

# SciDraw: A Quick-Start Guide to Make Level Schemes

Clark Casarella

June 2, 2016

## 1 Impetus For This Guide

The purpose of this tutorial is to provide a concise, straightforward method to begin making simple, publication quality level schemes for nuclear physics. This guide is *not* meant to replace M. Caprio's complete and exhaustive guide, examples, or built-in documentation delivered with the SciDraw distribution, but will provide the barebones knowledge on how to go from no level scheme to a non-zero number of level schemes in a quick fashion.





## 2 Installing SciDraw with Wolfram Mathematica 9.0+

Use of M. Caprio's SciDraw package for Mathematica assumes a few things: a) the user has downloaded and installed Wolfram Mathematica for his/her computer (a software license can be obtained from <http://oit.nd.edu>'s software download page for undergraduates, graduate students, and faculty), b) the user has downloaded the latest SciDraw distribution from M. Caprio's webpage at Notre Dame. (<http://scidraw.nd.edu/>) provides the web URL (as of April 29, 2016) for users that printed this tutorial and a hyperlink to the URL for individuals with this *pdf* distribution. On this page, there is a link to the latest release(s) of SciDraw, and download the latest .zip release (as of April 29, 2016, *SciDraw-0.0.7.zip*) to your local machine.

Next, we must tell Mathematica to look for SciDraw in a particular location to use this custom package set. Inside the .zip archive, there are several folders, all of which need to be extracted to one of two locations on your machine: the Mathematica Applications directory (where the installation files for Mathematica exist), or to any other (easily accessible) directory. Unzipping the archive to the Applications folder is generally the more difficult of the two options, especially if the current user does not have root access to view/edit hidden files associated with the Mathematica installation. My preferred method is obviously the latter of the two, pending ease of instruction across multiple Operating Systems.

With the archive unpacked *somewhere*, open a new Mathematica notebook and type:

```
$UserBaseDirectory
```

For those unfamiliar with the workings of Mathematica, any command in Mathematica is executed by pressing the SHIFT+ENTER/RETURN keys (  +  or  +  , depending on your keyboard layout). Execute this `$UserBaseDirectory` command, and Mathematica will produce an output that tells you the location of your Mathematica executable(s). This output is different for the user and Operating System, but as an example, my `$UserBaseDirectory` is:

```
/home/clarkc/Mathematica
```

Navigate to this directory and open the 'Kernel' directory; inside that, you will find a single file, '*init.m*'. Open this file with a text editor and you should see a mostly empty file, where you will need to specify the path to the SciDraw folder you extracted from the archive. To do this, type (anywhere in the *init.m* file):

```
AppendTo[$Path, "~ /path/to/SciDraw/folder/"];
```

Here, the "`~/path/to/SciDraw/folder`" should be the directory path to the SciDraw folder. **IMPORTANT NOTE:** this path must be the path that includes the folders "BlockOptions", "CustomTicks", "InheritOptions", among others. This must be in double quotations, and the formatting is somewhat dependent on

the user's Operating System. Mac OS and Linux users will use the standard forward slash when specifying a path ("`~/clarkc/Desktop/SciDraw/`"), while Windows users need to use a double backslash to separate folders ("`C:\\Users\\clarkc\\Downloads\\SciDraw`"). Save and close this *init.m* file, and restart Mathematica completely; Mathematica will now automatically look in the location to find SciDraw, which we must initialize before the user can start drawing.



