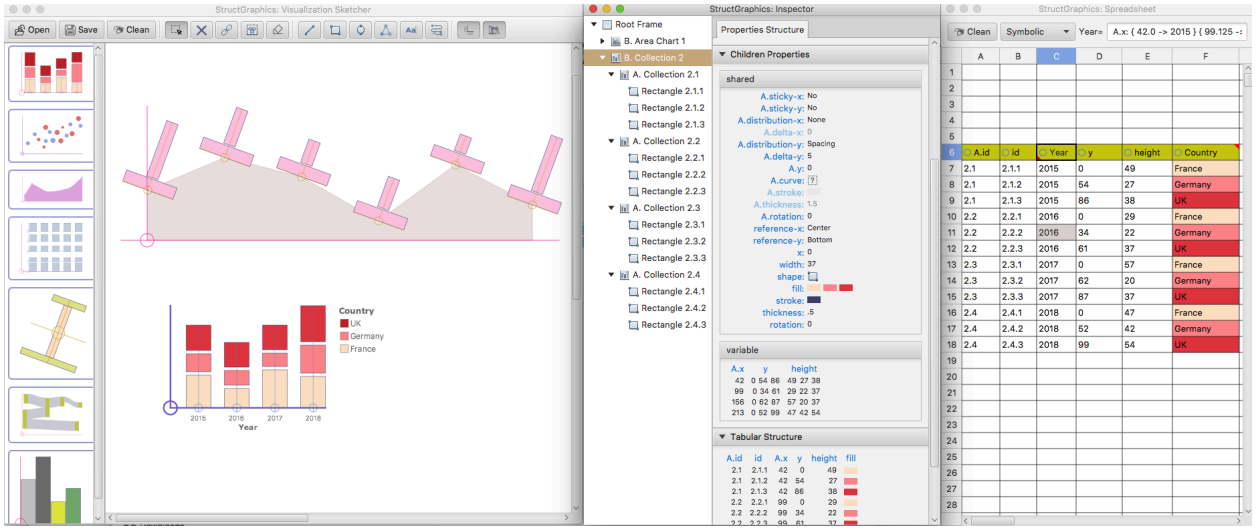
StructGraphics: Introductory Tutorial

Overview

StructGraphics is a desktop application for creating data visualizations without relying on any specific dataset. StructGraphics allows you to design the graphical elements (e.g., circles, rectangles, lines) of your visualizations as in common vector graphics editors, such as Adobe Illustrator, Sketch, Affinity Designer, Corel Designer, etc. Tables are generated automatically as you construct the structure of the graphics and the relationships of their graphical properties. Later, you can bring these tables into a *spreadsheet* to assign variable names, apply data transformations, and then visualize selected variables as legends, labels, and axes. Have in mind that this is a research prototype, missing some key functionality of commercial software, such as undo/redo operations and advanced graphics-editing tools.

