# Logbook Control (VBX **version 1.0**, OCX **version 1.1**)

**Properties** **Events** *Help file revision 1.1.1*

## Overview

Thank you for evaluating or using the Logbook control. This software component implements a scrolling message console with very flexible file logging and time stamping capabilities.

Check out what is new in **this release**.

The most relevant features of this control are:

- It comes as an ActiveX control (OCX) in both 16-bit and 32-bit versions. An earlier version (1.0) is still available as a VBX for use in Visual Basic 3.
- Clients write messages to the object, which time stamps and formats them. The messages are then shown in a configurable scrolling window and optionally logged to a file.
- Messages can have a "severity" attribute (e.g. 'debug', 'info', 'alert'). The object can filter messages whose severity is below a certain level.
- The control can track time in several ways, such as: 'current time', 'time since log started' and 'time since last message'. A user-defined time stamp format can also be specified.
- A path (directory) for the log file location and an explicit file name can be specified. Optionally, the object will automatically name the files by encoding the time of creation.
- Logging can be started, stopped, paused and resumed. Event procedures are available for log-a-message and status change (i.e. start/stop/pause/error) events.
- The control can open the log file automatically and close it after a time-out has occurred or a certain number of messages has been logged.
- The control keeps statistics of the number of messages that were logged, ignored and lost (i.e. sent when logging was off or paused). These counters can be dumped to the log itself, or queried as separate read-only properties.
- Finally, the control supports DDE conversations. It can receive messages to be logged via DDE and/or echo the logged messages to DDE clients.

## Introduction

**Information** **Installation**

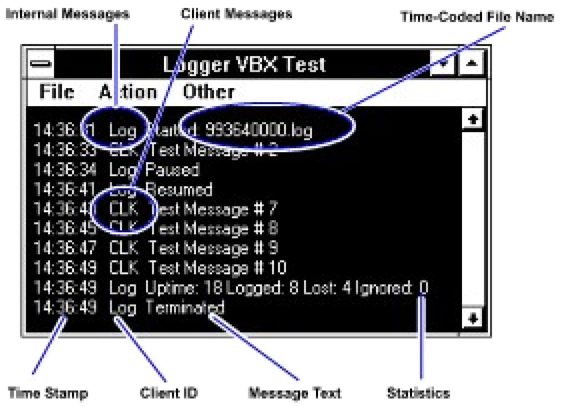
## Programmers's guide

**Tutorial** **Properties**
**Methods** **Events**

## Tutorial

The following figure shows a simple Visual Basic form filled with a single Logbook instance.

The control appears as a scrolling list box. Messages are formatted and time-stamped before being sent to the console and file. Here the logbook is using a disk file, and the file name has been automatically generated by encoding the current date and time.



```
14:36:[illegible]1  Log  Started: 993640000.log
14:36:33  CLK  Test Message # 2
14:36:34  Log  Paused
14:36:41  Log  Resumed
14:36:4[illegible]  CLK  Test Message # 7
14:36:45  CLK  Test Message # 8
14:36:47  CLK  Test Message # 9
14:36:49  CLK  Test Message # 10
14:36:49  Log  Uptime: 18 Logged: 8 Lost: 4 Ignored: 0
14:36:49  Log  Terminated
```

The message source can be identified with a ClientID tag. This tag can be provided together with the text message or is taken from the **ClientID** property.

In the example shown above, a single "Clock" Timer object is sending the messages. Its messages are identified by the "CLK" ClientID tag and are shown as "Client Messages" in the figure

Here housekeeping messages generated internally by the control are marked with the "Log" ID tag and are shown as "Internal Messages" in the figure. The ID for internal messages is taken from the object Caption property.

To learn how to build a simple program like the one shown above, select one of the following topics:

**Design**
**Operation**
**Notes**

## Design

### Using the OCX version of the control

If you are using a Visual Basic 4 or better, you can use the OCX version of the control. You should include the appropriate version of the control (LOGBK16.OCX for Visual Basic 4/16 bit or LOGBK32.OCX for 32-bit versions of Visual Basic) in your project. To pick the useful constants for the control you can then either use the provided support file LOGBOOK.BAS or get them directly from the OCX using the Visual Basic object browser (hitting the F2 key).

For simplicity, in this help file we will use the constants as they are defined in the LOGBOOK.BAS file (see below). The equivalent new-style constants should be easy to find and pick from the object browser.

### Using the VBX version of the control

If you are using a 16-bit version of Visual Basic, such as version 3 or version 4/16, you can use the VBX version of the control. Be aware that the VBX version, using a now obsolete object architecture, is missing some functionality with respect to the OCX. Such differences are noted in the help file wherever appropriate.

In order to use the control, you should first of all include the files LOGBOOK.VBX and LOGBOOK.BAS in your project. The .BAS file will enable you to use the predefined constants that appear in this document. You can then add one or more Logbook control instances to your forms.

### Basic settings

If only one client is using the Logbook (as is the case with the example shown in the first **Tutorial** page) the messages attributes **ClientID** and **ClientLevel** can be set to default values from Visual Basic *Design-Mode* editor. These will determine the default identifier for the message source and the default severity level for messages.

Note that these attributes are quietly overridden when the Logbook generates messages internally and can also be overridden on a message-by-message basis by clients. They will be used whenever the source ID or the severity level are left unspecified in subsequent logging commands.

The following table summarises the control properties that have been set at design time for the tutorial example (some actually are the default values):

| Name | Value | Comment |
|---|---|---|
| **Append** | True | Append to existing file (if any) |
| Name | "Logbook1" | Automatically set at creation |
| Caption | "Logbook1" | As above, used as ID |
| **ClientID** | "CLK" | Used as default when logging |
| **ClientLevel** | 2 - Information | As above |
| **Epoch** | 94 | Year '0' (1st digit in file name) is '94 |
| **FileName** | "" | Generate time-coded file names |
| **FilePath** | "C:\TMP" | Place where files are stored |
| **FilterLevel** | 1 - Diagnostic | Discards only 0-level messages |
| **LogToFile** | True | Enables writing to a disk file |
| **TimeFormat** | "HH:MM:SS" | Used for making time-stamps |
| **TimeStyle** | 1 - Time of day | Source time for time-stamps |

***Note***: If the sample messages in the control appear with odd times, you probably need to set the **TimeZone** environment variable.

