**CSO Labeling using CSOLabelRenderer**

Introduction

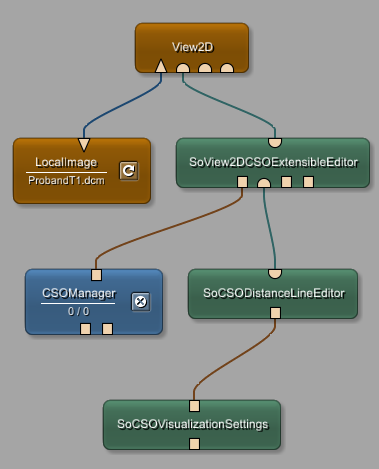
In this example, we explore different CSO types, specifically focusing on labeling using CSOLabelRenderer alongside Python scripting for precise communication of segmented regions in medical images.

Formularbeginn

Steps to do

Develop your network

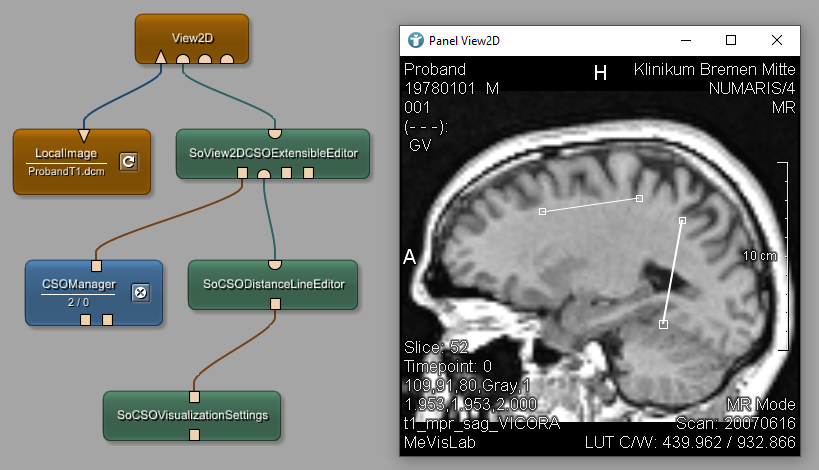
Add the following modules to your workspace and connect them as shown. Load the example DICOM image *ProbandT1.dcm.*

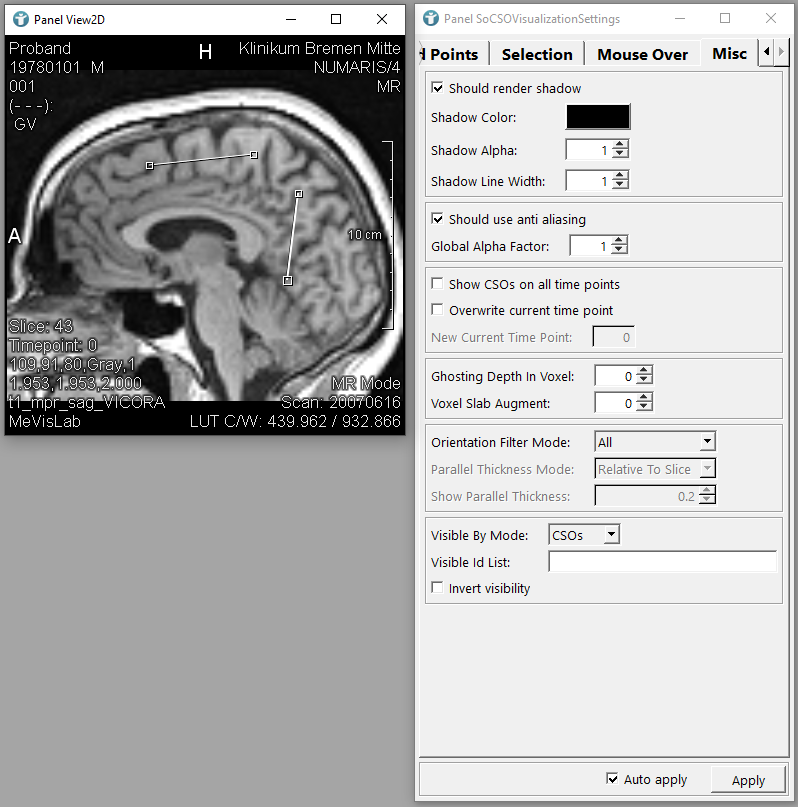


The ‘SoView2DCSOExtensibleEditor’ is like a control center for managing how CSOs are displayed. It helps us visualize CSOs in a 2D viewer, making it easy to create, edit, and view them.