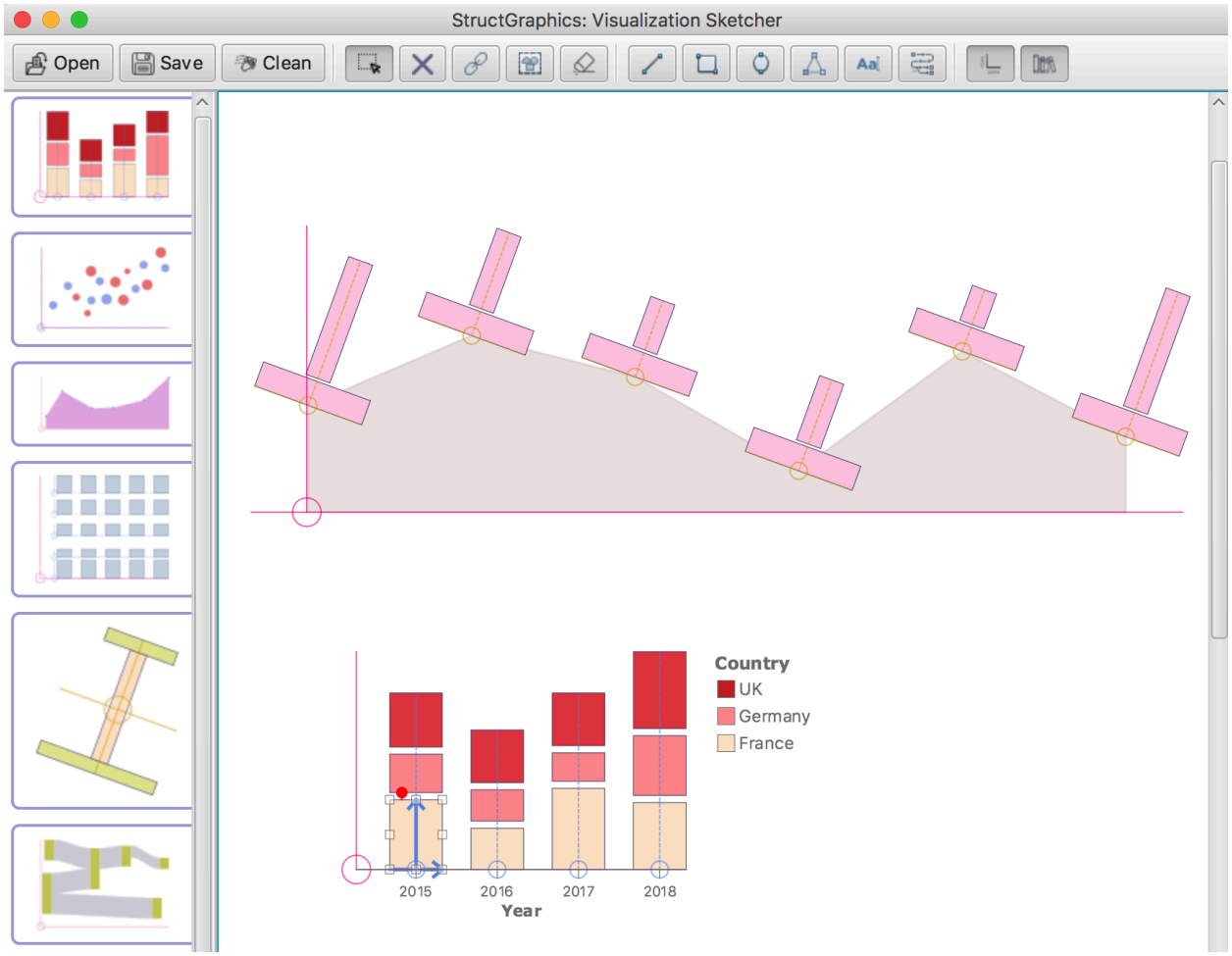
But let's first introduce the main features of StructGraphics. As you see on the screen, the user interface consists of three windows: (1) the visualization sketcher (left), (2) the properties inspector (center), and (3) a spreadsheet (right). At the beginning, we will ignore the spreadsheet and concentrate on the first two windows.



Drawing and Structuring Visualizations

Visualization Sketcher: This is where you draw visualizations. The large component of the sketcher (right) is the canvas. In the figure shown below, we have created a visualization that resembles an area chart (top) and a regular nested barchart (bottom).

The left smaller component of the window displays some miniature visualizations. This component serves as a visualization *library* where you can save a visualization (or part of a visualization hierarchy). Later, you can drag them from the library to the canvas and reuse them. We will now hide this library component to make some more space.