Next Page: **Operation**

## Operation

### Controlling the Log

Logging is normally started and stopped with a program statement, calling one of the appropriate **Methods**, such as *Start* or *Stop*.

A very simple program may choose to start and stop the log from the Form_Load and Form_Unload event SUBs, as in the following example.

```
' the following will only work for OCX versions of the control,
' VBXs can only use the "Action" property
Sub Form_Load ( )
    Logbook1.Start
End Sub
```

An even simplier program can set the **AutoStart** property to *True* and let the control initialize and open the log file automatically when the first message is logged.

With VBX versions of the control, you cannot use the above methods, but rather assign a value to the **Action** property. The example Visual Basic form shown in the first **Tutorial** page has an "Action" menu, where   the single menu entries trigger the following Visual Basic code:

```
' the following will work for both VBX and OCX versions of the control
Sub mAction_Click (Index As Integer)
    Logbook1.Action = Index
End Sub
```

This means that menu item #0 will *start* the log (open the file and enable message logging to the console and file), while menu item #1 will *stop* the log (dump statistics, close the file and disable further logging), and so on for the other indices.

### Logging Messages

There are several ways of logging a message:

1 - Use the .Log method (OCX only). Optionally, client ID and severity can be specified, overriding the default values.

```
' Note: only the first parameter is needed, others are optional
Logbook1.Log "Test Message", MyId$, SeverityLevel%
```

2 - Assign a value to the **LogMessage** property, using the default values for client ID and severity level:

```
 Logbook1.LogMessage = "Test Message"
```

3 - Use the standard Visual Basic AddItem method, optionally composing a string with the client id, separated from the message with a TAB character.

```
Logbook1.AddItem "CLK" & Chr$(9) & "Test Message", SeverityLevel
```

Additionally, since **LogMessage** is the control default property, you could even write statement #2 as:

```
Logbook1 = "Test Message"
```

Note that, in case #3, the control interprets the optional numeric parameter of the AddItem method as the "severity level" of the message being logged. This value overrides the current setting of the **ClientLevel** property without permanently changing its value. As shown in the sample code, the **ClientID** can also be overridden by prepending a string to the message text and inserting a TAB character (ASCII 9) in between.

In case #2, the message is simply logged using the current setting for the **ClientLevel** and **ClientID** properties. The same happens in case #1 if you omit to specify the last two parameters.

Next Page: **Notes**

## Notes

### Using the Logbook with DDE

The Logbook control can receive messages through DDE while acting as either a *client* or *server*. These situations are depicted in the two diagrams below.

When the Logbook is running as a *server* (diagram 1), multiple clients can send messages at the same time *poking* the link (i.e. a Visual Basic program can invoke a LinkPoke method on a client control). To prevent the messages from being echoed back to the clients, these links should be set to MANUAL.

Note that, at the same time, there may be one or more other DDE clients that do not send messages to the logbook, but only act as "receivers" (i.e. Client #3 in diagram 1). These will usually have their link set to AUTOMATIC and will receive copies of the formatted logged messages.

When the Logbook is running as a *client* (diagram 2), only 1 source can send messages to it through the link, which may be set to AUTOMATIC, MANUAL or NOTIFY as appropriate.

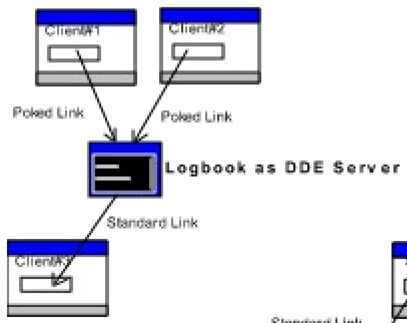


*Diagram 1*

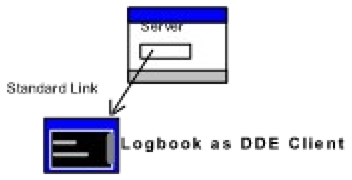


*Diagram 2*

### Note:

When sending messages *to* a logbook control via DDE, there is no way to provide the **ClientLevel** information separately from the message. For this reason, messages sent through DDE can only override the current **ClientLevel** setting by embedding the level as a 1-character digit at the front of the string. The **ClientID** string can be included between the level digit and the message body, separated from the rest by a TAB (ASCII 9) character.

Example:

```
SourceControl.Text =  "3"+"CLK" & Chr$(9) & "Test Message"
SourceControl.LinkPoke
```

When receiving messages *from* a logbook, the whole, formatted message is sent.

## Version History

### OCX Version 1.1

OCX-only (now ActiveX) version of the control, available in 16 and 32-bit versions, with several additional features:

- New **Methods** (Start, Stop, Pause, Resume) can be used instead of the old-style **Action** property.
- There is a new preferred method: **Log**, for logging messages to the control, as opposed to the old **LogMessage** property and **AddItem** method.
- The control now has convenient property pages for all relevant options.
- It is possible to show the property page dialog at run time using the **Properties** method.
- It can show the severity level of a message as 1st field in the log file (off by default for compatibiltiy with old log file format).
- There are 4 flag properties for showing or turning off the different fields of a message: **ShowLevel**, **ShowTime**, **ShowID**, **ShowMessage**
- It is now possible to specify a **Separator** string which is inserted between the log fields. By default, a single space is used, but this can be set to a TAB, a comma, or any other string for subsequent easy manipulation of the log file.
- There is now a **LogToConsole** property, for simmertry with **LogToFile**. Note: changing LogToFile is only effective before Log is opened.
- The **FilePath** property can now explicitely contain one DOS-style environment variable, like %TEMP%, that is interpreted at run-time. The full log file path used while logging can be retrieved from the **FileFullPath** property.
- An **AutoStart** property has been defined that, when True, makes the control start a log file on the first received message. You no longer have to explicitely start the log.
- The 3-char limit in the **ClientID** has been eliminated. Ids can now be as long as you like. The default client ID is now "???".
- When the control is used in a different host from Visual Basic, the Format$() syntax for **TimeFormat** is not longer available. In this case the control falls back to a C-language style syntax for the time stamps.
- The control can now be shown with either a "flat" or a 3D look, according to the setting of the **Appearance** property.

### OCX Version 1.0

A preliminary port of the custom control as a 16-bit OLE control (OCX), with equivalent functionality to the VBX version.

