



OCX Controls & software technologies  
<http://www.precisionocx.com>

---

## Precision Level Indicator

---

### Overview

---

#### Purpose

Precision Level Indicator is a powerful and professional-looking OCX control for your applications which can be used to graphically represent a quantity. It displays a cylindrical glass tank that is linearly filled with a colored liquid according to the value you want to represent, which can vary in a specified range. All the geometry parameters can be customized.

#### Optimization

Precision Level Indicator was written entirely in C++ and is composed of strongly optimized code, the distributed OCX file only contains the essential code needed to perform the required features. There are no memory leaks, all the allocated memory and graphic resources are gracefully released. The only required libraries are the MFC and MSVCRT dll that are shipped with every version of Microsoft Windows. You do not need any additional libraries.

#### Small binary size

There is no useless code into the executable. Level Indicator is only 40k in size ! This means that it is very fastly loaded into your application and it does not significantly increase your final application size.

#### High customisability

Precision Level Indicator has 16 parameters you can freely set, in order to vary the control geometry, and its look. For the best graphic appearance, you can decide the sharpness of the ambient lighting (regulating the gradient factor); conversely, for VERY high performances you can disable at all the color gradients and reach very fast redrawing time (> 100 Hz refresh frequency).

#### Flicker-free refresh

Precision Level Indicator makes use of a double-buffering technology in order to avoid unpleasant flickering effects: every graphic manipulation is done in memory and only eventually the final result is transferred to the video buffer, using all the available hardware optimizations that the graphic card provides for the BitBlt.

#### Any special requirement ?

If you are developing an application and found Precision Level Indicator nearly perfect for you needs except for small details, please feel free to contact us: let's talk about it !

#### Documentation

All the documentation you may need on Precision Level Indicator is on line !

---

## Quick start

---

You can start using Precision Level Indicator incredibly fast: the control default factory parameters are already

sensible for a basic use, you have only got to:

- | If you are using the full version (NOT the trial one) you must call the *SetLicense* method passing your license number as argument and make sure it returns *True*. Simply ignore this step if you are testing the trial version.
- | Set the *MinimumValue* and the *MaximumValue* properties to the desired values.
- | Set your *CurrentValue* over and over again.

Now you can take a more accurate look at the properties to fully exploit the power of Precision Level Indicator.

**Warning: in the trial version the current value is fixed to 50.** The same happens to full version as far as you set a correct license number. In the trial version the *SetLicense* function has no effect.

---

## Properties

---

### *BackColor*

**Domain:** any color

**Default value:** 0x00C0C0C0 (grey)

**Meaning:** this is the color used to fill the back plane of the control; this color will be lightened towards the lower and right border of the control, and darkened towards the upper and left ones, in order to simulate a sunken bevel.

### *CurrentValue*

**Domain:** any real number between *MinimumValue* and *MaximumValue*

**Default value:** 50

**Meaning:** this is the value you want to graphically represent with this control. If you set a value that is greater than the *MaximumValue*, the *CurrentValue* will be set to *MaximumValue*; conversely, if you try to set a value that is less than the *MinimumValue*, it will be brought to *MinimumValue*.

### *DrawTextLabels*

**Domain:** True, False

**Default value:** True

**Meaning:** If this property is set to True, the measuring marks on the front of the tank will be accompanied by a text label (formatted accordingly to the content of the *FormatString* property), which displays the actual value to which the mark corresponds. Text labels are actually written only if *NumberOfDivisions* is greater than 1 and if *FormatString* is not an empty string.

### *Font*

**Domain:** any font

**Default value:** MS Sans Serif, 10, Normal

**Meaning:** This is the font used to write the text labels.

### *FormatString*

**Domain:** an appropriate format string (see below)

**Default value:** %.0f

**Meaning:** : this string tells to the control how to format the text inside the labels; this is a standard C "*printf*()-style" format string, but do not worry if you do not know C, every information you will need is hereby given.

- | If you do not want a text label on your control, you can simply set this property to an empty string.
- | If you want to specify a simple text string (constant), set this property to the desired string (strings are replicated many times unchanged).
- | Otherwise, you can use a format string composed by a preceding text, then the placeholder for the value and a following text. The placeholder itself is composed of
  - | a leading percent sign, followed by
  - | an (optional) padding size indicator (the number of total characters which compose the number representation, including digits, spaces and punctuation); if you put a zero "0" before this indicator, the padding character is the zero, otherwise the space is used.
  - | an (optional) precision indicator, preceded by a dot (the number of digits after the decimal point; set it to 0 if you want integer values);
  - | and an "f" character, which stands for "floating point value".