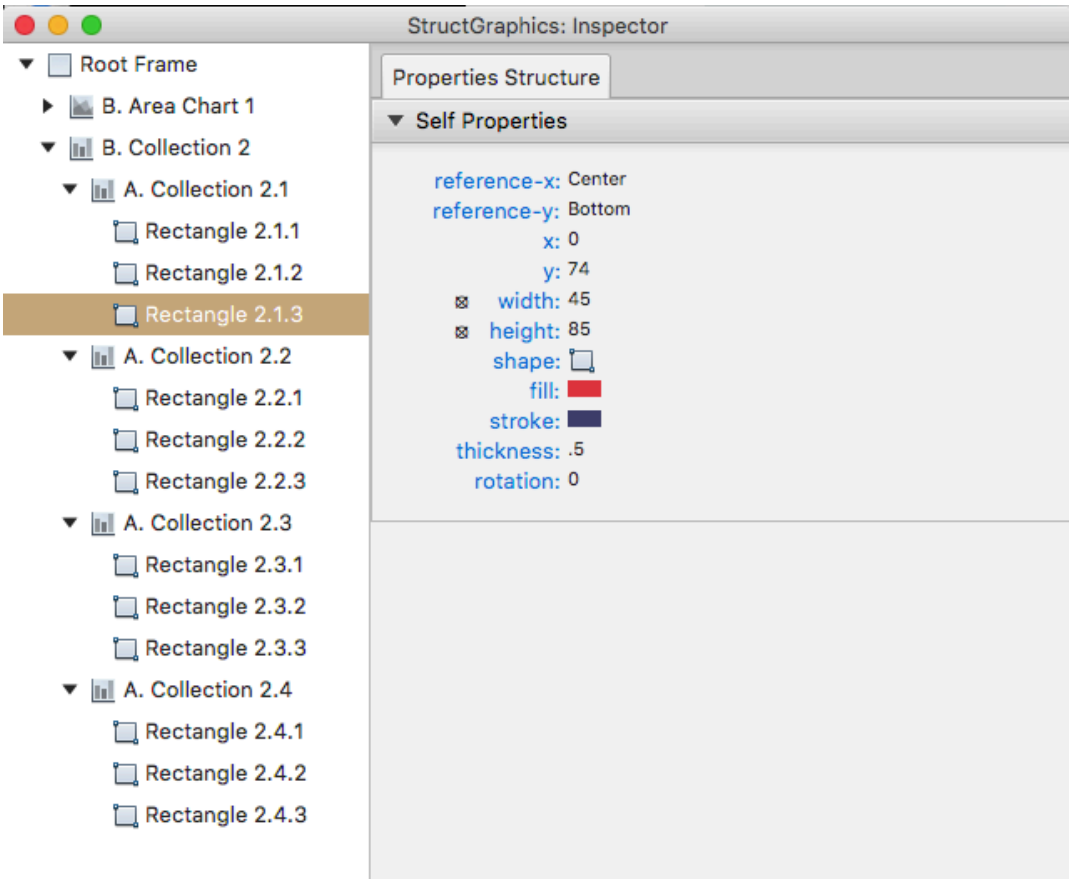
Inspector: The inspector shows the tree of a visualization hierarchy (left) but also exposes the graphical properties of its nodes, which can be shapes, collections, and groups (discussed later). Representative graphical properties are the *x* and *y* coordinates of a shape, its *width* and *height*, its *fill* and *stroke* color, etc. The inspector

allows you to edit their values. The *width* and *height* property can be locked together (by pressing on the ⊠ icon next to them) to change them proportionally. We will see later that the inspector also allows you to structure the graphical properties of collections, by creating structural *relationships* (bindings) between them.

Please, observe how the nodes in the tree are numbered:

1. The **numbers (e.g., 2.3.3)** are *unique identifiers* that show the position of a node in the hierarchy. Nodes with larger numbers appear on top of nodes with a lower number.

