

CREATING GAUGES WITH EASYGAUGE, by Hector Molina

Introduction

My first experience with MS Flight Simulator was with the 98 version. Soon, my children discovered some of the specialized web sites that publish third party's aircrafts, sceneries, panels and gauges. Then I realized the large world of new opportunities to interact with the game.

I got myself a utility software that allows to graphically edit the panel.cfg file and soon I was creating my own panels using third party's gauges and publishing them at specialized sites. Problems began soon thereafter, when I started to receive emails from the authors complaining because I was using their gauges without authorization. I immediately removed my panels from the sites and got a very frustrating feeling because I was not able to create my own gauges. Needless to say that I am a complete ignorant with regards to C or C++ programming.

By a coincidence I ran into a gauge that made reference to a software called EasyGauge. After doing some research I finally discovered this wonderful software that allows you to create gauges without any programming knowledge.

Most likely the great majority of you uses a word processor or an electronic spreadsheet almost every day. Those utilities are nothing but graphical interfaces that allow a normal user to edit text or make complicated calculations without having to get into program languages. Well, two smart Germans, Marcel and Steffen Burr, created what today may seem obvious: a graphic interface to program gauges for FS without having to get into programming language itself.

And that is exactly what EasyGauge (EG) is: a friendly software that allows programmers ignorants like myself to create gauges for FS. Of course, the software has some restrictions and very sophisticated gauges cannot be created with EashGauge, but the great majority of the gauges that you would normally use in a panel can be created with this software.

The purpose of this tutorial is to help the software beginners understand some of the principles involved in designing gauges with EG.

To follow this tutorial you must have EG version 2.0 (or later) installed in your computer as well as FS 2002 or 2004. You may acquire EG at www.easygauge.net. Also, to better understand this tutorial you should have read thoroughly the EG manual first in order for you to be familiar with its functions and commands.

THE FS AMBIENT

You may consider FS as an ambient which broadcasts variables (Token variables) at high rates (nearly 100 per second). Examples of such token variables are: altitude, airspeed, radio frequencies, engine information (N1, N2, ITT, oil pressure, oil temperature, fuel), bearings, headings, cruises, etc. Gauges will normally “read” one or more of those variables and react to them (i.e. a needle will move as a reaction to the airspeed variable) displaying the result of that “reading”. You can also interact with the FS ambient through the use of Events, which send commands to your aircraft. Examples of Events are: throttle increase and decrease, flaps extension and retraction, radio frequency setting, pitch increase and decrease, landing gear up and down, etc.

The FS ambient depends on the aircraft design, therefore it may vary according to the code established in both the aircraft.cfg and the .air files. The first one defines the type of engine (jet, turbo prop, reciprocate engine), available frequencies, autopilot characteristics, aircraft identification, etc. The .air file defines the aircraft dynamics. Both files may limit the variables available within the FS ambient. For instance, if the aircraft.cfg defines that the flaps will have 3 positions available, it would be useless if you try to create a gauge with more flap positions.

Take this in mind when you design a gauge, particularly because most of the time the aircraft designer is not the same person that designs panels and gauges.

CREATING GAUGES

This chapter will show you how to create one of the most complex gauges of a panel, the Primary Flight Display (PFD), and will use my Boeing 737-800 panel as an example. The reason for choosing this gauge is because it contains all the elements you can use to create gauges: static, sprites, movings, needles, sliders and strings. Should you like to see the panel as you read this tutorial you may download it from either flightsim.com or simviation.com. The name of the file is b7378pn2.zip.

From experience I found out that the best way to begin any gauge is to first design a bitmap showing how the gauge will finally look and then use portions of that bitmap to create the different elements.