### VBX Version 1.0

Initial release, VBX only.

## Properties

All of the non-standard properties for this control are listed in the following table. For information on the remaining properties, see the standard Visual Basic documentation.

**Properties:**

| | | |
|---|---|---|
| **About** | **hFile** | **ShowLevel** |
| **Action** | **LogMessage** | **ShowMessage** |
| **Append** | **LogToConsole** | **ShowTime** |
| **AutoStart** | **LogToFile** | **StopMsgLimit** |
| **ClientID** | **MaxConsoleMsg** | **StopTimeOut** |
| **ClientLevel** | **MsgIgnored** | **Status** |
| **Epoch** | **MsgLogged** | **TimeFormat** |
| **FileFullPath** | **MsgNotLogged** | **TimeStyle** |
| **FileName** | **Separator** | **TimeZone** |
| **FilePath** | **ShowID** | **UpTime** |
| **FilterLevel** | | |

## About

Dummy property, used to trigger a dialog box with some information about the VBX (version and copyright) from a visual programming environment (e.g. VBasic in design mode).

| | |
|---|---|
| **Notes:** | Not available at run-time. |

## Action

This property is actually used to implement custom methods that "trigger" some action in the object. This is the only way such commands can be given to the VBX version of the control. With OCX versions of the control, it is possible to use the corresponding **Methods**.

The following actions are available:

| | | |
|---|---|---|
| LOG_ACTION_START | = 0 | Start logging and tracking time, possibly open log file. |
| LOG_ACTION_STOP | = 1 | Stop logging and tracking time, possibly close log file. This action implicitly perform a LOG_ACTION_STATUS before logging is stopped. |
| LOG_ACTION_PAUSE | = 2 | Pause logging (messages will not be displayed or written to the file), but keep stats and possibly leave the log file open. |
| LOG_ACTION_RESUME | = 3 | Resume a paused logging. |
| LOG_ACTION_RESET | = 4 | Reset internal counters, turn off auto-deactivation (zeroes time-out and message limit), resets the filtering threshold to LOG_LEVEL_INFO. It does *not* clear the console contents, and does *not* alter the logging status. |
| LOG_ACTION_STATUS | = 5 | Log a dump of the current Logbook statistics. The message has severity = LOG_LEVEL_DIAGN (= 1), and will be filtered out with the default threshold settings. This allows you to enable or disable the status output by simply changing the filtering level, rather than modifying the code. |
| LOG_ACTION_PROPS | = 6 | (OCX only) Show the *modal* Properties dialog box. The application is suspended while the dialog is shown. |

If , as a consequence of this action a change of the **Status** property occurs, a **Change** event will be fired, and an appropriate message will be logged, with severity = LOG_LEVEL_INFO. These messages can be filtered out by rising the current severity threshold.

Naturally, some actions require that the logbook object be in a certain status to be effective. E.g.: you can resume a paused log, but not a stopped one. In the current version of the control, applying an action when the object is in the wrong state does *not* cause an error: the action is silently ignored.

| | |
|---|---|
| **Type:** | Integer |
| **Notes:** | Available only at run-time, write-only. |

## Append

When this option is True, if the file that is going to be opened for logging already exists on disk, the object shall append messages to the existing file rather than overwrite it. This option applies to both user-specified file names and to automatically generated, time-coded file names.

| | |
|---|---|
| **Type:** | Integer |
| **Default Value:** | True |

## ClientID

When logging messages using the **LogMessage** property, it is possible to indicate the default "sender ID" of the client by setting this property. The ID remains in effect for all subsequent messages that do not explicitely specify a different ID.

When using other techniques for logging messages that explicitely allow you to specify a client ID, such as the Log **Method**, the setting of ClientID will silently be overridden for that specific message (i.e. the message will get the ID specified in the Log method, but this property will not be set to the new value).

| | |
|---|---|
| **Type:** | String |
| **Default Value:** | "???" |

## ClientLevel

When logging messages using the **LogMessage** property, it is possible to indicate the severity level at which the client is sending messages by setting this property. The level remains in effect for all subsequent messages that do not explicitely specify a different severity level.

For a list of the available levels, see the **FilterLevel** property.

| | |
|---|---|
| **Type:** | Integer |
| **Default Value:** | LOG_LEVEL_INFO (=2) |

## ShowLevel, ShowTime, ShowID, ShowMessage

These properties indicate which fields of the message should be displayed on the console and written to the log file. When all are *True*, the message is formatted as follows:

```
<level> <time stamp> <client id> <message text>
```

Example:

```
2 10:22:37 CLK A sample message
```

| | |
|---|---|
| **Type:** | Integer |
| **Default Value:** | False, True, True, True |

## Epoch

The year that is taken as '0' when building a time-coded **FileName**. For example, when Epoch=95, a log-file named "20122117.LOG" indicates it was generated on Jan 12, *1997*, at 21:17.

| | |
|---|---|
| **Type:** | Integer |
| **Default Value:** | 94 |

## FilePath

Directory where log files (if **LogToFile** =True) are created. This property does *not* contain the actual file name, which can be controlled with the separate **FileName** property.

It is possible, in the OCX version of the control, to include in the path specification one environment variable, using the standard DOS syntax (%*var_name*%). In this way, it is possible to specify a system-dependent path in a portable way. In effect, the default value for this property specifies the log files to be placed in the temporary DOS/Windows system directory.