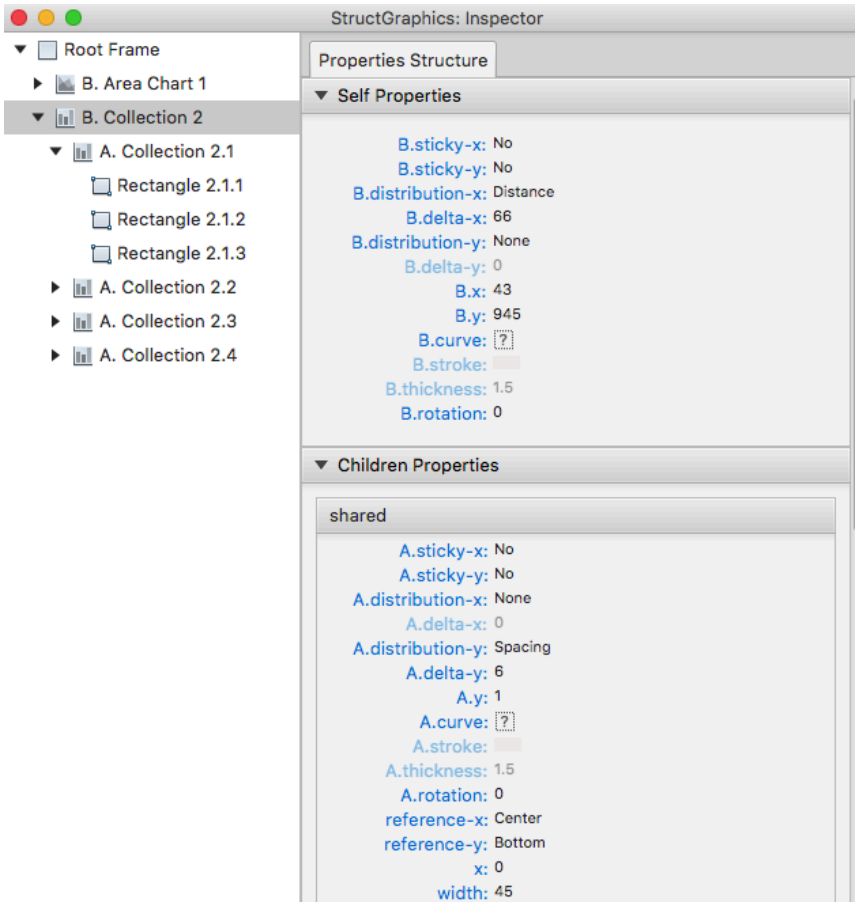
2. The **capital letters (e.g, A and B)** used as a prefix characterize the nesting level of a node. It is omitted for leaf nodes (shapes) and grows (starting from "A") as we move to a higher level in the visualization tree.



Familiarizing with the above notation is important. Property names may also have a character prefix that corresponds to the level of the nodes that they refer to. In the next figure below, you see the properties of a *B-level* collection (shown with a "B" prefix). But you also see the properties of its containing nodes in its subtree. The "A" prefix

corresponds to the properties of its children subcollections (A-level collections). The properties of leaf nodes (shapes) do not have a prefix.

Based on these conventions, you can distinguish between the properties of different nesting levels, e.g., between the coordinates of the shapes (*x* and *y*) and the coordinates of their parent collections (*A.x* and *A.y*). Note that the *x* and *y* coordinates of a node are defined with respect to their parent collection.



Creating Shapes (or Marks)

Line, Rectangle, Ellipse or *Circle, Triangle,* and *Textbox*

We will often refer to elementary shapes (leaf nodes in a visualization hierarchy) as marks. To **draw a mark**, you can use the drawing tools provided on the top of the window: 

You can then interact with the marks you created to move them or change their dimensions. You can also use the *replicate tool*  to create multiple copies of the

mark. Finally, you can apply additional operations through a contextual menu (right click).

Observe that when you draw or select a mark, an horizontal and vertical arrow appear. These arrows define an *x* and a *y* reference for measuring the position of a mark. The *x* reference takes the values *Left*, *Center*, and *Right*. The *y* reference takes the values *Bottom*, *Center*, and *Top*. Those properties can be modified from the inspector.

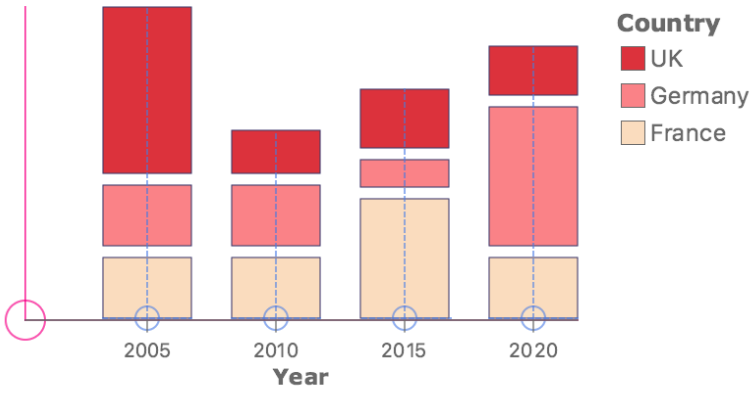
Arrows also show the direction of the marks: upwards and rightwards are *positive* directions, while downwards and leftwards are *negative* directions.

