Slices in a LinePlot Display

This “tutorial” was created by Bernhard Schaffer (FELMI/ZFE) during the course of figuring out, how the different slices in a LinePlot display are arranged. It is merely a “memory scratch”, so use it with caution. If you find any strong mistakes, please contact me at Bernhard.Schaffer @ felmi-zfe.at

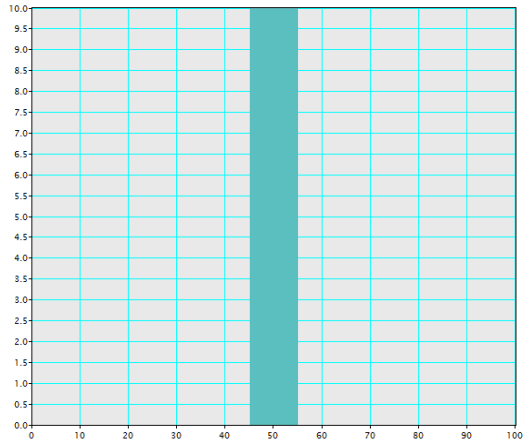
There are several related coordinate systems in a LinePlot Display. This tutorial should help to sort things out concerning how the systems are related and can be used.

- All slices in a LinePlot have their own *image-coordinate-system* (ICS).
- All slices in a LinePlot are grouped into one group, and this group has a *group-coordinate-system* (GCS).
- Also if only one slice is present, it is “grouped” into one group.
- A LinePlot has a certain displayed area called display. This has a *display-coordinate-system* (DCS)
- All coordinate systems (ICS, GCS, DCS) come in uncalibrated units.

Additionally to the coordinate systems of the LinePlot there is the coordinate system of the displaying window having a coordinates for the position of the *content* (within the window) and a *window position* (within the application window). These coordinate systems are not dealt with in this tutorial.

Lets now start with a simple LinePlot containing one slice, which is not calibrated and has 100 channels:

```
image IMG := RealImage("Test",4,100,1)
IMG = 0
IMG[0,45,1,55]=10
IMG.showimage()
```



The display is assumed to have the size 1 x 1. The group of slices is “scaled” into this display and thus has *group-to-display* transformations (scale and offset) GCS → DCS, which can be accessed and changed using the two commands:

- `LinePlotImageDisplayGetGroupToDisplayTransform()`
- `LinePlotImageDisplaySetGroupToDisplayTransform()`

Lets look at the example above and read out the transformation values with the following script:

```
image IMG
imagedisplay DISP
object SLICEID

IMG.GetFrontImage()
DISP = IMG.ImageGetImageDisplay(0)
SLICEID = DISP.ImageDisplayGetSliceIDByIndex(0)

number GDxs,GDxo,GDis,GDio
DISP.LinePlotImageDisplayGetGroupToDisplayTransform(SLICEID,GDio,GDis,GDxo,GDxs)
result("\n\n GROUP TO DISPLAY TRANSFORM yields:")
result("\n\t x: scale = "+GDxs+"\t offset:"+GDxo)
result("\n\t I: scale = "+GDis+"\t offset:"+GDio)
```