**To summarize:**

1. Install Mathematica 9.0+
2. Download SciDraw and extract the contents to a location
3. Find the *init.m* file from the Mathematica Kernel with `$UserBaseDirectory`
4. Use the `AppendTo` command to point to the location of SciDraw on your machine
5. Save the *init.m* file, and restart Mathematica

## 2.1 Initializing SciDraw

SciDraw must be called into use by Mathematica before the user can start generating level schemes, and must only be done once for a particular session of Mathematica. Any STOP to the local computational kernel of Mathematica will also stop/close SciDraw, and must be re-loaded. Luckily, a kernel STOP is uncommon, meaning it is done manually, and is *generally never done by accident*. To load SciDraw in a Mathematica notebook, type:

```
<< SciDraw`
```

Take special care to not forget the  (the grave accent/tilde key above  on most keyboards) keystroke at the end of the command. Once this command is executed, you will see a small splash screen:

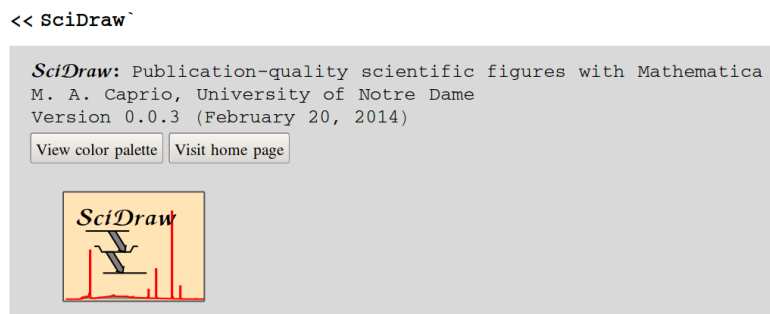


Figure 1: SciDraw splash screen inside Mathematica, indicating that the package has been loaded for use.

Once the package has been loaded (remember to insert this command at the top of *every* separate Level Scheme file you want to use), you are now ready to make level schemes!

## 3 Drawing Figures/Level Schemes in SciDraw

Included with this guide distribution, there is a Mathematica notebook to serve as a template, or minimally working example, for the user's convenience. The remainder of this guide serves to outline what each command in the template will do, with explanations on the syntax.

### 3.1 Preparing the Canvas and FigurePanel

Level schemes are drawn onto a Canvas of some set size, and in Mathematica, this is done by making a `Figure[ ];` which only needs the `CanvasSize→Fscale*{x,y}` argument to state how large the image will be, with  $F_{scale}$  being a final image scaling factor applied to the aspect ratio ( $\{x,y\}$ ). Inside the figure, we must then place at least one `FigurePanel`, as SciDraw can place multiple inset plots onto a canvas (this is an advanced user option, refer to M. Caprio's full documentation). Each `FigurePanel` needs a `PlotRange→{{xmin,xmax},{ymin,ymax}}`, the range over where the plot is displayed. To make a canvas that is 100% size and 4:3 aspect ratio with a single, empty plot from  $x \in \{0,1\}$  and  $y \in \{-10,100\}$ , type the text from Figure 2 into Mathematica.

```
Figure[
FigurePanel[
{

},
PlotRange → {{0, 1}, {-10, 100}}

],,
CanvasSize → 1.00*{4, 3}
]
```

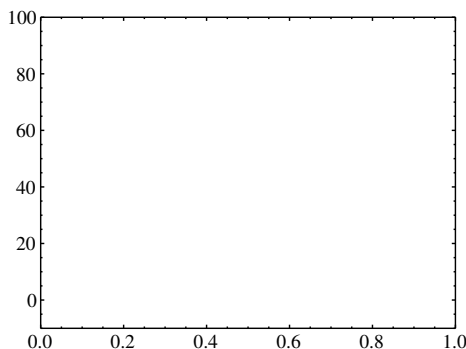




Figure 2: Blank SciDraw canvas. The ‘→’ is  followed by  in Mathematica.

**A note going forward:** All commands to make objects (levels, transitions, labels, *etc*) are placed in the ‘body’ of the `FigurePanel` (inside the curly brackets).