Creating Collections

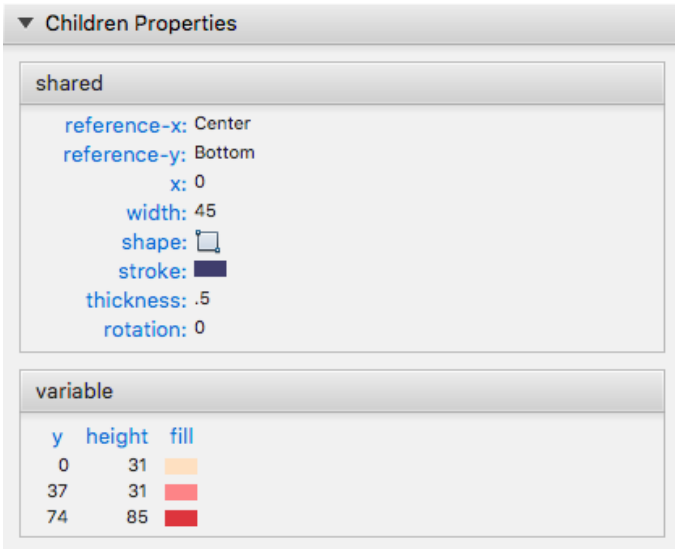
Collections are nodes that group multiple marks together. To create a collection, you need to use the *create collection* tool  and apply it to the marks of interest. A collection is visually represented by a Cartesian coordinate system, where a circle denotes its (0,0) point. The circle and the two orthogonal axes (*x* and *y*) can be used to select and interact with the collection. They are not part of the final visualization (but they define where final visualization axes will be displayed). Axes can be hidden (and shown back again) by clicking on the tool icon  .

Adding new marks to the collection. After a collection has been created, you can still use the replicate tool  to create multiple copies of a mark within the collection.

Nesting collections. You can create higher-level collections (collections of collections) by using the *create collection tool*. For example, the following barchart is a collection of four collections of rectangular shapes. The nested collections in this example align shapes vertically. The outer collection aligns them horizontally. Observe that the axes and circular handle of the nested collections appear in blue.



Property sharing. Within a collection, some of the properties of the containing nodes are common (thus *shared*), while others are *variable*. For example, in the above figure, the *shape* (rectangle), the *width*, and the *x* coordinates within each nested collection are shared by all containing marks. In contrast, the *fill color*, the *y* coordinate, and their *height* vary. These variable properties are the ones that are usually assigned to data variables (as we will see later).



The *property sharing* mechanism is central for a data visualization design. Click on a nested collection and check its *children properties* with the inspector. You can see that children properties are split between “shared” and “variable” properties. Edit and change a shared property. Then, change a variable property. What do you observe?

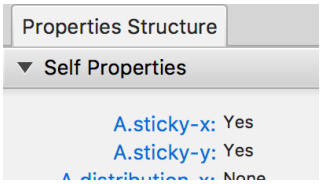
Grab the shape property and move it from the shared properties to the variable properties. As you see, you can now change individual shapes. Move the shape property back to the shared properties.

In a similar fashion, you can define the sharing of children properties at the higher level collection (see figure below). Notice that the *fill* property is shared at this level. The reason is that the same three colors are repeated across all subcollections. In contrast, the *height* property is variable at both hierarchy levels. Try to change the height of a rectangle -- as you see no other rectangles are affected. Take some time to play by moving properties from shared to variable, and vice versa, and change property values (either in the Sketcher or in the Inspector) to see how your changes affect the visualization design.

Distribution constraints. Move a subcollection in the Sketcher and then change the height of a rectangle. As you see, the *distance* or *spacing* between nodes is preserved. You can change the distance or spacing by using the red controls that appear when you select a rectangle or a collection.

This is what we call distribution constraints. These constraints are defined through the *distribution-x* and *distribution-y* properties of collections. Try to change these properties with the inspector and then see how your changes affect the behavior of the visualization layout.