Note: the full log file path used while logging can be retrieved from the **FileFullPath** property.

| | |
|---|---|
| **Type:** | String |
| **Default Value:** | "%TEMP%" |

## FileFullPath

This read-only propery will return the full log-file path name currently used by the control while writing to a disk log. It is made up by the catenation of the **FilePath** property (resolving any embedded environment variables) and the **FileName** property (resolving any dynamic generation of time-encoded file names).

| | |
|---|---|
| **Type:** | String |
| **Notes:** | Available only at run-time, read-only. |

## FileName

Name of the log file. If different from the empty string (""), the object will use it literally, appending it to the **FilePath** property. When set to an empty string, the logbook will use a default file naming scheme, where the name is a time-code string formatted as follows:

YDDDHHMM.LOG

Where:

| | |
|---|---|
| Y | 1-digit year (starting with 0 = **Epoch**) |
| DDD | day-of year (1, 2....) |
| HH | hour (0..23) |
| MM | minute (0..59) |

Note: the full log file path used while logging can be retrieved from the **FileFullPath** property.

| | |
|---|---|
| **Type:** | String |
| **Default Value:** | "" (forces time-coded naming scheme) |

## hFile

MS-Windows operating system file handle used by the Logbook control. This 16-bit value can be passed to MS-Windows system calls in order to write to the file directly, without going through the control methods. It will be valid (non-null) only when the log file is open (i.e. **Status** = LOG_OPEN o LOG_PAUSED and **LogToFile** = True).

| | |
|---|---|
| **Type:** | Integer |
| **Default Value:** | 0 (closed file) |
| **Notes:** | Available only at run-time, read-only. |

## FilterLevel

Severity threshold. Messages tagged with a severity level below the current **FilterLevel** will be ignored. The available levels are:

| | | |
|---|---|---|
| LOG_LEVEL_DEBUG | = 0 | A debugging message |
| LOG_LEVEL_DIAGN | = 1 | A diagnostics message, still not an error |
| LOG_LEVEL_INFO | = 2 | Normal information or notification of events |
| LOG_LEVEL_WARN | = 3 | Notification of unusual events, may require attention |
| LOG_LEVEL_ERROR | = 4 | Definitely an error condition, but may still be harmless |
| LOG_LEVEL_ALERT | = 5 | Serious error, definitely requires immediate intervention |
| LOG_LEVEL_CRITICAL | = 6 | Critical error or damage, system may be blowing, bail out. |

| | |
|---|---|
| **Type:** | Integer |
| **Default Value:** | LOG_LEVEL_INFO (=2) |

## LogMessage

This property can be assigned to the actual message to be logged.

If the current status of the object is LOG_ON, this triggers a **Log** event and eventually causes the message to be logged to console and (optionally) to a disk file. This is exactly the same thing that happens when calling the **AddItem** or the **Log** methods.

The only difference is that, when using this property, the message level and the client ID are taken from the **ClientLevel** and **ClientID** properties, while **AddItem** and **Log** allow to specify one or both these parameters in a single action, overriding the default properties.

It is also possible to *read* this property. It will return the last message that was written to the log (by either the control itself or by a client). The returned string will hold the complete, formatted message, including time-stamp and ClientID tag. If no message has been logged yet, an empty string ("") will be returned.

| | |
|---|---|
| **Type:** | String |
| **Notes:** | Unavailable at design time. This is the default control property, and is used as the control value. |

**Example:**

```
' Set client id, client message severity, then log a message.
Sub Button1_Click  ( )
     Logbook1.ClientID = "XYZ"
     Logbook1.ClientLevel = LOG_LEVEL_ERROR
     Logbook1.LogMessage = "A button-triggered error"

     ' Test sending another message using the control default property feature
     Logbook1 = "A message logged by assigning to the default property"
End Sub
```

## LogToFile

Actually write messages to file or not. If this property is False when the log is Started, no file is opened. Subsequent messages will only be sent to the console window, according to the setting of the corresponding **LogToConsole**.

Note that leaving a blank string as the log **FileName** will *not* disable writing to a disk file, but rather causes the object to use an incremental time-code naming scheme. Hence this separate flag to turn disk file usage on or off.

| | |
|---|---|
| **Type:** | Integer |
| **Default Value:** | False |

## LogToConsole

Write messages to the console or not. This is independent from logging to a file, which is controlled by the corresponding **LogToFile** property.

| | |
|---|---|
| **Type:** | Integer |
| **Default Value:** | False |

## MsgLogged

Number of messages logged to window (and optionally to file).

This value is reset to 0 when a new log is started or when the log object is explicitly reset with *LogbookObj*.**Action**=LOG_ACTION_RESET or when a **Clear** method is invoked.

| | |
|---|---|
| **Type:** | Integer |
| **Default Value:** | 0 |
| **Notes:** | Available only at run-time, read only. |

## MsgNotLogged

Number of messages not logged because the object was in LOG_PAUSED or LOG_OFF **Status**, but that were *above* the current **FilterLevel** threshold (that is, they would have been logged under normal conditions).

This value is reset to 0 when a new log is started or when the log object is explicitly reset with *LogbookObj*.**Action**=LOG_ACTION_RESET or when a **Clear** method is invoked.

| | |
|---|---|
| **Type:** | Integer |
| **Default Value:** | 0 |
| **Notes:** | Available only at run-time, read only. |

## MsgIgnored

Number of messages ignored because their severity was lower than the current severity threshold defined by the **FilterLevel** property.

The count of ignored messages proceeds regardless of the current logging status. This means the logbook will count ignored messages even if the log file is currently closed.

This value is reset to 0 when a new log is started or when the log object is explicitly reset with *LogbookObj*.**Action**=LOG_ACTION_RESET or when a **Clear** method is invoked.

| | |
|---|---|
| **Type:** | Integer |
| **Default Value:** | 0 |
| **Notes:** | Available only at run-time, read only. |

## MaxConsoleMsg

Backlog of messages to show on the scrolling console window. Excess messages shall be removed from the console history, and will be lost, unless the object is also logging them to a file.

| | |
|---|---|
| **Type:** | Integer |
| **Default Value:** | 100 |
| **Notes:** | If the user is holding the scrollbar with the mouse, removal of excess messages is deferred until the mouse button is released, so this limit can be temporarily exceeded. |

## Status

The current status of the object, as affected by an **Action** performed on it or some other external event. The possible states are:

| | | |
|---|---|---|
| LOG_OFF | = 0 | Not logging (file closed, not tracking time). Messages sent when in this status that would have been logged are "lost" (the object just counts them in **MsgNotLogged**). |
| LOG_PAUSED | = 1 | Has an active log (tracks time and maybe has an open file , but is currently not logging the messages sent to it). Messages sent when in this status are also "lost" (the object just counts them in **MsgNotLogged**). |
| LOG_ON | = 2 | Running and logging to console and possibly to file. |
| LOG_ERROR | = 3 | Something failed (e.g. failed to open the log-file, etc.). The object behaves like in the OFF status. |

On a change of the **Status** property, a **Change** event will be fired. Note that the VBX does not generate run-time errors internally. So, to catch events like a log-file open failure, you have to use the **Change** event and check for a LOG_ERROR status.

| | |
|---|---|
| **Type:** | Integer |
| **Default Value:** | LOG_OFF (=0) |
| **Notes:** | Available only at run-time, read only. |

## StopMsgLimit

When this property is set to a value > 0, the logbook will automatically execute a LOG_ACTION_STOP after the specified number of messages have been logged.