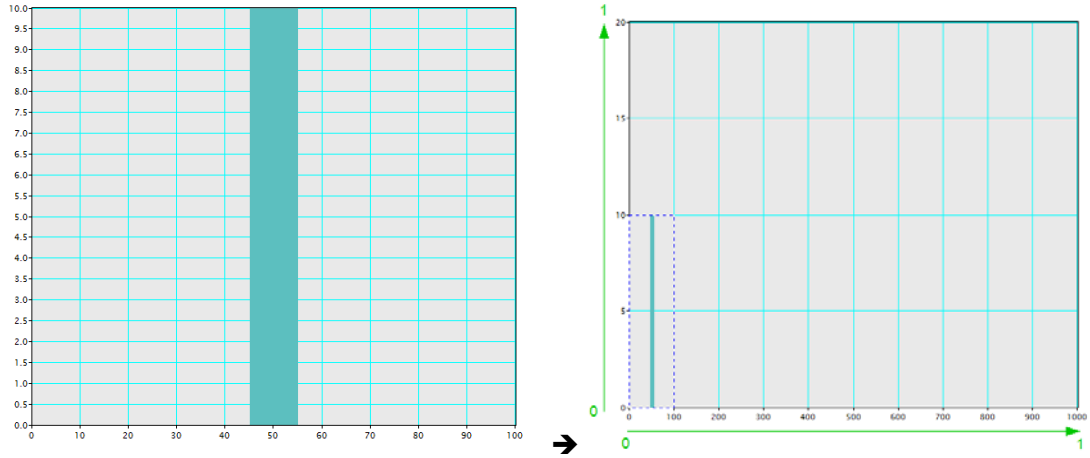
This yields the output:

```
GROUP TO DISPLAY TRANSFORM yields:
      x: scale = 0.01 offset:0
      I: scale = 0.1  offset:0
```

What does this mean? We have a group with 100 channels (X axis) and a value range from 0 to 10 (I axis). Therefore we have a display (100x10) which is transformed onto our display (1x1) by a scaling of (0.01x0.1) and no offset. Things become more clear if we now use the following script to change this GCS → DCS transformation to a different scaling of (0.001x0.05).
Please note, that the command takes the I-axis parameters before the X-axis parameters while I will follow to reference for the coordinates in the more natural way of X-axis / I-axis.

```
image IMG
imagedisplay DISP
object SLICEID
IMG.GetFrontImage()
DISP = IMG.ImageGetImageDisplay(0)
SLICEID = DISP.ImageDisplayGetSliceIDByIndex(0)
DISP.LinePlotImageDisplaySetDoAutoSurvey( 0, 0 )

DISP.LinePlotImageDisplaySetGroupToDisplayTransform(SLICEID,0,0.05,0,0.001)
```

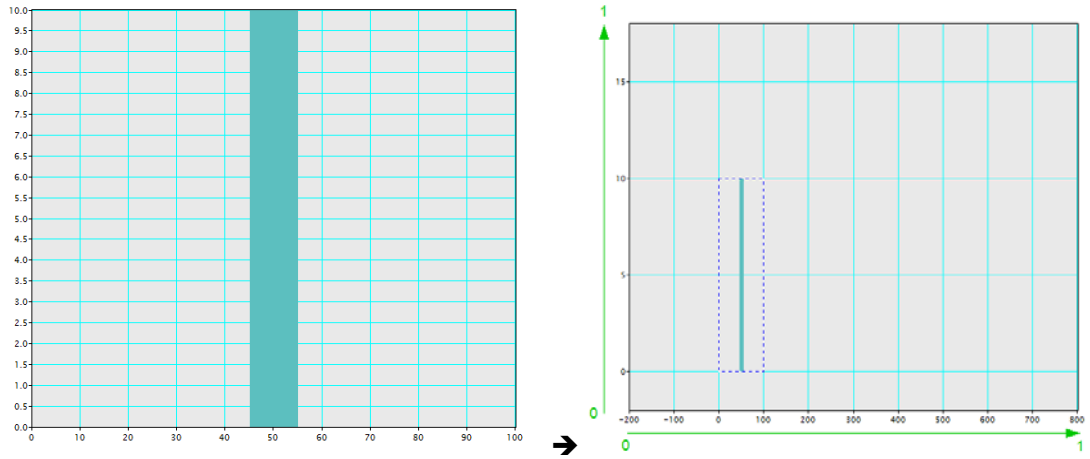
Note, that we had to turn off the LinePlot AutoSurvey of the Display first, else all changes would be overridden automatically. The outcome of this script is:



We have "squeezed" our group 10 times more in X direction and 2 times more in I direction. In other words, the coordinates of our group [0,0,100,10] has turned into [0,0,0.1,0.5] as indicated by the blue dashed box.
Note, that I used the specific notation of a box as [X_{min}, I_{min}, X_{max}, I_{max}] with 0/0 being the bottom-left corner as indicated by the green arrows!