### 3.2 Adding Levels to the Figure

A level scheme without levels is just a scheme, no? We will now place a level, the fundamental object every command in the remainder of this guide will use, reference, or be anchored to. Levels require only a few inputs: a) a name to reference it by (“`LevelName`”), b) a position on the x-axis where the level begins ( $x_0$ ), c) a position on the x-axis where the level ends ( $x_1$ ), d) the position on the y-axis where the level lies ( $E_{lev}$ ), and optionally, e) any options to modify the level (*options*). Options for levels, transitions, and labels are discussed at the end of the document and are *always* optional. To draw a level, use the ‘Lev’ command

inside the FigurePanel:

$$\text{Lev}[\text{"Level\_Name"}][x_0, x_1, E_{\text{lev}}, \text{options}]; \quad (1)$$

A note to the user, SciDraw automatically places a margin of length 0.1 on every level. For example, a level drawn from  $x_0=0$  to  $x_1=1$  will really be drawn from 0.1 to 0.9; this is done to aid the user to allow sufficient room for small labels on the side with ease. The  $\llbracket$  and  $\rrbracket$  are done by simply typing two square brackets (  $\llbracket$   $\llbracket$  and  $\rrbracket$   $\rrbracket$  ). Adding:

$$\text{Lev}[\text{"162Dy_0gs"}][0, 1, 0];$$

Inside the  $\{ \}$  of `FigurePanel[ { }, ]`; will produce:

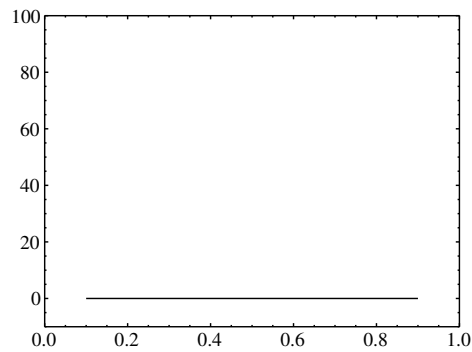


Figure 3: First level in SciDraw!

Granted, this all seems like a lot of work just to draw a single horizontal line at  $y=0$ , so let's add another level, (`Lev["162Dy_2gs"][0,1,80];`) to get the output in Figure 4:

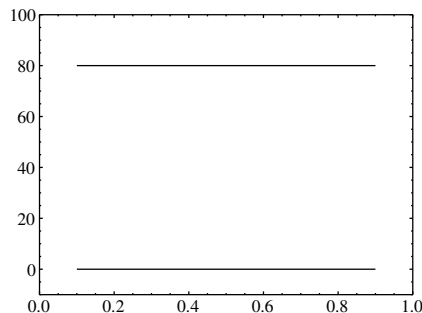


Figure 4: Second level being drawn in.

This is obviously not a pretty drawing, as it contains no label information or transitions, but a `Lev` is the basis of all drawing done in SciDraw. The user can add any number of levels, given they have a discrete, unique name.

### 3.3 Adding Transition Arrows to the Figure

Drawing a transition arrow between two levels is simple; the `Trans` command takes care of this, where the syntax is shown in equation 2:

$$\text{Trans}[\text{"Parent\_Name"}, x_0, \text{"Daughter\_Name"}, x_1, \text{options}]; \quad (2)$$

`"Parent_Name"` should be the name of the level you want to start from, similarly, `"Daughter_Name"` is the name of the level you want the transition arrow to end at.  $x_0$  is the position (relative to the start point of the level) at which the transition arrow should start on the parent level, with  $x_1$  being the end position on the daughter level (again, relative to the level).  $x_1$  can be a numerical value, or (in the vast majority of cases), one can use `Automatic` to have SciDraw drop the transition arrow straight down. Again, *options* clauses are purely optional, but will be discussed later. By adding:

### 3.4 Adding Text/Labels to the Figure

Most labels and text are added as *options* to either `Lev` or `Trans`; to add text, simply add:

$$\text{XLabel} \rightarrow \text{"text"} \quad (3)$$

Where any *options* would go. Here, the X in `XLabel` is one of five anchor points, `Left`, `Top`, `Right`, `Bottom`, or `Center`. `RightLabel` will make a label on the right side of the object to display whatever is in *text*, which can be plain-text input, or a spin-parity assignment. Mathematica inputs do not adhere to the same standard of sub/superscripts that you may be used to; M. Caprio has aided the creation of spin-parity labels via `LabelJP`. Typing `LeftLabel→LabelJP[2,+1]` as an *option* for `Lev` will place a  $2^+$  label on the left side of the named level. A standalone label (not as part of the *options* of a command) for use is the `BandLabel` command, which places a label under a named level at the midpoint of said level. The syntax for `BandLabel` is:

$$\text{BandLabel}[\text{"Level\_Name"}, \text{"text"}, \text{options}]; \quad (4)$$

```
Trans["162Dy_2gs",0.25,"162Dy_0gs",Automatic];
Trans["162Dy_2gs",0.50,"162Dy_0gs",0.75];
```

To the body of the `FigurePanel`, we produce:

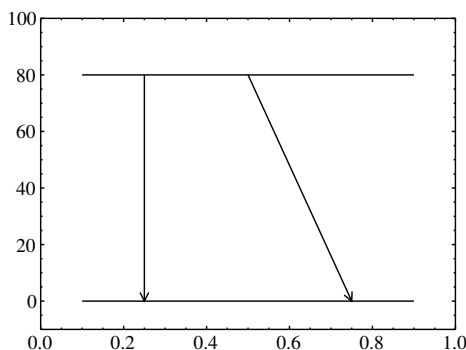


Figure 5: Adding transition arrows in (straight down and from point-to-point).

Again, text can be plain-text input, or, another built-in SciDraw function (again due to the input limitations of Mathematica and superscripts), `Isotope`. This is the easiest way to make atomic notation possible in SciDraw labels, by using:

```
Isotope["A", "Nuclear_Abbreviation"]
```

 (5)

In `Isotope`, take care not to forget the double quotations around each argument, where `A` is the atomic mass number for a particular `Nuclear_Abbreviation`. Changing font sizes of labels is also done as part of the *options* of a particular object via:

```
FontSize → VALUE
```

 (6)

Where `VALUE` is the numerical value (in printer point size) that you want the label's font to be displayed as. Refer to the included notebook walkthrough to see exact usages of labels, but a combination of all of the mentioned labels and options can be seen in figure 6:

### 3.5 Commonly Used Options

If the user grows tired of the default, simple line-style transition arrow, he/she can change this to be a thicker, block style transition arrow by using (7) as an *option* for `Trans`:

```
ArrowType → Block
```

 (7)

With this more robust arrow shape, the user can define several aspects of the `Block ArrowType`.

```
Width → VALUE
HeadLength → VALUE
```

All allow the user to change the width of the arrow, as well as the length of the head. Measured in an absolute, arbitrary unit of measure, both `Width` & `HeadLength` are defaulted to 5.

SciDraw tends to follow the design philosophy of Henry Ford, in that the default color for any object is black. To help guide the eye, the user/reader can change the color of any object via *options* on said object.

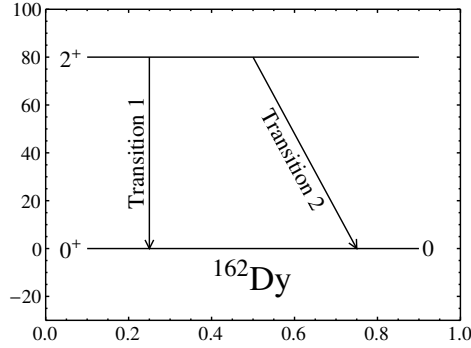


Figure 6: Labels drawn on levels & transitions, and Isotope notation used in a `BandLabel`. Note the increase in y-axis range to account for the `BandLabel`.