Sticky constraints. Try to move a rectangle around. Then, select any subcollection and from the inspector, change the *A.stick-x* and *A.stick-y* properties to "Yes."

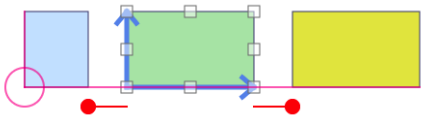


Try again to move a rectangle. What do you observe? You can use these properties to *stick* the nodes on the *x* or the *y* axis.

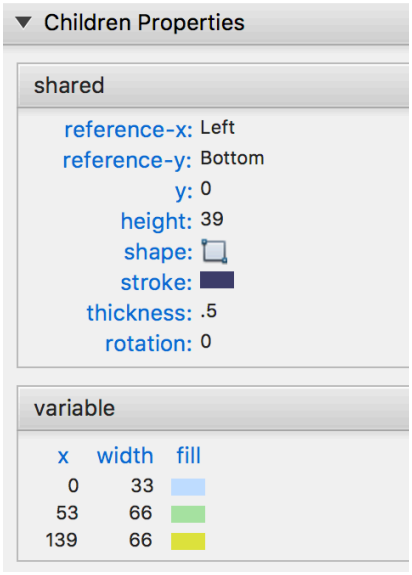
By example specification. The StructGraphics software supports a by-example approach for specifying the properties sharing structure. Let's see an example.

I start by drawing a mark (e.g., a rectangle). I then horizontally replicate this rectangle twice. I keep the height of all rectangles the same, but I make the width of the first rectangle smaller, while I make sure that the spacing between them is approximately the same. I also vary their fill colors. The result will be as follows:

Now, I use the *create collection tool* to group together into a collection. The resulting collection will be as follows:



And this is how its children properties are shared:



As you see, StructGraphics tries to automatically infer the structure of the properties of a collection based on how the graphical properties in the examples that you provide are either constant or vary. However, it may fail to correctly infer your goals so you may need to refine the properties through the inspector.

Generating Datasets

In contrast to traditional visualization systems, when working with StructGraphics, you do not need to start with an existing dataset. But you can follow the inverse direction and generate a dataset from the visualizations themselves. For this purpose, we will use the spreadsheet interface in combination with the property structures shown in the inspector.

As a first exercise, drag any property from the inspector and drop it into the spreadsheet, e.g., height of a rectangle. Then try to change its value from any of the three interfaces: the sketcher, the inspector, or the spreadsheet. You can see that all three user interfaces are fully synchronized.

Now, click on the top-level visualization collection and inspect its property structure (Inspector). Concentrate on the two tables that appear at the bottom: (1) the table of variable children properties, and (2) the *"Tabular Structure*" that includes all properties that are variable at least one level of the visualization hierarchy.

variable

A.x	y	height
62	0 37 74	31 31 85
128	0 37 74	31 31 22
193	0 67 87	61 14 30
258	0 37 114	31 71 25

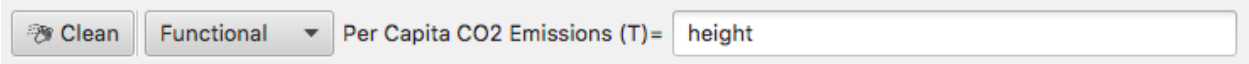
▼ Tabular Structure

id	A.x	y	height	fill
2.1.1	62	0	31	
2.1.2	62	37	31	
2.1.3	62	74	85	
2.2.1	128	0	31	
2.2.2	128	37	31	
2.2.3	128	74	22	
2.3.1	193	0	61	
2.3.2	193	67	14	
2.3.3	193	87	30	
2.4.1	258	0	31	
2.4.2	258	37	71	
2.4.3	258	114	25	