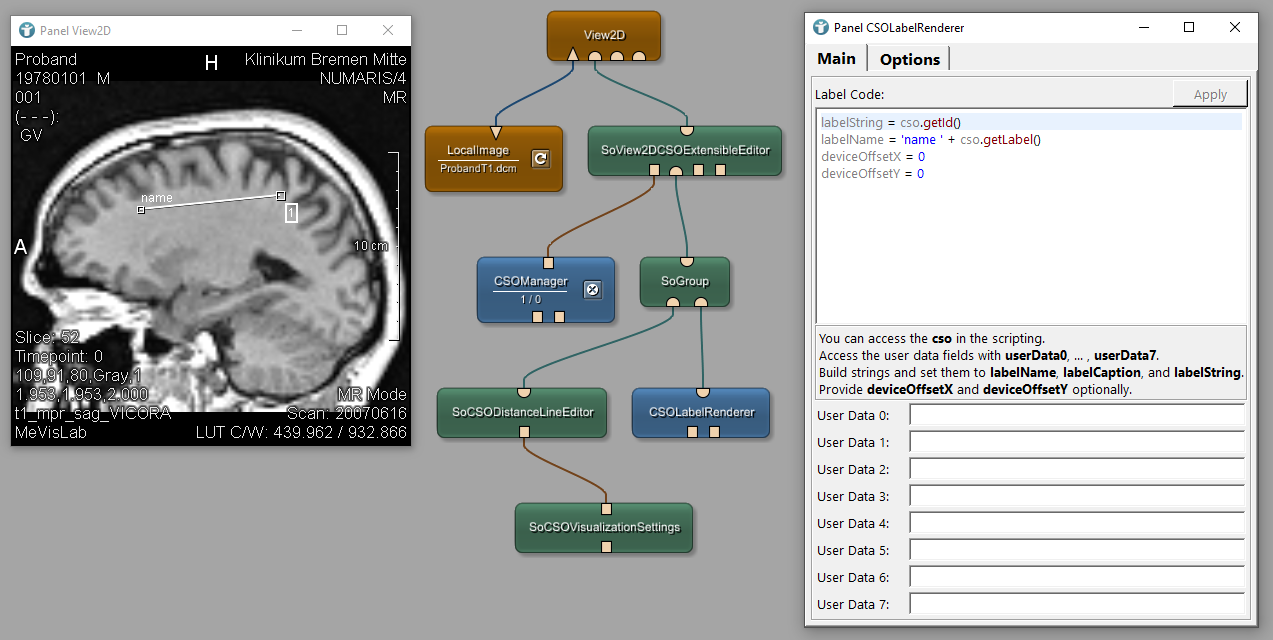
The ’SoCSODistanceLineEditor’ is used specifically for creating and interacting with CSOs that represent single line segments. It's handy for measuring distances and offers features like snapping to specific angles for accuracy.

By using these modules, we can effectively work with CSOs in our workspace. Visual properties, such as border width and style, shadow rendering, anti-aliasing, and optional border rendering, can be adjusted with the attached ‘SoCSOVisualizationSettings’ module.