Now this is just the scaling, what about the offset? We use the same script as above, but with the last line (the transformation) adapted like the following: We keep the scaling (0.001x0.05) but add an offset of (+0.2 x +0.1):

```
(…)
DISP.LinePlotImageDisplaySetGroupToDisplayTransform(SLICEID,0.1,0.05,0.2,0.001)
```



What happened? Following the calibration equation

$$V_{cal} = V_{ucal} \cdot scale + offset$$

We have mapped our group [0,0,100,10] into the new (view) coordinates [0.2,0.1,0.3,0.6] as indicated by the blue dashed box.

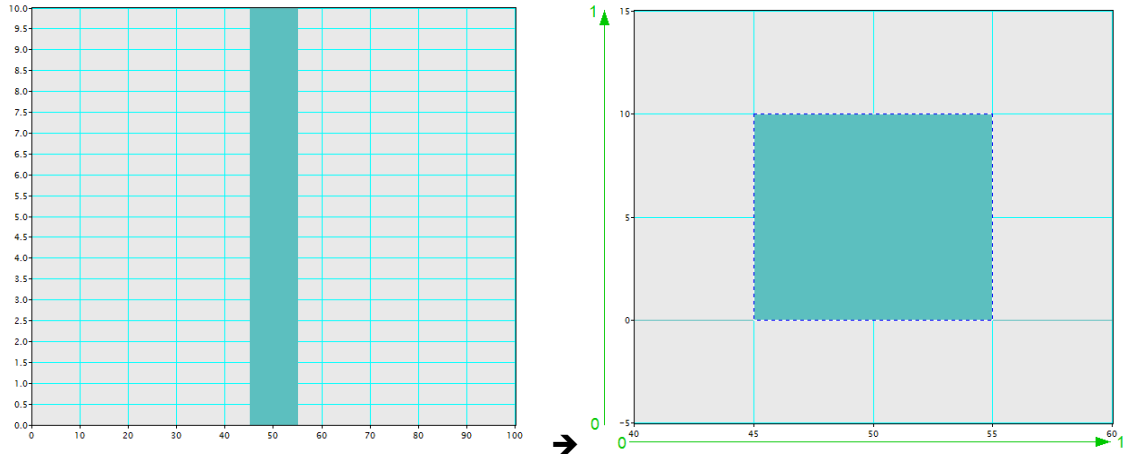
So far, we have only “squeezed” the graph into our display, of course it is also possible to enlarge it, as shown in the next example. This time, we do like to set the axis of the display to defined values. We would like to have the x-axis go from 40 to 60, and the I-axis from -5 to 15. This involves some “inverse” calculation though.

We have to solve the two equations:

I:	$0 = 40 * S_x + O_x$	I:	$0 = -5 * S_I + O_I$
II:	$1 = 60 * S_x + O_x$	II:	$1 = 15* S_I + O_I$
	➔ $S_x = 0.05$; $O_x = -2$		➔ $S_I = 0.05$; $O_I = 0.25$

This bringsd the proper transformation line:

```
(...)
DISP.LinePlotImageDisplaySetGroupToDisplayTransform(SLICEID,0.25,0.05,-2,0.05)
```



If one likes the display to show the range from *minimum channel* C_{min} to *maximum channel* C_{max},
one has to calculate the *scale* S and *offset* O of the necessary transformation for both intensity and x-axes by

$$S = 1 / (C_{max} - C_{min}) \text{ and } O = -C_{min} / (C_{max} - C_{min})$$

and then set the Group-To-Display transformation of the lineplot with the command:

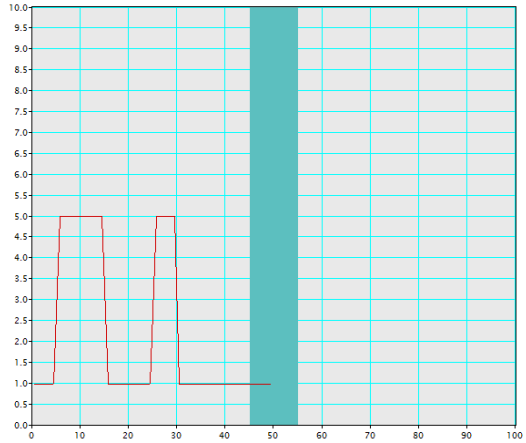
```
DISP.LinePlotImageDisplaySetGroupToDisplayTransform(SLICEID,O_I,S_I,O_x,S_x)
```

Next, we have to look into more detail of multiple slices in a lineplot. As stated at the beginning, all slices are grouped into one group which is connected to the display by the *group-to-display* transformation (GCS → DCS). Each of the slices has a image coordinate system (ICS) and the *image-to-group* transformation (ICS → GCS) allows adjusting the slices relative to the group (and therefore each other).

Lets create a lineplot containing two slices by the following script:

```
image IMG,IMG2
imagedisplay DISP
object SLICEID 0,SLICEID 1

IMG := RealImage("Slice0",4,100,1)
IMG = 0
IMG[0,45,1,55]=10
IMG.showimage()
DISP = IMG.ImageGetImageDisplay(0)
DISP.LinePlotImageDisplaySetDoAutoSurvey( 0, 0 )
SLICEID 0 = DISP.ImageDisplayGetSliceIDByIndex(0)


IMG2 := RealImage("Slice1",4,50,1)
IMG2 = 1
IMG2[0,5,1,15]=5
IMG2[0,25,1,30]=5

SLICEID_1 = DISP.ImageDisplayAddImage(IMG2,"Slice1")
```

Naturally, each added slice is added to the group by the transformation of scale 1 and offset 0, which means that channels of all slices are overlapping 1:1.



It is possible to read the image-to-group transformations parameters by the following script commands:

```
(…)
number S0Gxs,S0Gxo,S0Gis,S0Gio
number S1Gxs,S1Gxo,S1Gis,S1Gio
number GDxs,GDxo,GDis,GDio

DISP.LinePlotImageDisplayGetGroupToDisplayTransform(SLICEID 1,GDio,GDis,GDxo,GDxs)
result("\n\n GROUP TO DISPLAY TRANSFORM yields:")
result("\n\t x: scale = "+GDxs+"\t offset:"+GDxo)
result("\n\t I: scale = "+GDis+"\t offset:"+GDio)

disp.LinePlotImageDisplayGetImageToGroupTransform(SLICEID 0,SLICEID 0,S0Gio,S0Gis,S0Gxo,S0Gxs)
disp.LinePlotImageDisplayGetImageToGroupTransform(SLICEID 1,SLICEID 1,S1Gio,S1Gis,S1Gxo,S1Gxs)
result("\n\n IMAGE TO GROUP TRANSFORM yields:")
result("\n\t SLICE 0 -x: scale = "+S0Gxs+"\t offset:"+S0Gxo)
result("\n\t SLICE 0 -I: scale = "+S0Gis+"\t offset:"+S0Gio)
result("\n\t SLICE 1 -x: scale = "+S1Gxs+"\t offset:"+S1Gxo)
result("\n\t SLICE 1 -I: scale = "+S1Gis+"\t offset:"+S1Gio)
```