When the log is closed, the **Change** event procedure is fired, with Reason = LOG_RES_MSG_LIMIT (=1).

| | |
|---|---|
| **Type:** | Integer |
| **Default Value:** | 0 (off) |

**Example:**

You can use this feature is to limit the size of individual log-files to some manageable extent. In this case, you should leave the **FileName** property blank (to get unique, automatically generated file names) and re-start the file from within the **Change** event procedure, as shown below:

```
' Start a new log file after it auto-closed at a certain size
Sub Logbook1_Change (Reason As Integer)
     If Logbook1.Status = LOG_OFF And Reason = LOG_REAS_MSG_LIMIT Then
       Logbook1.Action = LOG_ACTION_START
     End If
End Sub
```

## StopTimeOut

Time-Out (in seconds) after which the object automatically performs a LOG_ACTION_STOP. Can be used to record messages in the log for a given period of time. When the log is closed, the **Change** event procedure is fired, with Reason = LOG_RES_TIME_OUT (=2).

The count starts from the moment the log was started (if the property is assigned at design time or before starting the log), or, if the log is already open, every time that the property is assigned.

For example, setting the property to 10 with the log open will stop the log 10 seconds after the assignment.

Setting this property to 0 disables the time-out feature. This action will also turn off a time-out count that was already in progress.

Stopping and restarting the log manually will also restart the count.

| | |
|---|---|
| **Type:** | Integer |
| **Default Value:** | 0 (off) |

## TimeStyle

Time-tracking method . This property defines the *meaning* of the time stamp, i.e. *what* value is formatted into the message, rather than *how* it is formatted.

Possible values:

| | | |
|---|---|---|
| LOG_TIME_NONE | = 0 | No time-tracking. No time-stamp will be added to the messages, regardless of the selected format. |
| LOG_TIME_OF_DAY | = 1 | Uses current date and time. |
| LOG_TIME_SINCE_START | = 2 | Uses time since the log was started. |
| LOG_TIME_DIFFERENCE | = 3 | Time since last message. |

Styles 2 and 3 are accurate within a year. When these time-tracking methods are selected, using month name, day-of-week and other such components in the time-stamp format string, will give unpredictable and weird results.

| | |
|---|---|
| **Type:** | Integer |
| **Default Value:** | LOG_TIME_OF_DAY (= 1) |

## TimeFormat

This string property defines the formatting template used when composing the time stamp that is printed in front of logged messages.

When the control is used with Visual Basic, the format of this string is exactly as defined for the Visual Basic Format$() function, so it can be used to specify hours, minutes, seconds or dates in short and long format, day-of-the-week, etcetera. See the standard Visual Basic help for the Format$() string syntax specification.

When the OCX version of the control is used in different hosts from Visual Basic, the Format$() syntax will not be available. In this case the OCX will fall back to a subset of the C-style *strftime*() function syntax, as specified in the following table:

| | |
|---|---|
| %j | Day of the year as a decimal number (001-366) |
| %H | hour in 24 hour format |
| %M | minute (0-59) |
| %S | seconds (0-59) |
| %d | Day of the month as a decimal number (01-31) |
| %m | Month as a decimal number (01-12) |
| %y | Year without the century as a decimal number (00-99) |
| %w | Weekday as a decimal number (0-6; Sunday is 0) |
| %% | percent sign |
| %B | Full month name (January-December) |
| %A | Full weekday name (Sunday-Saturday) |
| %a | Abbreviated weekday name (Sun-Sat) |
| %b | Abbreviated month name (Jan-Dec) |
| %Z | Time zone name or abbreviation; no characters if time zone is unknown |
| %I | Hour in 12-hour format (01-12) |
| %p | AM/PM indicator for a 12-hour clock |
| %Y | Year with the century as a decimal number |

Note that the range of formatting options available in the two environment are not fully equivalent.

| | | | |
|---|---|---|---|
| **Type:** | String | | |
| **Default Value:** | "HH:MM:SS" | in Visual Basic e.g. | "18:20:00" |
| | "%H:%M:%S" | in other hosts | |

## UpTime

Number of seconds elapsed since the log was started. This value is zero when log is OFF. It will keep incrementing even if logging is currently paused.

This value can be formatted into hours, minutes and seconds from Visual Basic with a statement like:

```
Print Format$(TimeSerial (0,0,Logbook1.UpTime),"HH:MM:SS")
```

| | |
|---|---|
| **Type:** | Long |
| **Notes:** | Not available at design time, read-only at run-time. |

## TimeZone

The time routines inside the control require that the Time Zone environment variable (TZ) should be properly set up. This is usually done inside your *autoexec.bat,* (in Windows 3.x and Windows 95) with a statement like:

```
SET TZ=<your time zone string, e.g. MET-1MEDST>
```

In Windows NT, there is no autoexec.bat file. Environment variables can be set or checked using the System tool in the control panel.

Note that setting the time zone in Windows from the control panel is *not enough*. The run-time library for the C programming language, used in the control, insists that the world should be a UNIX system, so it will not fetch the information from Windows, but rather from an environment variable as it happens in UNIX.

Even more stubbornly, If you dont have this variable set in *autoexec.bat*, by default the library will assume that you live in California and set itself for Pacific Standard Time (PST8PDT).

Last, note that the syntax for the TZ environment variable uses a time offset which is *opposite* to the normal conventions. The reason is that, given its american origins, the variable indicates the difference *from* local time *to* Universal Coordinate Time (i.e. GMT) and not viceversa. As a consequence, a central european time zone of GMT+1 must be encoded with a -1 offset (MET-1), while the american pacific time zone (GMT-8) gets encoded as PST+8.

A wrong or missing setting of the TZ variable will cause odd times to appear when you select the time-of-day **TimeStyle**. If you live in Europe, your times will be ahead by 8 or 9 hours.

So, we suggest you fix your *autoexec.bat* file if you haven not already done so. We provide the TimeZone control property only for debugging your setting and playing around before you fix it permanently.