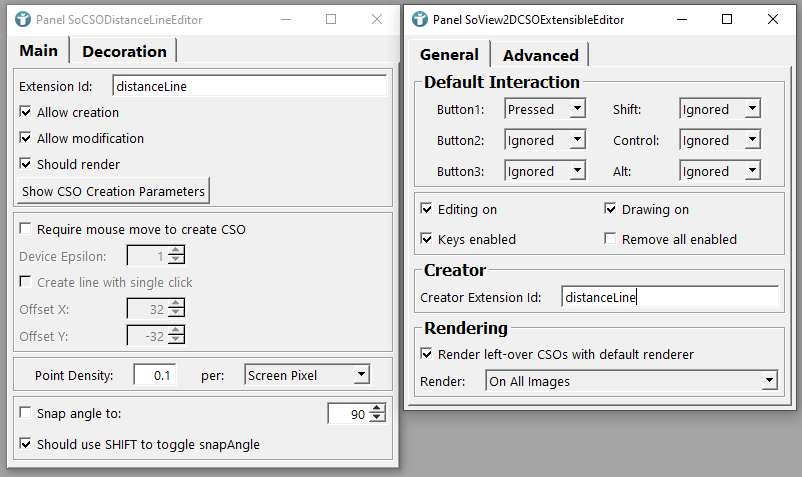
To customize visual properties like border width, style, shadow rendering, anti-aliasing, and optional border rendering, use the attached ‘SoCSOVisualizationSettings’ module. Check the field 'Should render shadow' to enable shadow rendering for CSOs. When this option is selected, CSOs will be displayed with a shadow effect, enhancing their visual presentation

Connect the ‘CSOLabelRenderer’ module to the ‘SoGroup ‘module, and then link the ‘SoGroup’ to the ‘SoView2DCSOExtensibleEditor’. By doing so, you'll notice that labels are now directly visible in the ‘View2D’.



The ‘CSOLabelRenderer‘ module is responsible for rendering labels for CSOs in the medical imaging context. These labels can be configured using Python scripting.

To ensure synchronization between modules and specify the active editor for creating \*distanceline\* CSOs, we need to set the same \*Extension Id\* in both the 'SoCSODistanceLineEditor' and 'SoView2DCSOExtensibleEditor' modules. In the 'SoView2DCSOExtensibleEditor' module, navigate to the \*General\* tab and locate the \*Creator Extension Id\* field. Here, input the same Extension ID used in the 'SoCSODistanceLineEditor' module. This ensures proper coordination and enables the creation of new \*distanceline\* CSOs using the designated editor.



When attached to a ‘SoView2DCSOExtensibleEditor’, the ‘CSOLabelRenderer’ module allows access to the currently rendered CSO. Python scripting is used to configure the labels and to define the properties of the label displayed for each CSO in the viewer. \*labelString\* is set to the unique ID of each CSO, while \*labelName\* combines the word \*name\* with the label assigned to the CSO. \*deviceOffsetX\* and \*deviceOffsetY\* determine the label's position relative to the CSO, both set to 0 for direct overlay. The labeling displaying the number 1 represent a default label because no specific name is provided for the CSO.

Replace the default \*Label Code\* with the following script:

if cso.getSubType() == 'distanceLine':

labelString = 'Length: ' + str(cso.getLength())

labelName = 'Distance '

else:

labelString = ''

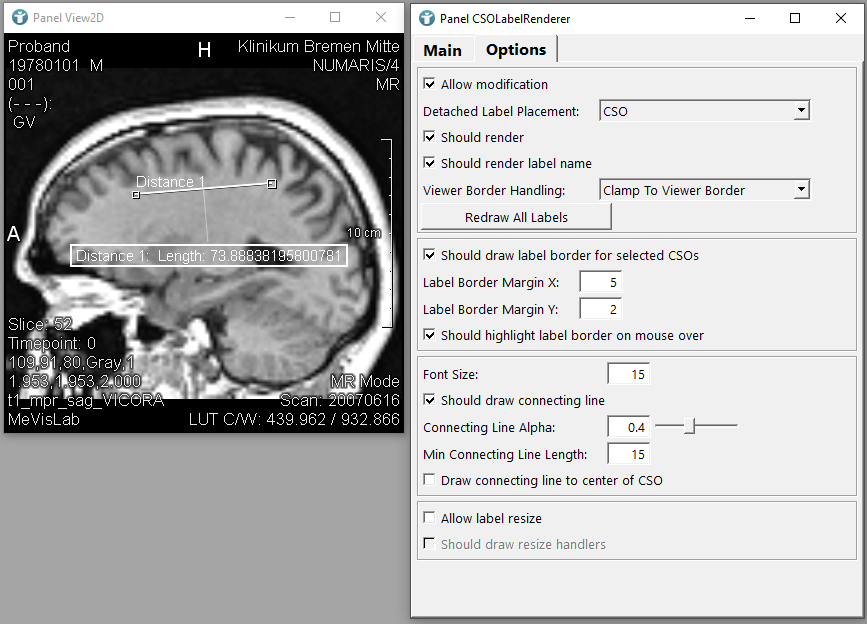
labelName = ''