Our PFD will finally look as follows:

(0,0)



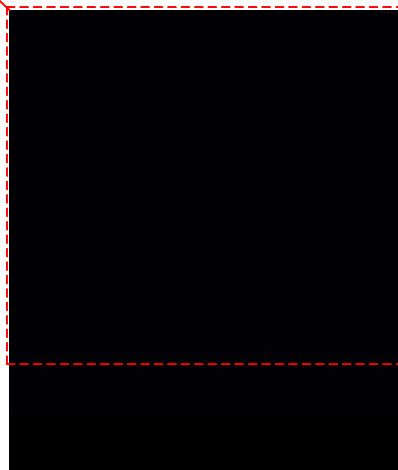
Center of rotation of the
Attitude Indicator.
Coordinates (100,108)

+ Center of rotation of the HSI. Coordinates (99,261)

The Static

I prefer to use a transparent bitmap (RGB color 0,0,0) for the static because this allows you to increase its size when you place it on the panel, without hiding adjacent areas. Moreover, the center of rotation of the HSI is below the bottom of the gauge, therefore the static has to be high enough to contain that point, and it has to be transparent in order not to cover any part of the panel. Remember, color (0,0,0) will always be displayed transparent. So our static is a 225 x 263 pixel transparent bitmap.

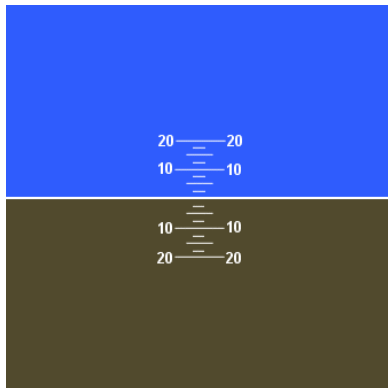
Coordinate (0,0)



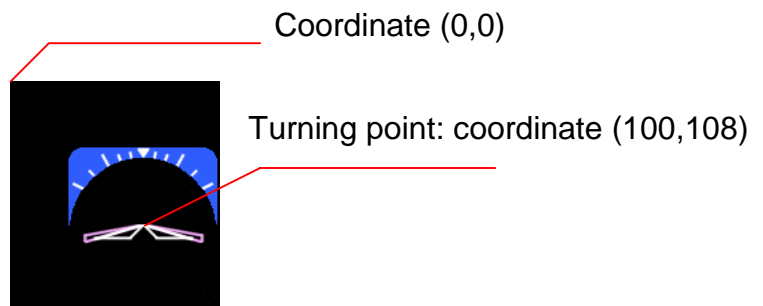
Only this portion of the gauge will
be shown on the panel

The attitude indicator

This indicator moves vertically and it also turns, therefore we need to use a sprite element. We need two bitmaps for this element: the sprite itself and the mask, as follows:

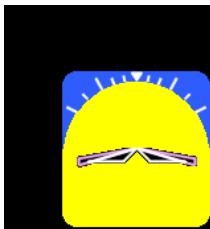


335 x 335 pixel bitmap



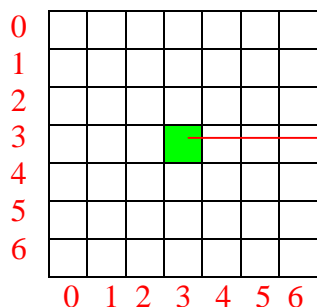
158 x 171 pixel bitmap

The portion of the mask that will show the sprite has to be RGB (1,1,1). The image below shows the mask with the display area painted on yellow, just to show its shape. The rest of the area is transparent (0,0,0)



Tips for sprites

a) Sprites are to be perfectly square and should have an odd number of pixels, thus you will make sure that there is a turning point positioned exactly in the center of the image. For instance, the image below is 7 x 7 pixel (the first pixel is always 0):



This pixel is exactly in the center of the image

If you use an even number of pixels you won't have a perfectly centered turning point. Although for rather large sprites this may not be noticed, for small images you may get an eccentric turning effect.

b) The mask does not have to be the same size of the static but it has to be large enough to cover the area of the visible portion of the sprite, always measured from the top left corner of the static. Remember, the smaller the bitmaps of your gauges the better the frame rates you get with your panel.

c) The size of the sprite has to be large enough to allow it to perform the maximum displacement in every direction and still be visible through the mask. In our case I used a 335 x 335 pixel image for the sprite, thus it will be shown even at a 40° or a 50° pitch.

Setting

As per the EG manual, the setting coordinates of the sprite are the coordinates of the point of the static around which the sprite will rotate, always measured from the top left corner of the static, whose coordinates are always (0,0). In our example the general setting for the attitude sprite will be (100,108)

Token variables

For the vertical displacement we use:
ATTITUDE_INDICATOR_PITCH_DEGREES

For the turning movement we use:
ATTITUDE_INDICATOR_BANK_DEGREES

Scale

Before defining the scale it's better to create the sprite element and then select it again for editing, thus allowing the element to be displayed in the working window of EG. By doing this it's easier to graphically define the maximum and minimum values of Y and O.