This property will show you the current TZ setting (so if you see PST8PDT you know it is missing). It will also let you modify the control TZ setting, but after changing it you will notice some strange effects:

1 You will affect ALL Logbook 16-bit controls in the system (because time routines use shared, global variables, including TZ, contained in the control DLL)

2 Changes will NOT be saved in the Visual Basic project. You do not want to distribute applications which force an incorrect TZ on the target system, do you? So, next time you reload the project, TZ will still be wrong if you have not set it up in *autoexec.bat*.

3 Changes are not system-wide, i.e. the environment of other, different applications also using the TZ variable will not be affected, because the 'master copy' created at system startup cannot be touched by any program, as happens for other application-specific copies created from it.

| | |
|---|---|
| **Type:** | String |
| **Notes:** | Will NOT be saved in the project like other VB properties. |

## Separator

This property specifies a string that is used to separate the fields in a message when it is logged to the console or to the file. By default, a single space is used, for compatilbility with the old log file format used by the VBX version of the control.

You can set this property to anything convenient to enable post-processing of the log file, such as TAB character, a comma and any convenient spacing. The same string is inserted between *all* of the log message fields.

| | |
|---|---|
| **Type:** | String |
| **Default Value:** | A single space (" ") |

## Methods

The following methods are only implemented in the OCX versions of the control:

| | |
|---|---|
| **Log** MsgText, *ClientID*, *ClientLevel* | Log a message. The *ClientID* and *ClientLevel* paramters are optional. When unspecified, the corresponding properties are used as default values. |
| **Pause** | Pause logging, messages sent to the control while paused will be lost. |
| **Resume** | Resume normal logging |
| **WriteStatus** | Write log statistics to the log file itself |
| **Start** | Start Logging, opening a new log file if required |
| **Stop** | Stop Logging, closing the log file if required |
| **Reset** | Reset counters and options to the default values. |
| **Properties** | Show the *modal* Property Page dialog for changing options at run-time. The application is suspended while the dialog is shown. |

A typical logging sequence using the above methods would be:

```
Sub Sample_Log ( )
    Logbook1.Start                          ' open the log file
    Logbook1.Log "message 1", "test", 1     ' log messages at various levels
    Logbook1.Log "message 2", "test", 2
    Logbook1.Log "message 3", "test", 3
    Logbook1.Pause                          ' temporarily suspend logging
    Logbook1.Log "Lost Message", "test",3
    Logbook1.Resume
    Logbook1.WriteStatus                    ' write logging statistics
    Logbook1.Log "Last Message"             ' use default ID and level
    Logbook1.Stop
End Sub
```

Unlike OCX's, VBX controls can not define arbitrary new methods. For this reasons, the VBX version of the control do not have the methods described above, but can only provide a dummy property, **Action** to implement the same functionality. The Action property is also available in the OCX version, for compatibility with older code, but it is recommended that you use the methods shown above.

However, even VBX's can *re-implement* some of the predefined methods if this makes sense. So, both VBX and OCX versions of the control can support the following methods, traditionally available in all Visual Basic controls:

**AddItem** **Clear**

## AddItem

Log a message to the object console (and possibly to the log file), optionally specifying a severity level.

**Note:** *In OCX versions of the control, this method is considered obsolete. You should use the* **Log** *method instead, which is simplier and gives you more control over logging.*

If *Message$* contains a TAB character (ASCII 9), the characters before the tab are taken as a "client ID", truncated to 3 characters and placed between the time-stamp and the message body, with proper formatting.

This method allows clients to specify all 3 parameters (sender id, message and severity) in a single action. Alternatively, it is possible to assign values to the **ClientID** and **ClientLevel** properties, then send the message by assigning it to the **LogMessage** property.

The message level and client id specified with this method override the current settings of the **ClientLevel** and **ClientID** properties, which, however, are left unaltered.

If the current status of the object is LOG_ON, a **Log** event will be fired, before the message is filtered and written to the window and file. This means that the event procedure has a chance to modify the message and/or its severity level before the message is processed for logging.

**Syntax:** Logbook.*AddItem* Message$, Severity

**Example:**

```
' Send client id,   message and severity in a single action.
Sub Button1_Click  ( )
     Logbook1.AddItem "XYZ" & Chr$(9) & "A button-triggered error",   LOG_LEVEL_ERROR
End Sub
```

## Clear

Clears the console window and reset all the internal logbook statistics. This method does not affect the current log-file (if any is used).

This command is slightly different from **Action**=LOG_ACTION_RESET. Both commands reset the internal counters, but:

- **Clear** also clears the console window, but does *not* alter the current **FilterLevel** or zap the automatic log-off features.
- LOG_ACTION_RESET does *not* clears the console , but resets the current **FilterLevel** and disables the automatic log-off features by zeroing their counters.

**Syntax:** Logbook.*Clear*

## Events

The following table lists the event that apply *only* to the Logbook control or that require special consideration when used with it. For information on the remaining events, see the standard Visual Basic documentation.

| **Events:** | |
|---|---|
| ***Change** | **Log** |

## Standard Properties

Please refer to the Visual Basic on-line help documentation regarding custom controls for information on this property, event or method.

## Log

A message was submitted for logging, while logging is active (**Status** = LOG_ON). The event is fired *before* the message is processed (i.e. filtered against the current severity threshold and written to the console and file).

This event procedure is fired only when a *client* logs a message, not when the message is written by the control itself (even if this happens in response to an **Action**):

This provides a flexible hook for implementing sophisticated message filtering and formatting. (e.g. filtering messages from a specific client). A message can be filtered out by lowering its *Level* below the current filtering threshold.

**Syntax:**

Logbook_*Log (SenderID As String, Message As String, Level As Integer)*

**Example:**

```
' Declass some messages and cause them to be filtered out.
Sub Logbook1_Log (SenderID As String, message As String, Level As Integer)
     If SenderID = "XYZ" Then
           Level = 0
           Debug.Print "Msg from XYZ will be ignored"
      End If
End Sub
```

## Change

This event is fired when some **Action** or other event triggered a change of status in the logbook object. This hook can be used to provide some visual feedback of the logbook status, or to perform some sophisticated log-file management task, such as re-opening a log file that was closed on a time-out or when a message number limit was exceeded.