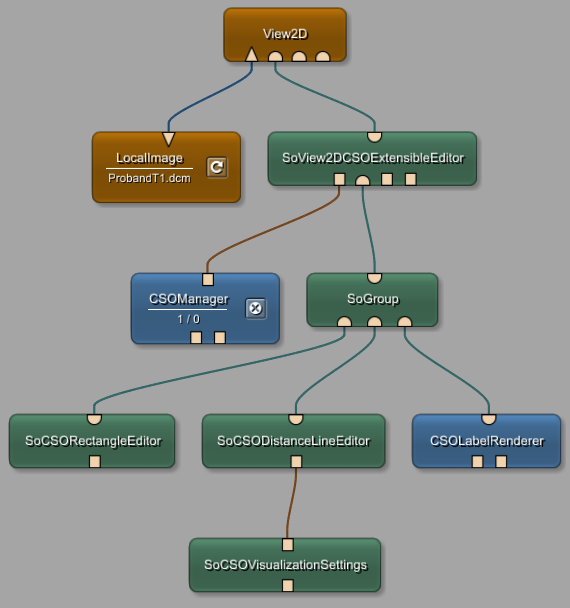
labelName += str(cso.getId())

labelCaption = labelName + ": "

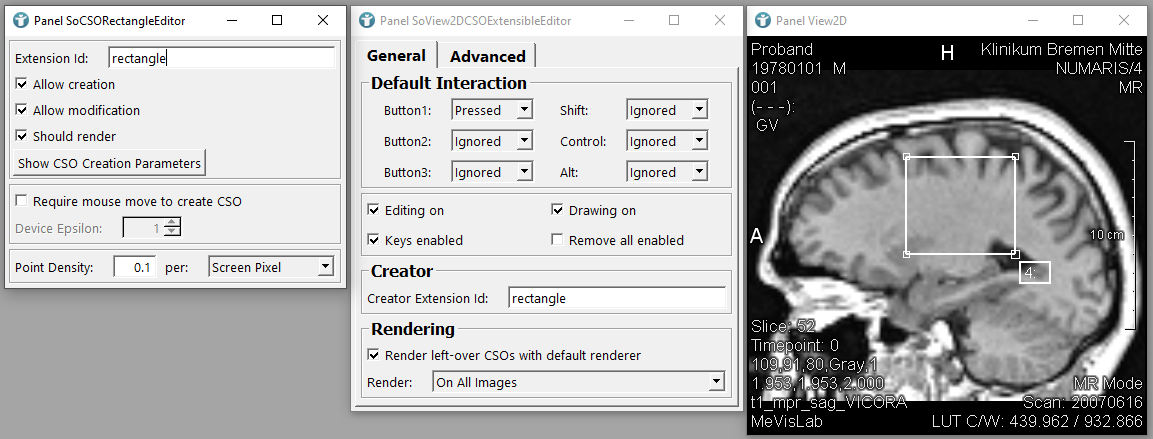
This script will ensure that the label displays the length of the line for distanceLine CSOs and remains empty for other CSO types.

In the 'CSOLabelRenderer' panel, head to the \*Options\* tab. Here, you can fine-tune various settings to enhance the appearance of CSO labels. Set the \*Label Border Margin X\* to \*5\* to create a margin around selected CSOs, providing a neat visual separation. Increase the \*Font Size\* to 15 for larger and more readable text, ensuring clarity and visibility. Additionally, adjust the \*Min Connecting Line Length\* to \*15\* to specify a minimum distance for drawing connecting lines, maintaining a clean and organized layout.