The ATTITUDE_INDICATOR_PITCH_DEGREES returns positive numbers when pitching up and negative numbers when pitching down. You have to define how far the sprite will vertically move when pitching up or down. Let's assume that the maximum pitch up and down will be 40° and -40° , respectively. Then, on the field for minimum Y we introduce 40 and we click on the change button of the left field and then click on the upper portion of the image to define the point where the sprite will move when pitch is 40° up. To do so, we use the sprite degrees scale as a reference and even though 40° is not drawn in the scale just click on a point that is approximately 40° above the horizon. Repeat the same procedure for the

down pitch. Introduce -40° for Maximum Y, click over the change field and click over the image approximately 40° below the horizon.

All this procedure only sets the movement scale of the pitch but it will not prevent the sprite from moving beyond the two limits that you have just selected. If you wish to limit the sprite movement, then you have to do so in the function section.

Introduce the following code:

```
IF ValueReturn Y>40 THEN ValueReturn Y=40
```

```
IF ValueReturn Y<-40 THEN ValueReturn Y=-40
```

The code above will limit the sprite movement to a maximum of 40° up and -40° down.

Finally add the following code:

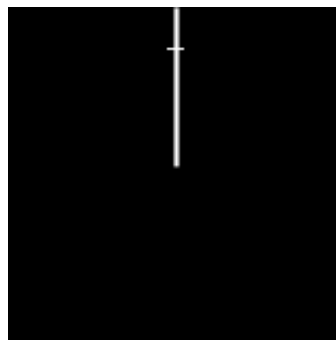
```
IF BATTERY_VOLTAGE < 24 THEN ValueReturn O= 0.
```

The reason for this is that when the battery starts to run out of power, the bank variable returns negative numbers. Even though this may be valid for a backup attitude indicator, the PFD is an electronic instrument and therefore if the battery has low power the value shown should be zero (no bank at all).

Now let's create the HSI. We need the following sprite bitmaps:



Rose: 155 x155 pixels



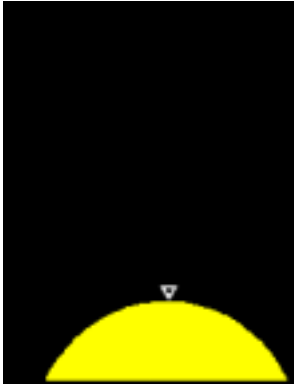
CRS: 155 x 155 pixels



HDG: 175 x 175 pixels

Notice that the HDG bitmap is 20 pixels larger than the others. This is because the heading bug has to rotate over the rose circle.

For each sprite bitmap we need a mask bitmap. As described for the attitude indicator, the HSI masks have to be large enough to display the portion of the sprite we want to show in the gauge.



Rose mask
177 x 232 pixels



CRS mask
177 x 232 pixels



HDG mask
177 x 232 pixels

Again, I painted with yellow the area that should be RGB (1,1,1,) just to show its shape. All the rest is transparent (0,0,0)

Notice that the masks DO NOT need to contain the rotation point of the sprites, therefore they are smaller than 262 pixels (the vertical coordinate of the rotation point). Thus, you may use smaller bitmaps and save frame rates. Generally speaking, masks are to be just large enough to contain the area through which you want the sprite to be displayed.

Settings

The settings for the three sprites above are (99,261), which are the coordinates of the point around which the sprites will turn, always measured from the top left corner of the image.

Variables for the HSI

For the rose: in the “display variables” section maintain the X-axis and the y-axis as `MODULE_VAR_NONE`, since they will not affect the Rose. For the 0-axis, select `PLANE_HEADING_DEGREES_MAGNETIC`.