You can grab any of these two tables (or alternatively, individual columns), drag it, and then drop them into the spreadsheet. For this exercise, do this test with the second longer table. This is what you should get:

id	A.x	y	height	fill
2.1.1	62	0	31	0xffe6ccff
2.1.2	62	37	31	0xff9999ff
2.1.3	62	74	85	0xe64d4dff
2.2.1	128	0	31	0xffe6ccff
2.2.2	128	37	31	0xff9999ff
2.2.3	128	74	22	0xe64d4dff
2.3.1	193	0	61	0xffe6ccff
2.3.2	193	67	14	0xff9999ff
2.3.3	193	87	30	0xe64d4dff
2.4.1	258	0	31	0xffe6ccff
2.4.2	258	37	71	0xff9999ff
2.4.3	258	114	25	0xe64d4dff

You can treat columns as data variables and decide what each represents. Click on the *“height”* label and change its name, e.g., change it to “Per Capita CO2 Emissions (T)”. Observe that the area on top of the spreadsheet defines a “functional” mapping between the *height* property and your variable.



Edit the text field to enter a mathematical formula, such as:

Per Capita CO2 Emissions (T) = height/2

What do you observe?

Now click on the “fill” column label and change its name to “Country.” Then, use the drop-down list and change the type of the mapping from “functional” to “symbolic.”

What do you observe?

Try to change the textual values under the Country column to real country names. You can see that other values of the same color also change.

As a final step, right click on the Country label and choose “Show on Legend” from the menu. Also, right click on the “Per Capita CO2…” column and choose “Show on Axis.” You view on the Spreadsheet must like in the figure below:

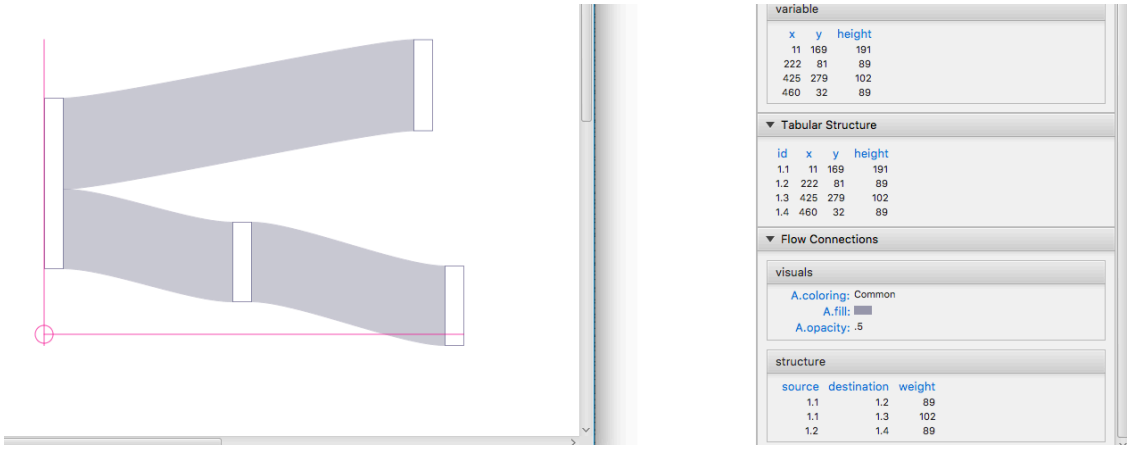
id	A.x	y	Per Capita CO2 E...	Country
2.1.1	62	0	16	Germany
2.1.2	62	38	16	France
2.1.3	62	76	23	USA
2.2.1	128	0	16	Germany
2.2.2	128	38	16	France
2.2.3	128	76	11	USA
2.3.1	193	0	30	Germany
2.3.2	193	66	7	France
2.3.3	193	86	15	USA
2.4.1	258	0	16	Germany
2.4.2	258	38	36	France
2.4.3	258	116	47	USA

Move to the window of the Sketcher and play with your visualization, e.g., try to change the height of the rectangles.

Now, think about which variable to assign to the *“A.x”* property that defines the *x* property of the subcollections and find a way to display it on the *x* axis of your visualization.

Other Visualization Designs

StructGraphics supports a range of other visualizations. Here, I will briefly show you how you can create connected graphs that look like *flow maps*.



You can create connections between shapes that are already within a collection by using the *draw connection* tool 

You can then review your connections with the inspector by looking at the “Flow Connections” section in the properties structure of the top visualization collection. As with other tables, you can drag this property structure to the spreadsheet.