 Now, let's proceed with another type of CSO and follow similar steps. Add the 'SoCSORectangleEditor' module to your workspace and connect it to the 'SoGroup' module..

Ensure to update the \**Creator Extension Id\** field in the ‘SoView2DCSOExtensibleEditor‘ module with the\* *Extension Id\** of the ‘SoCSOEllipseEditor‘ module.



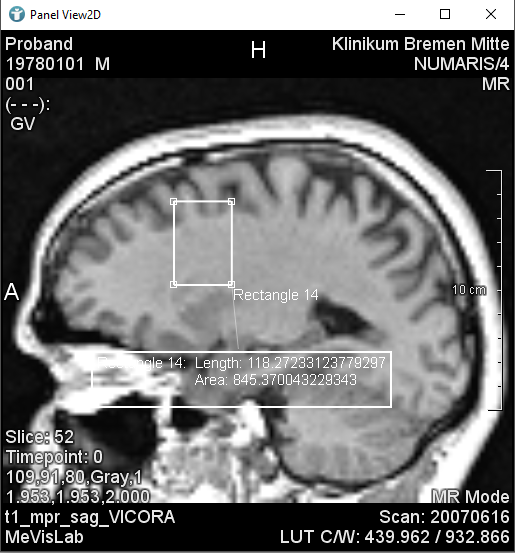
Open the ‘CSOLabelRenderer‘ panel and add the same script as for the \*distanceLine\* subtype, but include \*Area\* as well, since we are dealing with a rectangle.

elif cso.getSubType() == 'rectangle':

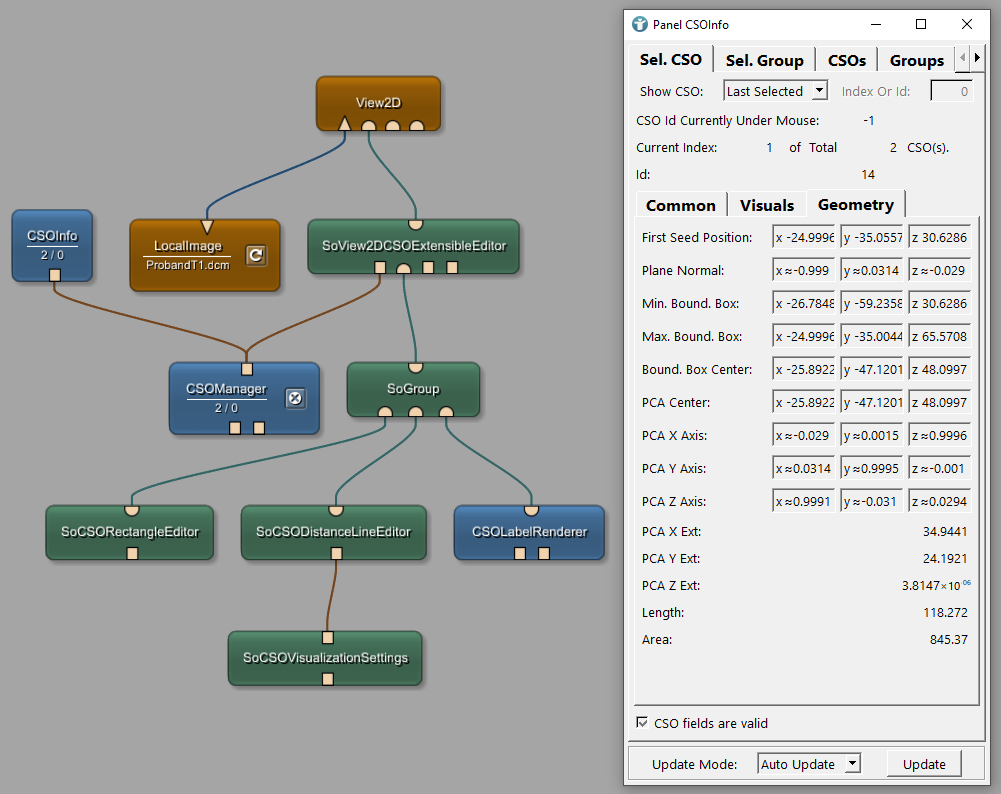
labelString = str('Length: ') + str(cso.getLength()) + '\n'

labelString += str('Area: ') + str(cso.getArea()) + '\n'

labelName = 'Rectangle '