In the Scale section enter Scale of turn 1

For the CRS: go to the Function section and enter the following code:

```
ValueReturn 0 = HSI_OBI_NEEDLE + 360 -  
PLANE_HEADING_DEGREES_MAGNETIC
```

In the Scale section enter Scale of turn -1

For the HDG: go to the Function section and enter the following code:

```
ValueReturn 0 = AUTOPILOT_HEADING_LOCK_DIR + 360 -  
PLANE_HEADING_DEGREES_MAGNETIC
```

In the Scale section enter Scale of turn -1

The Bank Arrow

We need the following bitmap for this needle:



53 x 14 pixels image

Turning point. Coordinates (1,7)

The background of the image is RGB (0,0,0) because we only want to arrow the be displayed.

The settings are the same as for the Attitude indicator, since the bank arrow will turn around the same point. Coordinates (100,108). We want this needle to move from -60° to 60° .

In the function section enter the following code:

```
ValueReturn = ATTITUDE_INDICATOR_BANK_DEGREES + 60
```

When dealing with arrows I have had less problems working always with positive values. Therefore, the above code forces to return values from 0 to 120, instead of negative numbers when banking to the left.

In addition, introduce the following code:

```
IF ValueReturn < 0 THEN ValueReturn = 0
```

```
IF ValueReturn > 120 THEN ValueReturn = 120
```

This will limit the arrow displacement between 0 and 120 (-60° and 60°)

Finally, and for the same reason explained for the attitude indicator, enter the last code:

```
IF BATTERY_VOLTAGE < 24 THEN ValueReturn = 60
```

In the Nonlinearity Table section select the points for the zero position (-60°) and the 120 position (60°).

In the Points on picture section select 1,7 for the turning point of the arrow.

The Flight Director Bars

For these elements we need the following bitmaps:



Horizontal bar: 65 x 3 pixels



Vertical bar 3 x 65 pixels

Since these elements will displace in just one direction, either horizontally or vertically, we use the slider element.

Settings for the vertical bar: 100,76

Settings for the horizontal bar: 68,107

For the horizontal bar enter the following code in the function section:

```
ValueReturn Y = FLIGHT_DIRECTOR_PITCH –  
ATTITUDE_INDICATOR_PITCH_DEGREES
```

I normally limit the bar displacement to 15° , therefore I would introduce the following code:

```
IF ValueReturn Y < -15 THEN ValueReturn Y = -15
```

```
IF ValueReturn Y > 15 THEN ValueReturn y = 15
```

For the vertical bar enter the following code in the function section:

```
ValueReturn X = FLIGHT_DIRECTOR_BANK –  
ATTITUDE_INDICATOR_BANK_DEGREES
```

I normally limit the bar displacement to 30° , therefore I would introduce the following code:

```
IF ValueReturn X < -30 THEN ValueReturn X = -30
```

```
IF ValueReturn X > 30 THEN ValueReturn X = 30
```

The Vertical Speed Needle

For this element we need the following bitmaps:



21 x 11 pixels background



56 x 5 pixels arrow

The yellow color should in fact be (0,0,0). It is yellow just to show its shape.

The rest is black (3,3,3)



28 x 177 pixels cover

The background could in fact be drawn directly onto the static image. However, I like to introduce the function Light Image to all my gauges in order to automatically illuminate at night, and since the static cannot receive that function, I create the background as an Icon. In this case the settings for this icon are: 204,54

The settings for the arrow are: 224,129

I use VERTICAL_VELOCITY to drive this arrow. Since that Token variable returns the value in feet/second and we want it feet/minute, we have to multiply it by 60. Also the value return may vary between -6000 and 6000 feet/minute, whether the plane is pitching down or up. Since I prefer to work with positives values when dealing with needles, then I introduce the following code in the Function section:

```
ValueReturn = VERTICAL_VELOCITY * 60 + 6000
```

```
IF ValueReturn < 0 THEN ValueReturn = 0
```

```
IF ValueReturn > 12000 THEN ValueReturn = 12000
```

The code above will limit the arrow displacement within a range of -6000 and 6000 feet/minute.

The purpose of the cover is to hide the arrow from all the areas that are out of the blue area of the background, thus adding a “stretching” effect to the arrow as it moves up and down. We use the Icon element for this cover with the following settings: 196,0. This element MUST be placed after the needle in order to make sure that the arrow is displaced only through the desired area.