The *Reason* parameter can be used to discriminate what caused the object to change its status. The possible values are:

| | | |
|---|---|---|
| LOG_REAS_ACTION | = 0 | Caused by an assignment to the **Action** property |
| LOG_REAS_MSG_LIMIT | = 1 | The obj turned itself off because the **StopMsgLimit** was reached |
| LOG_REAS_TIME_OUT | = 2 | The obj turned itself off because the **StopTimeOut** was reached |
| LOG_REAS_ERROR | = 3 | Some error occurred (like a file I/O error) |

**Syntax:** Logbook_*Change (Reason As Integer)*

**Example:**

```
' Show the reasons why the object changes state
Sub Logbook1_Change (Reason As Integer)

    Select Case Logbook1.Status
        Case LOG_OFF
            Select Case Reason
                Case LOG_REAS_ACTION
                    Debug.Print "STOP: on command"
                Case LOG_REAS_MSG_LIMIT
                    Debug.Print "STOP: Message limit reached"
                Case LOG_REAS_TIME_OUT
                    Debug.Print "STOP: Time Out"
            End Select
        Case LOG_PAUSED
            Debug.Print "PAUSE: Reason = "; Reason
        Case LOG_ON
            Debug.Print "START: Reason = "; Reason
        Case LOG_ERROR
            Debug.Print "ERROR: Error   = "; Reason
    End Select

End Sub
```

# Logbook Control Installation

Welcome to the Logbook Control installation!

Before you proceed, maybe you should check out what is new in **this release**. Also make sure you understand how to use the different **Variants** of the control: VBX, 16-bit OCX and 32-bit OCX (ActiveX).

## Notes

The Logbook controls contained in the distribution archives are fully functional. However, you will require a license file in order to get rid of the warning dialog both at programming time and in executables. To register and obtain the license, see the **Information** page.

In order to keep the file size down, this archive does not contain the shared Microsoft DLL that are needed to run this control:

| | |
|---|---|
| MFC40.DLL, MSVCRT40.DLL | (32-bit version) |
| OC25.DLL | (16-bit version) |

You probably already have these DLLs on your system, but just in case, they are available for download, together with the latest release of the control, at **http://www.multimedia.it/andy/logbook/download** or from in most on-line software archives.

Depending on the package you have downloaded, you may be able to perform an automatic installation or a manual one, as described in the following.

## Automatic Installation Procedure

If you have unpacked the OCX distribution and are running Windows 95, NT 4 or anything better, you can **install** the software by double clicking on the **SETUP.EXE** program from the Windows Explorer. This will install the controls, help file and sample applications on your system.

Please check out the Manual Installation Procedure detailed below anyway. Point 4 explains how you can use Visual Basic to check that the controls are properly registered, while point 2 explains how to install the license file that you may receive separately.

To **uninstall** the software, use the Windows Control Panel, "software" applet. In the list you find on the Add/Remove Software page, you should see an entry named **Logbook Control**. Select it and click on the **Add/Remove** button.

## Manual Installation Procedure

1. Move the selected control file(s) and LOGBOOK.HLP in a directory of your choice. Usually custom controls are installed in the C:\Windows\System directory (Windows 3.x, Windows 95), but this is not a strict requirement.

   For simplicity, in the following we will assume you are installing on either a Windows 3.x or Windows 95 system. But, be aware that, on Windows NT computers, 16-bit controls normally go in the C:\Winnt\System directory, while 32-bit controls go in the C:\Winnt\System32 directory.

   In *design mode*, the control can be loaded from any directory, and it will attempt to locate the help file in the directory where the control was loaded from.

   In *run-time mode* (when used by a compiled program), the control should be somewhere in the search path.

2. The license file should be placed in the \WINDOWS directory. Of course, you should *not* distribute the license file with executable programs you write.

3. The LOGBOOK.BAS file can be placed in your program source code directory and added to your VB project. Using this file is not strictly needed with OCX versions of the control, since constants can be read and copied into your project source code using Visual Basic object browser.

4. Before you load any of the example programs, make sure you REGISTER the control with the system from Visual Basic. To do this, load Visual Basic with a new, blank project, then select the TOOLS menu, CUSTOM CONTROLS. Click the BROWSE button. Go to the directory where the OCX control is installed, then select it into the project. Close the Controls window clicking OK. At this point Visual Basic should have registered the control and it should work with all existing programs.

5. For automatic conversion of Visual Basic programs from VBX to the OCX version of the control, edit VB.INI (normally in c:\WINDOWS); locate the section named

   [VBX Conversions16]

   and add the following (on a single line):

   ```
   LOGBOOK.VBX={813AE543-40C5-11D0-B4D8-444553540000}#1.0#0;C:
   \WINDOWS\SYSTEM\LOGBK16.OCX
   ```

   You can do the same for the 32-bit version of the control. The VB.INI section to edit is [VBX Conversions32] and the line to add is:

   ```
   LOGBOOK.VBX={813AE543-40C5-11D0-B4D8-444553540000}#1.0#0;C:
   \WINDOWS\SYSTEM\LOGBK32.OCX
   ```

   Of course if you installed the controls in other directories than the default C:\Windows\System, you should correct the paths in the two statements above accordingly.

## Information

### Foreword

This control began its life as a spare time self-training project on custom controls and Windows programming. Day after day, feature after feature, the control became a nice piece of software, and it was really a shame to let it gather dust in a drawer.

This software was never actually intended for commercial use. However, just in case, there is a license mechanism built in that should convince you to check with the author before you attempt to make money out of it. Please read the following sections for details.

See also the **version history**.

I sincerely hope that you have as much fun using this software as I had writing it.

### Copyright Notice

The Logbook Control is Copyright © 1995 by A.Zanna. - All rights reserved.

Windows, Windows NT, Windows 95, Visual Basic, Excel, Access, Office, Visual Basic for Applications, VBScript, Windows Entertainment Packs, Visual C++, Front Page, ActiveX and Internet Explorer are either trademarks or registered trademarks of Microsoft Corporation.

### Disclaimer

THIS SOFTWARE AND THE ACCOMPANYING FILES ARE PROVIDED "AS IS" AND WITHOUT WARRANTIES AS TO PERFORMANCE OF MERCHANTABILITY OR ANY OTHER WARRANTIES WHETHER EXPRESSED OR IMPLIED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DAMAGES, INCLUDING ANY LOSS OF PROFITS, LOSS OF DATA, INCLUDING BUT NOT LIMITED TO SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR INDIRECT DAMAGES ARISING FROM THE USE OF THIS SOFTWARE.

