Timer** events only occur when a Countdown object's Enabled property is set to True.

Settings	**Description**
True	Allows object to trigger to events.
False	(Default) Prevents object from triggering events.

The **Enabled** property is automatically reset to **False** when a countdown finishes.

To temporarily delay a countdown, query and store the **TimeRemaining** property and set **Enabled** to **False**. Later, set the stored value into the **Duration** property, and set **Enabled** to **True**.

Data Type

Boolean

Default Value

False

Interval Property (ccrpCountdown Object)

See Also   Examples   Applies to

Returns/sets a value that determines how often a client is notified of countdown progress. Measured in milliseconds.

Syntax
[*form*!]*ccrpCountdown*.**Interval**[= *value As Long*]

Remarks
The **Interval** property of the Countdown object determines how often **Tick** events will fire *during* the countdown.

Use the **Duration** property to set how long you want the countdown to last. The **Timer** event will fire when this duration has transpired.

The time remaining in a countdown may be determined at any time by querying the **TimeRemaining** property.

Set the **Enabled** property to **True** to begin a countdown, or to **False** to interrupt a countdown.

Data Type
Long

Default Value
100   (1/10 second)

Notify Property (ccrpCountdown Object)

See Also   Examples   Applies to

Secondary interface provided for notification via method calls rather than events.

Syntax
[*form*!]*ccrpCountdown*.**Notify**[= *ICcrpCountdownNotify*]

Remarks
The **ccrpCountdown** object will call methods within an object (form or class) that implements the *ICcrpCountdownNotify* interface, and registers itself using this property. There are several advantages to choosing this approach, as opposed to the more classic WithEvents declaration. Principally, interfaces are early-bound, and hence much faster in execution. Secondly, events may be blocked (within the IDE) by such things as message boxes or modal dialogs. Therefore, debugging with Implements is much easier than debugging with WithEvents.

Data Type
ICcrpCountdownNotify

Default Value
Nothing

NotifyEx Property (ccrpCountdown Object)

See Also Examples Applies to

Secondary interface provided for notification via method calls rather than events. Supports identification of originating object.

Syntax
[*form*!]*ccrpCountdown*.**NotifyEx**[= *object*]

Remarks
The **ccrpCountdown** object will call methods within an object (form or class) that implements the ICcrpCountdownNotifyEx interface, and registers itself using this property. There are several advantages to choosing this approach, as opposed to the more classic WithEvents declaration. Principally, interfaces are early-bound, and hence much faster in execution. Secondly, events may be blocked (within the IDE) by such things as message boxes or modal dialogs. Therefore, debugging with Implements is much easier than debugging with WithEvents.

This property has been declared As Object so that it can accomodate a variety of interfaces. The intention is that all supported interfaces may be passed here. Presently, the **ICcrpCountdownNotify** and **ICcrpCountdownNotifyEx** interfaces are supported. ICcrpCountdownNotifyEx is identical to ICcrpCountdownNotify with one very important exception. With each method, a reference to the ccrpCountdown object firing that method is passed (see **Secondary Interfaces** for details). This allows simulation of control arrays, as well as the ability to sink "events" from multiple timer objects via a single interface without the need to resort to use of WithEvents. See **Demo: ObjArray.vbp** for complete details on implementation.

Valid Interfaces
ICcrpCountdownNotify, ICcrpCountdownNotifyEx

Data Type
Object

Default Value
Nothing

Stats Property (ccrpCountdown Object)

See Also Examples Applies to

Provides a class of information relevant to the current system's multimedia timer.

Syntax
[*form*!]*ccrpCountdown*.**Stats**[= *ccrpTimerStats*]

Remarks
The **Stats** property of all ccrpTimers objects provides information regarding the current state of the system's multimedia timer and how the ccrpTimers library is using it.