The Glide Slope

For this element we need the following bitmaps:



7 x 100 pixel background



7 x 11 pixel slider

RGB (0,0,0)

The settings for the background icon are: 157, 59

We want this image to be displayed only when the VOR1 flag is available;
therefore, in the function section we introduce the following code:

If VOR1_GS_FLAG = 1 THEN Show Element else Hide Element

For the arrow we use the slider element:
The settings are: 158,103

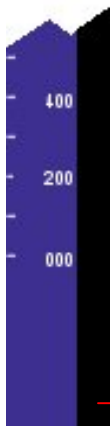
In the Display variables section we use the following variable for the Y-axis:
VOR1_GS_NEEDLE

The same as with the background we only want it to be displayed when the flag
is available, therefore in the Function section we introduce the same code:

If VOR1_GS_FLAG = 1 THEN Show Element else Hide Element

The Main Altimeter

To create this indicator we use a moving and we need the following bitmaps:



44 pixel wide moving
The height will depend on the
Maximum altitude you want to
display. In this case is 45,000 feet.

— This color is (0,0,0)



44 x 152 pixel bitmap
The yellow color in fact
should be RGB (1,1,1)
The black color is (0,0,0)

Even though the EG manual shows how to create an altimeter based on a moving, I decided to include it this tutorial in order to give you a couple of tips on how to define the maximum and minimum values. The reason for this is because I have seen in the EG forum several threads of users that have problems with the correct values displayed by this element.

But first, let's create the element. The settings are: 167,33

In the Display variables section choose
`ALT_FROM_BAROMETRIC_PRESSURE` as your y-axis variable.

To define where to start and to finish the altitude scale in the moving, use your drawing software, Paint Shop Pro for instance, and first create a 44 pixel wide bitmap with a height of around 2000 pixels, just to begin with. In order to define where to draw the 000 altitude, copy part of the mask image and paste it over the very bottom of the moving image as follows:



Then, start drawing your scale upwards beginning from 000 as shown in the figure. Thus, you will make sure than when the Value return is zero, the moving will be placed at the bottom of the mask. After you define the zero point erase the portion of the mask you used for reference.

When you finish drawing the scale to the maximum altitude, repeat the same procedure as follows:



Remove from the moving image everything that is above the top border of the mask.

Place a copy of part of the mask onto the moving image adjusting the moving in order that the arrow coincides with the maximum altitude you want to be displayed (in this case, 45,000 feet). Then remove from the moving image the portion above the top border of the mask. Thus, you will make sure that when the altimeter shows the maximum value you want to be displayed (in this case 45000 feet), the top of the moving image coincides with the top of the mask. After this procedure, erase the portion of mask you used as reference and finish drawing the scale until the top of the moving image.

Now, getting back to the moving element in EG, go to the Scale section and define the Value of the top border as 45000 and then the Value of the bottom border as 0. Thus, your scale will move precisely between those two limits and will accurately show the altitude returned by the FS token variable.

All the procedure above is valid for the other 4 altimeters. You will need the following images for each of them:



Tens mask
14 x 21 pixels

Tens moving
14 x 140 pixels



100s mask
7 x 21pixels

100s moving
7 x 201 pixels



1000s and 10000s
mask
8 x 21 pixels

1000s and 10000s
moving
8 x 202 pixels

The settings for each moving are:

For the tens: 196,99

For the 100s: 191,99

For the 1000s: 184,99

For the 10000s: 176,99



For all these movings select ALT_FROM_BAROMETRIC_PRESSURE as the variable to drive the Y-axis, in the Display variables section. However, in the function section, you have to introduce the following code in order to make sure that you get the right value returns:

For the tens:

ValueReturn Y = ValueReturn Y % 100 (if the variable returns 36250 feet, for instance, you only want to have the 50 feet portion displayed by this element)

For the 100s:

ValueReturn Y = (ValueReturn Y – ValueReturn Y % 100)/100

ValueReturn Y = ValueReturn Y % 10 (if the variable returns 36250 feet, for instance, you only want to have the 2 digit displayed by this element)

For the 1000s:

ValueReturn Y = (ValueReturn Y – ValueReturn Y % 1000)/1000

ValueReturn Y = ValueReturn Y % 10 (if the variable returns 36250 feet, for instance, you only want to have the 6 digit displayed by this element)

For the 10000s:

ValueReturn Y = (ValueReturn Y – ValueReturn Y % 10000)/10000 (if the variable returns 36250 feet, for instance, you only want to have the 3 digit displayed by this element)