Examples are provided in the following table. Since the percent character is used to indicate the placeholder, if you want to represent the percent sign itself, you will have use two percent sign.

Format string	Result
%f	50.000000
value=%f	value=50.000000
%.2f percent	50.00 percent
%06.2f	050.00
%.0f %%	50 %
blah %.0f blah	blah 50 blah

#### *GradientFactor*

**Domain:** any real value between 0 and 1

**Default value:** 0.7

**Meaning:** this value represent the sharpness of the ambient lighting: when this property is set to 0, the liquid inside the cylindrical tank is uniformly lit, when it is set to 1, the illuminated side of the tank (the left or the upper, depending on the value assumed by the *HorizontalLayout* property), strongly reflects light (and therefore is nearly white), whereas the shaded side is nearly black. For intermediate values, increasing this values makes shadows and reflexes sharper, whereas decreasing it makes them softer. If you try to put this property to a illegal value, it will be rebrought to the nearest border value.

**WARNING:** this value is actually used only if the *UseColorGradients* property is set to True, otherwise it will be ignored.

#### *HorizontalLayout*

**Domain:** True, False

**Default value:** True

**Meaning:** When this property is set to False (the default), the tank is vertically positioned and (as in the reality) liquid fills it from bottom to top. When this property is set to True, the tank is horizontal and liquid fills it from left to right.

#### *LiquidColor*

**Domain:** any color

**Default value:** 0x000000FF (red)

**Meaning:** this is the color of the liquid inside the tank. This color will be made lighter or darker to simulate reflexes and shadows, according to the values of the *GradientFactor* and *UseColorGradients* properties.

#### *MaximumValue*

**Domain:** any real number

**Default value:** 100

**Meaning:** this is the maximum value you will allow the *CurrentValue* to assume.

#### *MinimumValue*

**Domain:** any real number

**Default value:** 0

**Meaning:** this is the minimum value you will allow the *CurrentValue* to assume.

#### *NumberOfDivisions*

**Domain:**any integer number (short)

**Default value:**10

**Meaning:** this is the number of intervals in which the full range between *MinimumValue* and *MaximumValue* is divided. Therefore there will be *this\_number*+1 measuring marks drawn on the front side of the tank. This marks will be also accompanied by text labels, accordingly to the contents of the *FormatString* and *DrawTextLabels* properties. If you do **not** want this measuring marks to be drawn, put this property to zero.

#### *NumberOfSubdivisions*

**Domain:** any integer number (short)

**Default value:** 2

**Meaning:** this is the number of sub-intervals in which the range between a mark and another will be furtherly subdivided. Therefore there will be *this\_number*-1 measuring sub-marks between a mark and another. Subdivisions will not be accompanied by text labels. If you do **not** want this measuring marks to be drawn, put this property to zero.

### *RelativeDepth*

**Domain:** a real number between 0 and 1

**Default value:** 0.25

**Meaning:** This is the ratio between the apparent depth of the tank and the width (or the height, if the control is horizontally positioned) of the control bounding box. Increase this value to make the tank seem deeper (or to raise/shift right the apparent point of view); decrease this value to make the tank seem flat (or to lower/shift left the apparent point of view). If you try to put this property to a illegal value, it will be rebrought to the nearest border value.

### *RelativeHeight*

**Domain:** a real number (recommended between 0 and 1)

**Default value:**0.9

**Meaning:** this is the ratio between the tank height and the bounding box height. If you increase this value too much, your tank will be partially covered by the shaded area (if present), or it will partly finish out of the bounding box. Choose this value according with the value of the *ShadedAreaRelativeSize* property.

### *RelativeWidth*

**Domain:** a real number (recommended between 0 and 1)

**Default value:**0.9

**Meaning:** this is the ratio between the tank width and the bounding box width. The same considerations as the previous item apply. Choose this value according with the value of the *ShadedAreaRelativeSize* property.

### *ShadedAreaRelativeSize*

**Domain:**a real number (recommended between 0 and 0.5)

**Default value:** 0.05

**Meaning:** this is the ratio between the shaded border and the bounding box minimum side. Increase this value to make the control seem more deeply sunken, put it to zero if you do NOT want a sunken bevel. If you increase this value too much, the shaded and illuminated borders will unpleasantly overlap, or they will cover the tank.