Back on the splash screen of 1, M. Caprio has included the entire range of colors available to use in SciDraw. Colors of individual objects can be modified with:

$$\text{XColor} \rightarrow \text{COLOR} \quad (8)$$

Similar to `XLabel`, the `X` in `XColor` can reference multiple different objects, `Font` (to change the color of font in labels for an object), `Line` (to change the color of bounding lines for the `Block ArrowType` or the color of a `Lev`), `Fill` (to change the interior color of the `Block ArrowType`, among others (fully listed in M. Caprio's documentation)). Following the `Color` thread, the background color of text in a label (in either a level or transition label) can be changed (to add emphasis or aid readability) with:

$$\text{TextBackground} \rightarrow \text{COLOR} \quad (9)$$

If the default thickness (1 arbitrary unit) for a particular line or level is too thin, the user can change this thickness via:

$$\text{LineThickness} \rightarrow \text{VALUE} \quad (10)$$

Oftentimes, the text from multiple labels will overlap with other objects; the text anchoring system of SciDraw helps the user in that

$$\text{XTextNudge} \rightarrow \{x, y\} \quad (11)$$

Where `X` *again* needs to reference the label object you want to nudge (`Left`, `Right`, `Center`, etc.), and that `{x,y}` specify the shift (in printer pts) horizontally to the right and vertically upward (take note that `x` & `y` can be negative). **A note from experience:** a great deal of time will be spent nudging labels to get precise placements and for small tweaks. Figure 7 is the output of many of the above options to change colors, font sizes, and arrow types. The user should refer to the included notebook to see the exact nature of each change provided.

### 3.6 Global Options and Frame Management

If one of the particular *options* need to be expressed in the majority of commands, the user can set global *options* by declaring (at the top of the `FigurePanel` body):

$$\text{SetOptions}[\text{COMMAND}, \text{options}]; \quad (12)$$

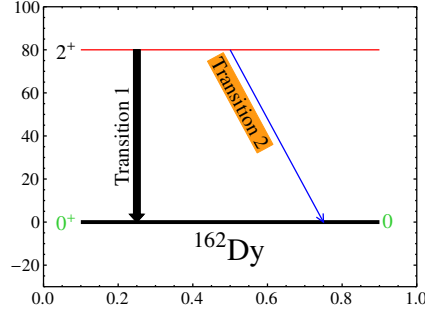


Figure 7: Various options being applied to levels, transitions, and labels

Any option that needs to be applied to all **COMMANDs** of a particular type go in *options*. For example, to make the font size 24 pt font for labels attached to a level, use `SetOptions[Lev,FontSize→24];`. While the y-axis is generally helpful to express the excitation energy of a level, oftentimes, the x-axis holds no use for a final drawing. It is sometimes beneficial to leave it in to troubleshoot level or transition placements, but we can remove the x axis from the frame with:

$$\text{XFrame} \rightarrow \text{False} \quad (13)$$

In this case, **X** does *not* refer to a particular object, but instead just to remove the x-axis (or y-axis by using `YFrame→False`). An axis label can be placed on the y-axis by invoking:

$$\text{YFrameLabel} \rightarrow \text{"text"} \quad (14)$$

These frame options are placed outside of the curly brackets of **FigurePanel**, just after **PlotRange**. Again, the supplementary notebook shows the exact usage of global options and the removal of the x-axis, but the final output can be seen in figure 8:

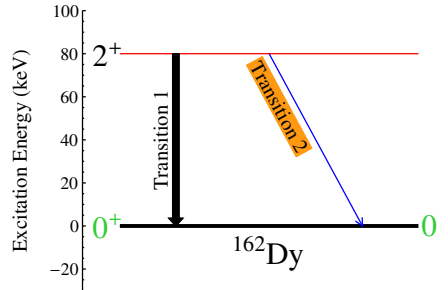


Figure 8: Global options applied to the font size on levels, a label placed on the y-axis, and the removal of the x-axis.



Given the above examples, a working template, and the overview of the most basic of SciDraw functions, one can make a publication-quality level scheme with a very small amount of work. Exporting the level scheme to an output file is simple: just right-click the image output and select “Save Graphic As...” to export to multiple image formats (.eps, .png, etc.). Happy drawing!

“We don’t make mistakes, we just have happy accidents.”

---

*Bob Ross*

The Joy of Painting (1983-1994)