The Scale values for all 4 altimeters above are: Value of the top border = 9 and Value of the bottom border = 0

The Indicated Air Speed (IAS)

For this element we need the following bitmaps:

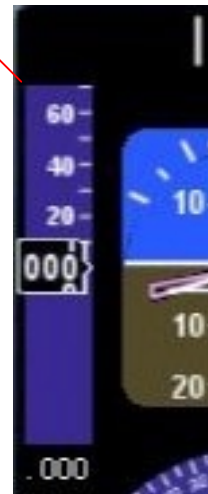


32 pixel wide moving. The height will depend on the maximum height you want to display.



32 x 152 pixel mask. The yellow color should in fact be (1,1,1)

The settings for this element are: 1,32

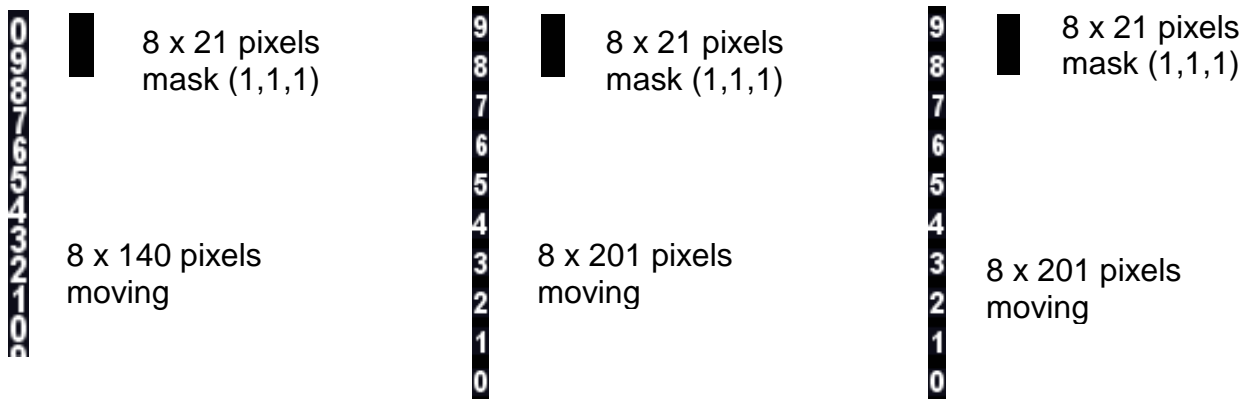


In the Display variables section select AIRSPEED for the variable driving the y-axis.

Use the same procedure explained for the altimeter to define where to start drawing the 000 speed scale and where to cut the top portion of the image. Thereafter go to the Scale section and define 0 as the Value of the bottom border and define your maximum displayed speed as the Value of the top border.

The Secondary IAS

For these elements we need the following bitmaps:



The settings for these elements are:

Units: 19,98
Tens: 11,98
Hundreds: 3,98



For all three movings select AIRSPEED as the variable that will drive the Y-axis

For the 'units' moving, in the function section, introduce the following code:

ValueReturn Y = ValueReturn Y % 10 (if, for example, the variable returns a value of 256 knots, you want this moving to show only the 6 digit)

For the 'tens' moving, in the function section, introduce the following code:

ValueReturn Y = (ValueReturn Y – ValueReturn Y % 10)/10

ValueReturn Y = ValueReturn Y % 10 (if, for example, the variable returns a value of 256 knots, you want this moving to show only the 5 digit)

For the 'hundreds' moving, in the function section, introduce the following code:

ValueReturn Y = (ValueReturn Y – ValueReturn Y % 100)/100 (if, for example, the variable returns a value of 256 knots, you want this moving to show only the 2 digit)

The Speed Bug

For this element we use a slider and the following bitmap:



13 x 8 pixels

The settings for this element are: 26,104 (this coordinates will place the slider exactly in the middle of the speed mask)

In the function section introduce the following code:

```
ValueReturn = AUTOPILOT_AIR_SPEED_HOLD_VAR – AIRSPEED
```

```
If AUTOPILOT_AIRSPEED_HOLD = 1 Show element else Hide element
```

```
If ValueReturn > 70 then ValueReturn = 70
```

```
If ValueReturn < -70 then ValueReturn = -70
```

The last two codes will limit the slide displacement to values between 70 and – 70. These values correspond to half the displayed speed scale and consequently half the height of the speed mask.