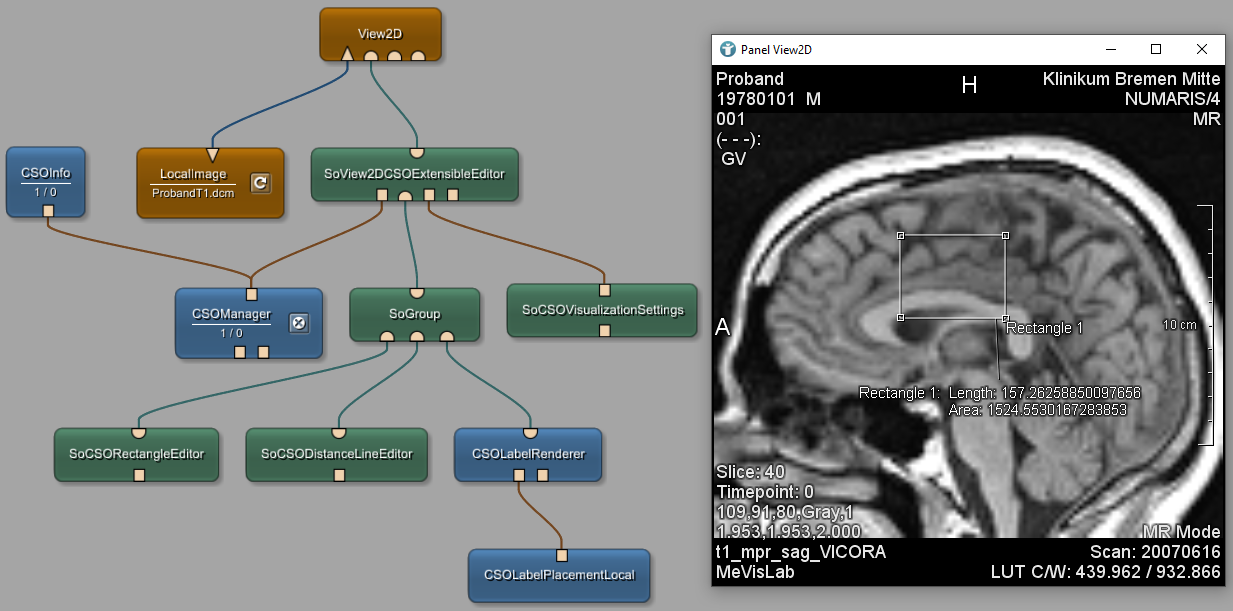


**Note: The 'Length' attribute in the context of rectangles represents the perimeter of the rectangle, calculated as 2a + 2b, where 'a' and 'b' are the lengths of the two sides of the rectangle**

Additionally, you can connect the 'CSOInfo' module to the 'CSOManager' module in your network. Then, navigate to the 'Geometry' tab to access detailed information about the width (\*PCA X EXT\*) and height (\*PCA Y EXT\*) of the rectangles.

When dealing with an ellipse, the "length" parameter typically refers to the perimeter or circumference of the ellipse, rather than its width or height. Since an ellipse doesn't have distinct edges like a rectangle, its "length" represents the distance around its curved boundary. It's calculated as 2 times the sum of the major axis (a) and the minor axis (b) of the ellipse, often denoted as 2πa + 2πb. This perimeter value provides a measure of the total length of the boundary of the ellipse.

Connect the ‘SoCSOVisualizationSettings‘ module to the ‘SoView2DCSOExtensibleEditor‘ module to ensure that the shadow effect is applied to all CSO subtypes you use



To define label placement, connect a ‘CSOLabelPlacementLocal‘ module to your ‘CSOLaberRenderer‘. This offers three positioning options: \*Seed Point\*, where labels align with the right-most seed point; \*Path Point\*, aligning with the right-most path point; or \*Seed or Path Point\*, which considers both for optimal placement.