Frequency
This property may be used to set or return the frequency of the multimedia timer being used by the ccrpTimers library. Internally, ccrpTimers runs off a single multimedia timer, and services all classes based on that timer. At initialization, this timer is set to fire once every 1 millisecond, to offer users the most frequent events possible. However, if you find that you need events fired at a lower frequency, you can conserve CPU usage by setting ccrpTimers to use a higher frequency.

Important Note: All objects exposed by ccrpTimers library use a single timer, thus a single frequency! Optimization of your CPU usage is achieved by selecting the Highest Common Denominator of all timer objects to use as your desired Frequency.

Resolution
This property returns the functional resolution, or accuracy, of the multimedia timers requested by ccrpTimers library. Resolution is considered the accuracy with which timer events occur. For example, if you request a timer event every 100 milliseconds, and the system's minimum resolution is 5 milliseconds, the timer events will occur within a range of 95 to 105 milliseconds apart. The ccrpTimers library requests a resolution of 1 millisecond, although the Frequency may be varied. It is important to recognize that resolution *requests* are just that -- a request for a desired setting -- and are entirely dependent on the hardware being used.

MinimumResolution and **MaximumResolution**
These properties are provided for informational purposes only. They is a measure of the minimum and maximum resolutions, or accuracy, of the multimedia timer on the hardware being tested. Resolution is considered the accuracy with which timer events occur. For example, if you request a timer event every 100 milliseconds, and the system's minimum resolution is 5 milliseconds, the timer events will occur within a range of 95 to 105 milliseconds apart. Most modern PCs are capable of true 1 millisecond minimum resolution, and generally return 1,000,000 under NT and 65,535 under Windows 95 as their maximum resolution.

Data Type
ccrpTimerStats

Tag Property (ccrpCountdown Object)

See Also Examples Applies to

Stores any extra data needed by your program.

Syntax
[*form*!]*ccrpCountdown*.**Tag**[= *value As Variant*]

Remarks
A place for you to store any information required to uniquely identify, or related to, each instance of this object. The setting of this property in no way affects the behavior of the object.

Data Type
Variant

Default Value
Empty

Tick Event

See Also   Examples   Applies to

Occurs through the countdown, at [Interval] milliseconds, to indicate progress toward completion. Remaining number of milliseconds is passed to event.

Syntax

Private Sub *ccrpCountdown*_**Tick**(ByVal *TimeRemaining* As Long)

Remarks

Tick events occur every [**Interval**] milliseconds during a countdown. The **TimeRemaining** in the countdown, expressed in milliseconds, is passed as the only parameter to this event.

Use the **Duration** property to set how long you want the countdown to last. The **Timer** event will fire when this duration has transpired.

The time remaining in a countdown may be determined at any time by querying the **TimeRemaining** property.

Set the **Enabled** property to **True** to begin a countdown, or to **False** to interrupt a countdown.

Timer Event (ccrpCountdown Object)

See Also   Examples   Applies to

Occurs when the countdown has competed.

Syntax

Private Sub *ccrpCountdown*_**Timer**()

Remarks

Timer events occur after [**Duration**] milliseconds following a countdown. The **Timer** event will fire immediately after the final **Tick** event. Use the **Duration** property to set how long you want the countdown to last. The **Timer** event will fire when this duration has transpired.

The **Interval** property of the Countdown object determines how often **Tick** events will fire *during* the countdown.

The time remaining in a countdown may be determined at any time by querying the **TimeRemaining** property.

Set the **Enabled** property to **True** to begin a countdown, or to **False** to interrupt a countdown.

TimeRemaining Property

See Also   Examples   Applies to

Returns a value that indicates the number of milliseconds remaining in the countdown.

Syntax
value As Long = [*form*!]*ccrpCountdown*.**TimeRemaining**

Remarks
The **TimeRemaining** property returns the number of milliseconds, if any, remaining in an active countdown. This property may be queried at any time from any procedure. The value of this property is automatically passed to **Tick** events.

Data Type
Long

Default Value
0

TimerEventTypes Constants

Used with
EventType (ccrpTimer)

Value	Constant
0	TimerOneShot
1	TimerPeriodic