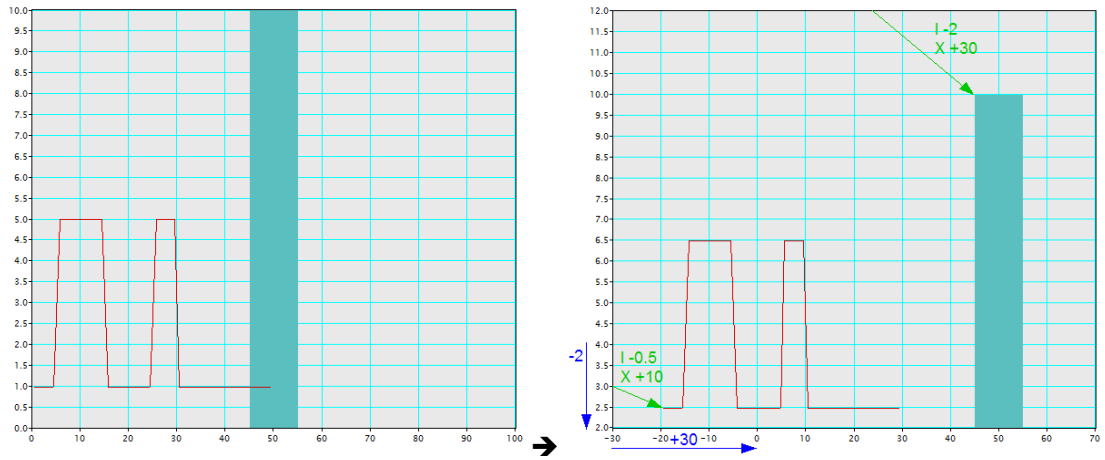
This results in the output:

```
GROUP TO DISPLAY TRANSFORM yields:
      x: scale = 0.01        offset:0
      I: scale = 0.1         offset:0

 IMAGE TO GROUP TRANSFORM yields:
      SLICE 0 -x: scale = 1   offset:0
      SLICE 0 -I: scale = 1   offset:0
      SLICE 1 -x: scale = 1   offset:0
      SLICE 1 -I: scale = 1   offset:0
```

Note, that the ImageToGroupTransform commands need two slice_ids. The first one is the actual slice, the second is the reference slice (which is not used).

We can now shift the slice1 relative to the group by an offset of +10 on X-axis and -0.5 on the I-axis and at the same time shift the slice0 relative to the group by an offset of +30 on X-axis and -2 on the I-axis:

```
(…)
disp.LinePlotImageDisplaySetImageToGroupTransform(SLICEID 1,SLICEID 1,-0.5,1,10,1)
disp.LinePlotImageDisplaySetImageToGroupTransform(SLICEID_0,SLICEID_0,-2,1,30,1)
```




```
GROUP TO DISPLAY TRANSFORM yields:
      x: scale = 0.01        offset:0
      I: scale = 0.1         offset:0

 IMAGE TO GROUP TRANSFORM yields:
      SLICE 0 -x: scale = 1   offset:30
      SLICE 0 -I: scale = 1   offset:-2
      SLICE 1 -x: scale = 1   offset:10
      SLICE 1 -I: scale = 1   offset:-0.5
```

What happened? Both graphs have been shifted as defined (see green arrows) with respect to group and therefore also with respect to the display. However, the scales on the axis have changed, too (blue arrows)!
Why is this happening? Because the labels of the axis are always taken from one slice called *CalibrationSlice*, which by default is slice0. (We will later learn more about that one.) Shifting the *CalibrationSlice* relative to the group therefore also shifts the axis-labels accordingly.

As this additionally complicates how the g*roup-to-display* transforms should be chosen to get a specific axis labelling, it is usually a good idea to keep the *CalibrationSlice* coordinate system and the group-system the same, i.e. having the *image-to-group* transform with scale 1 and offset 0 for this slice. However, if for some reason this is not possible, one has to expand the equations given before by the following:

If one likes the display to show the range from *minimum channel* C_{min} to *maximum channel* C_{max}, one has to calculate the *scale* S and *offset* O of the necessary transformation for both intensity and x-axes by

$$S = 1 \Big/ \left(C'_{max} - C'_{min}\right) \text{ and } O = -C'_{min} \Big/ \left(C'_{max} - C'_{min}\right)$$

with

$$C'_{min/max} = C_{min/max} \cdot Scale_{ICS \to GCS} + Offset_{ICS \to GCS}$$

Then set the Group-To-Display transformation of the lineplot with the command:

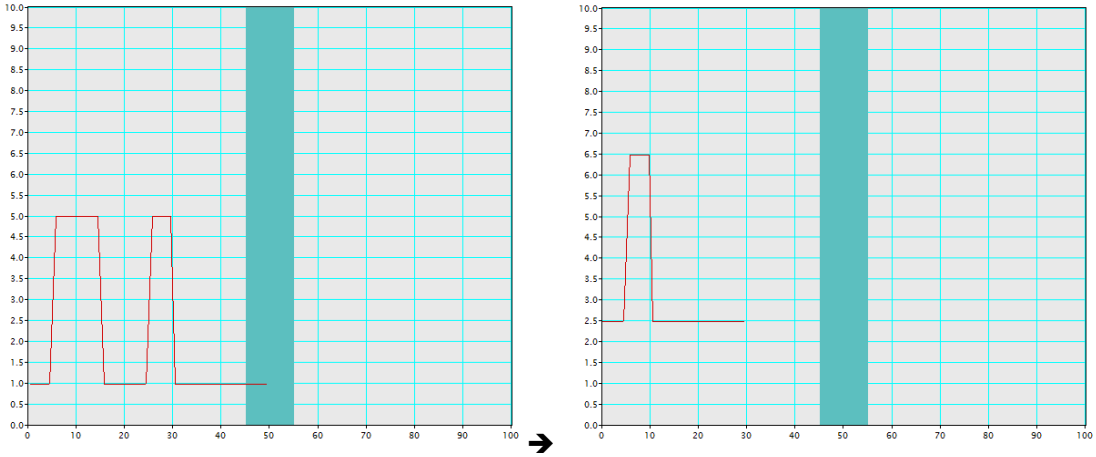
```
DISP.LinePlotImageDisplaySetGroupToDisplayTransform(SLICEID,O_I,S_I,O_x,S_x)
```

In other words: The axis-limits one would like to have need to be transferred from the image-coordinate-system (ICS) to the group-coordinate-system (GCS) first, before being used in the equations.

The following script will do the same shifts as before, but then adapts the display to have the same limits on the axis again.

```
(…)

disp.LinePlotImageDisplaySetImageToGroupTransform(SLICEID 1,SLICEID 1,-0.5,1,10,1)
disp.LinePlotImageDisplaySetImageToGroupTransform(SLICEID 0,SLICEID 0,-2,1,30,1)

(…)
number DiS,DiO,DxS,DxO
number Xmin = 0
number Xmax = 100
number Imin = 0
number Imax = 10

Xmin = Xmin*S0Gxs+S0Gxo
Xmax = Xmax*S0Gxs+S0Gxo
Imin = Imin*S0Gis+S0Gio
Imax = Imax*S0Gis+S0Gio


DiS = 1/(Imax-Imin)
DiO = Imin/(Imin-Imax)
DxS = 1/(Xmax-Xmin)
DxO = Xmin/(Xmin-Xmax)

DISP.LinePlotImageDisplaySetGroupToDisplayTransform(SLICEID 1,Dio,Dis,Dxo,Dxs)
result("\n\n GROUP TO DISPLAY TRANSFORM after adaptation yields:")
result("\n\t x: scale = "+Dxs+"\t offset:"+Dxo)
result("\n\t I: scale = "+Dis+"\t offset:"+Dio)
```

```
 GROUP TO DISPLAY TRANSFORM yields:
      x: scale = 0.01        offset:0
      I: scale = 0.1         offset:0

 IMAGE TO GROUP TRANSFORM yields:
      SLICE 0 -x: scale = 1   offset:30
      SLICE 0 -I: scale = 1   offset:-2
      SLICE 1 -x: scale = 1   offset:10
      SLICE 1 -I: scale = 1   offset:-0.5

 GROUP TO DISPLAY TRANSFORM after adaptation yields:
      x: scale = 0.01        offset:-0.3
      I: scale = 0.1         offset:0.2
```