### *UseColorGradients*

**Domain:** True, False

**Default value:**True

**Meaning:** this value indicates whether the tank is to be drawn in order to simulate reflexes and shadows. Putting this value to False apparently brings to the same effects than putting *GradientFactor* to zero, with one important difference: when this property is set to False, the whole gradient calculation code section is skipped, with dramatic performance improvements. Do this to reach a very high refresh rate (>100 Hz).

---

## Methods

---

### *bool SetLicense(long LicenseNumber)*

**Parameters:** *LicenseNumber*, a long integer number which represent your license number.

**Return value:** a boolean value that represents the result of the authentication.

**Meaning:** this method is used to activate the full-version control. You should call this method before calling any other method, passing the license number you obtained at purchase time as *LicenseNumber* parameter, otherwise the control will not draw itself on the screen.

**Attention:** you do not need to use this method using the trial version of the control.

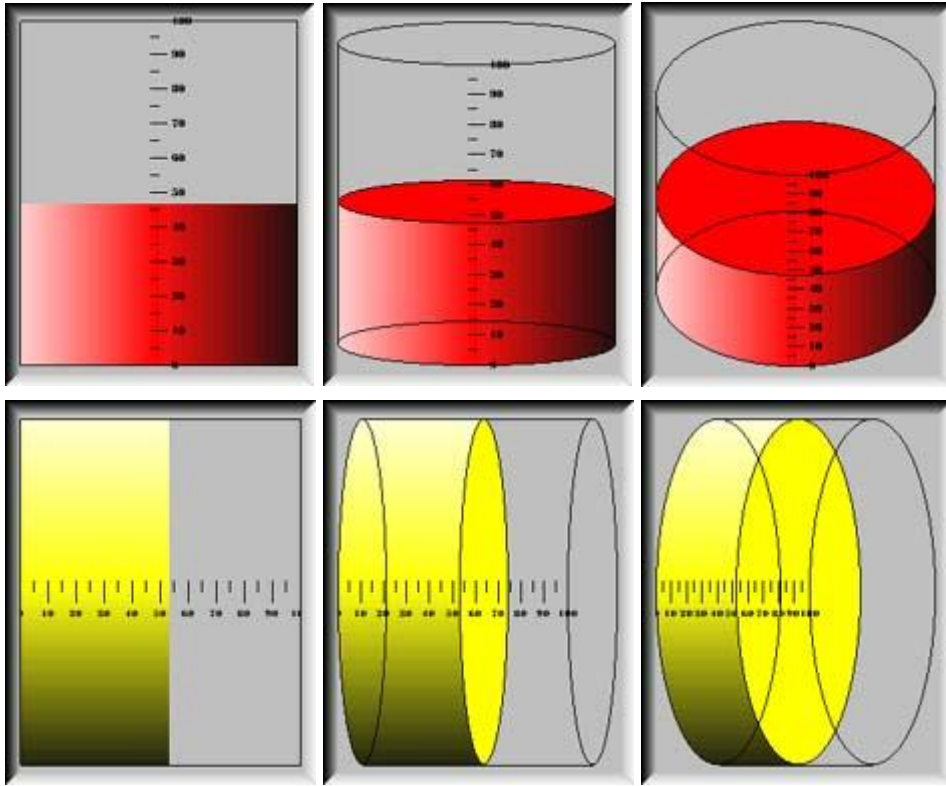
---

## Art Gallery

---

### Varying the relative depth

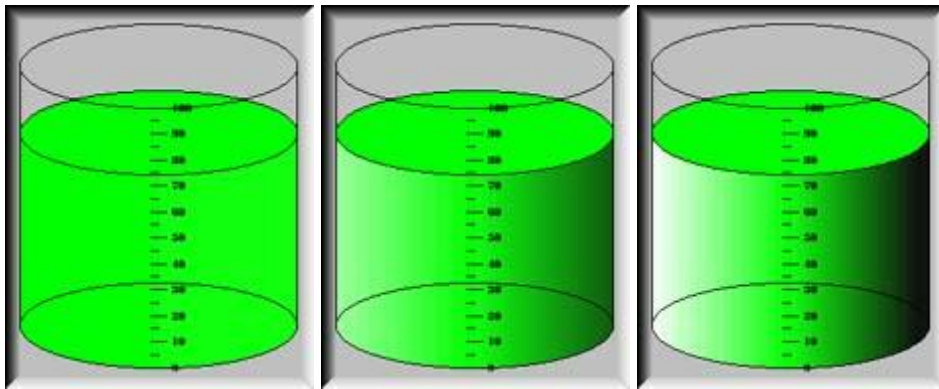
---



---

### Varying the gradient factor

---



---

### Varying the height and width factors

---