Attention: if we leave the speed bug slider just like explained above it will move over the secondary speed indicator. We don't want that because it will not look nice and will not correspond to reality. Therefore, we have to add and icon to cover the passage of the slider over the secondary speed indicator. To do this we need to place an icon with the following bimap:



28 x 23 pixels

The settings of this icon are: 3,97

The three movings of the secondary speed indicator have to be created AFTER the creation of the icon above otherwise the icon will cover the movings. If you have already created the movings, just push them down the "list of existing elements" of EG by using the down bottom.

The Altitude Bug

For this bug we also use a slider with the following bitmap:



16 x 19 pixels

The settings for this element are: 164,101

In the Function section introduce the following code:

```
ValueReturn = AUTOPILOT_ALTITUDE_LOCK_VAR –  
ALT_FROM_BAROMETRIC_PRESSURE  
If AUTOPILOT_ALTITUDE_LOCK = 1 then Show element else Hide element  
If ValueReturn > 450 then ValueReturn = 450 (this is half the value of the  
displayed altitude scale)  
If ValueReturn < -450 then ValueReturn = -450  
The last two statements limit the slider displacement within the top and the  
bottom of the altitude mask.
```

The same as with the speed bug, we have to add an icon in order to avoid the bug to pass over the altitude indicator. For this we need the following bitmap:



18 x 23 pixels

The settings for this icon are: 171,98

The Secondary Altitude Indicator

For this element we need the following bitmaps:



14 x 21 pixel
mask

14 x 140 pixel
moving for the
tens



7 x 21 pixel
mask

7 x 201 pixel
moving for the
100s



Two 8 x 21 pixel
masks, one for the
1000s and one for
the 10000s

Two 8 x 202 pixel
movings, one for the
1000s and one for
the 10000s

The settings for each moving are:

For the 10s:	196,99
For the 100s:	191,99
For the 1000s:	184,99
For the 10000s:	176,99



In the Display variables section select ALT_FROM_BAROMETRIC_PRESSURE as the variable that will drive the Y-axis for the four movings above.

In the Function section, introduce the following code:

For the 10s:

ValueReturn Y = ValueReturn Y % 100 (if the variable returns an altitude of 35670 feet, for instance, you want this moving to show only the 70 portion)

For the 100s:

ValueReturn Y = (ValueReturn Y – ValueReturn Y % 100)/100

ValueReturn Y = ValueReturn Y % 10

(if the variable returns an altitude of 35670 feet, for instance, you want this moving to show only the 6 digit)

For the 1000s:

ValueReturn Y =(ValueReturn Y – ValueReturn Y % 1000)/1000

ValueReturn Y = ValueReturn Y % 10

(if the variable returns an altitude of 35670 feet, for instance, you want this moving to show only the 5 digit)

For the 10000s:

ValueReturn Y = (ValueReturn Y – ValueReturn Y % 10000)/10000

(if the variable returns an altitude of 35670 feet, for instance, you want this moving to show only the 3 digit)

The Top Indicator Separator

For this element we use an icon with the following bitmap:



56 x 24 pixels

Again, this element could be drawn directly onto the static. However, if you want to use the “light image” function, you must use an icon.

The settings for this element are: 74,1

I will not get into specific explanation for the strings since they are pretty obvious to create. One hint, though, with regards to the STD string over the Kohlsman indicator:

The STD should be displayed only when the Kohlsman is set to 29.92 inches. However, if you just introduce this statement as above, the string will never be displayed since FS uses several decimals. In order to make sure that the string with the word STD is displayed correctly at 29.92 inches, do the following: In the Main Function section create a variable with any name you want; for this example, let's call it 'baro'. This variable has to be a floating number since it will have several decimals. Then, create the string with the word STD and in the function section introduce the following code:

If baro > 29.915 and baro < 29.925 then Show element else Hide element

In the Main Function section introduce the following code in the Always field:

```
baro = KOHLSMAN_SETTING_HG
```

O.K. All you have to do now is to add the other strings and your PFD will be ready.