### Distribution

The distribution package for this version of the Logbook software must include at least the following files:

| | |
|---|---|
| LOGBOOK.VBX | LOGBOOK.HLP |
| LOGBK16.OCX | LOGBOOK.BAS |
| LOGBK32.OCX | README.TXT |

This package can be freely distributed, provided that:

1) The distributor acknowledges that this software was designed, implemented and is still owned by the original author, who reserves all rights to it.
2) It is not bundled with a commercial product, with the possible exceptions of software collections as clarified below.
3) All the files in the package are included and no changes of any kind are made to the original files.
4) No charge or donation is requested for any copy of this software itself, however made, except for the cost of physical support, transmission medium and time required to perform the copy.

Given that clause #3 above is respected, it is explicitly allowed to include the Logbook software in shareware and freeware collections on networks, BBS and CD-ROMs or other distribution media, where a profit margin is taken for the sole sake of supporting the said software archives.

In all other circumstances, written permission must be obtained from the author prior to distribution.

The custom control files (LOGBOOK.VBX, LOGBK16.OCX, LOGBK32.OCX) can be distributed alone only as part of a commercial product and only if the author of that product has a valid license for the Logbook software. This license cannot be transferred to users or purchasers of the said commercial product.

## User License

This package can be freely used for any legitimate personal, non-commercial purpose.

When used as component of a commercial program, the producer is required to contact the Logbook author in order to obtain permission, in the form of a specific development license file (see *Registration Information* below).

The author reserves all right to change the distribution and licensing policy for any future major version of the Logbook software and derived products.

Source code for this package is not freely available. It can, however, be purchased from the author.

## About the Author

The author (Andy Zanna) is an electronic engineer with a broad experience in software design and production on a wide range of systems, from home computers to real-time systems, enterprise servers and large networks.

This software is in no way related to the author's real work or to the company where he is employed.

Registrations, information queries and suggestions should be addressed to:

A.Zanna

Hauptstr. 41
D-71263
Merklingen - Weil der Stadt
Germany

| | |
|---|---|
| phone: | +49-7033-33800 |
| E-mail: | zanna@multimedia.it |

## Registration Information

If you register this control, you will receive a license file that will get rid of the warning dialog that is displayed when the control is loaded at design or run-time. You will then be able to compile stand-alone applications and distribute them free of royalty. You will also receive notifications of updates and a right to use all minor updates (e.g. 1.x) with the same license.

The registration fee is US$ 25 or equivalent in major currencies (DM, UK Pound, F Franks), should be sent by *mail money order* to the address above.

You should add another 5$ if you require that the software and the license be sent to you via ordinary mail, rather than E-mail.

You should also send a *registration notice* by ordinary mail or E-mail (preferably) to the address above, indicating your name, company (if applicable), mail and E-mail addresses.

## AutoStart

This property, when *True*, makes the control start a log file on the first received message, exactly as if a *Start* **Methods**, or **Action** had been called.

Taking advantage of this property, extremely simple programs could contain no code for starting and stopping the log file.

| | |
|---|---|
| **Type:** | Integer |
| **Default Value:** | False |

## Control Variants

The Logbook control comes in three different variants, as described below. Depending on your development environment, you may want to install any or all of these.

| | |
|---|---|
| LOGBOOK.VBX | *(available in the VBX distribution only)* is suitable for use with 16-bit versions of Visual Basic. It will work fine with Visual Basic 3.0 and 4.0/16 bit. It will *not* work with 32-bit versions of the language or with Visual Basic for Applications |
| LOGBK16.OCX | *(available in the OCX distribution only)* is an ActiveX control suitable for use with all 16-bit environments that support this format, including Visual Basic 4.0/16 and 16-bit versions of Visual Basic for Application (i.e. 16-bit Microsoft Office products). |
| LOGBK32.OCX | *(available in the OCX distribution only)* is an ActiveX control suitable for use with all 32-bit environments that support this format, including Visual Basic 4.0/32 or better, 32-bit versions of Visual Basic for Application (i.e. 32-bit Microsoft Office products), Visual C++ and Web pages. |

*Note*: "ActiveX control" is just the most recent denomination of what used to be called OLE controls, or OCX's. With respect to the functionality of the control and its documentation, all three denominations are equivalent.

Note: the distribution archive should also include the following essential files:

| | |
|---|---|
| LOGBOOK.HLP | *the help file* |
| LOGBOOK.BAS | *support file for program development* |

## Web Pages

## General Considerations

The Logbook control (32-bit OCX version), like any ActiveX control, can be inserted in a Web page and used from VBScript or JavaScript programs. Users with an ActiveX-capable browser (currently only Microsoft Internet Explorer 3 or better) would then be able to load and see the programs.

However, given the specific functionality of the Logbook control, I believe it is very unlikely that a programmer will want or need to embed it in a web page rather than in a VC++ or Visual Basic program.

Just as a reference, if you really plan on doing this anyway, you will need all the documentation and tools that are available in the ActiveX SDK archive from the Microsoft Web site (**http://www.microsoft.com/activex**)

## Security

There is an important security issue with this control. By its nature, it must be able to write to an arbitrary-named physical disk file. As such, it is potentially dangerous. For instance, an Internet hacker may program the control to overwrite config.sys on one or more target computers.

For these reasons, the control is purposedly *not* marked as safe for loading and scripting in a Web environment. Internet Explorer *will* warn the user about potential security issues with it and will refuse to load it unless the safety level is set to *medium* or *low* in the Options panel.

## Property Pages

The **Logbook Control Properties** dialog box contains the following pages:

| | |
|---|---|
| **General** | Groups the most important properties that need to be set: what to call the log file, where to put it, how long the console history buffer should be and what level of severity is required before a message is logged |
| **Time** | Lets you specify how the control tracks time and stamps the messages in the log file. |
| **Messages** | Allows you to specify which fields of a message are shown and logged, as long as the default values for the client-specific fields. You can also set the field separator characters here. |
| **Open/Close** | Here you can enable the features that make the control automatically open and close the log file in specific circumstances. |
| **Colors** | Lets you specify the foreground and background colors for the console window text. |
| **Fonts** | Allows you to set font type, size and style for the control console window. |