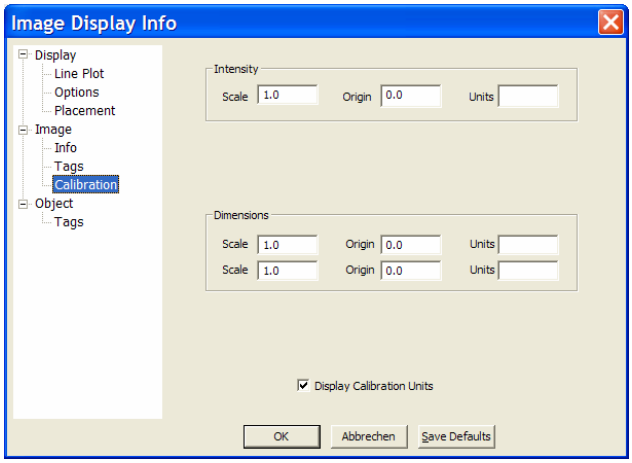
Note, that it *looks like* slice0 has not been shifted, but it has! Because both slices have been shifted, you can see the total shift of slice1 with respect to slice0 in the display, which is now +10-30 = -20 along x-axis and -0.5+2=-1.5 along I-axis. For simplicity, there also exists a command to get these relative transforms more directly. However, this command can only be used to *get* the transforms and not to *set* them to new values:

```
(…)
number S0S1io,S0S1is,S0S1xo,S0S1xs
disp.LinePlotImageDisplayGetImageToImageTransform(SLICEID 0,SLICEID 1,S0S1io,S0S1is,S0S1xo,S0S
1xs)
result("\n\n IMAGE TO IMAGE TRANSFORM (0->1):")
result("\n\t x: scale = "+S0S1xs+"\t offset:"+S0S1xo)
result("\n\t I: scale = "+S0S1is+"\t offset:"+S0S1io)
```

```
 IMAGE TO IMAGE TRANSFORM (0->1):
      x: scale = 1    offset:20
      I: scale = 1    offset:-1.5
```

Now that we know how we can put different slices, shift/scale them with respect to each other, and then display a certain value-range, it is time to consider how *calibration* fits into this.

Each slice in a lineplot can have its own individual calibrations for intensity and X-axis, which turns the uncalibrate values into calibrated ones along the display. These calibrations can be set using the image-display of the slice from the menu, or by using the according scripting commands:



- `ImageGet/SetIntensityOrigin(image IMG, number Origin)`
- `ImageGet/SetIntensityScale(image IMG, number Scale )`
- `ImageGet/SetIntensityUnitString(image IMG, string UnitString)`
- `Get/SetScale(image IMG, number ScaleX , number ScaleY)`
- `Get/SetOrigin(image IMG, number OriginX, number OriginY)`
- `Get/SetUnitString(image IMG, string UnitX, string UnitY)`

Note, that the calibrated values are calculated following a different equation[1]:

$$V_{cal} = (V_{ucal} - origin) \cdot scale$$

Though every slice can have its own calibration, only *one* calibration is used to turn the uncalibrated axes into the calibrated ones. Which one? The so called *CalibrationSlice*. It is possible to assign any of the slices to be the *CalibrationSlice* of the lineplot, and there exists of course also a command to get the slice-ID of the *CalibrationSlice*:

- `Object SLICE_ID = ImageDisplayGetCalibrationSlice( ImageDisplay DISP )`
- `ImageDisplaySetCalibrationSlice( ImageDisplay DISP, object SLICE_ID)`

[1] DM knows four different „calibration formats“.
These are:

Format 0: $V_{cal} = V_{ucal} \cdot scale + origin$

Format 1: $V_{cal} = (V_{ucal} - origin) \cdot scale$

Format 256: $V_{cal} = \left(V_{ucal} / scale \right) + origin$

Format 257: $V_{cal} = (V_{ucal} - origin) / scale$