Tutorial

on 3D Slicer python scripting and programming

Laboratory for Percutaneous Surgery

Queen's University

Agenda

Part 1

- Software architecture

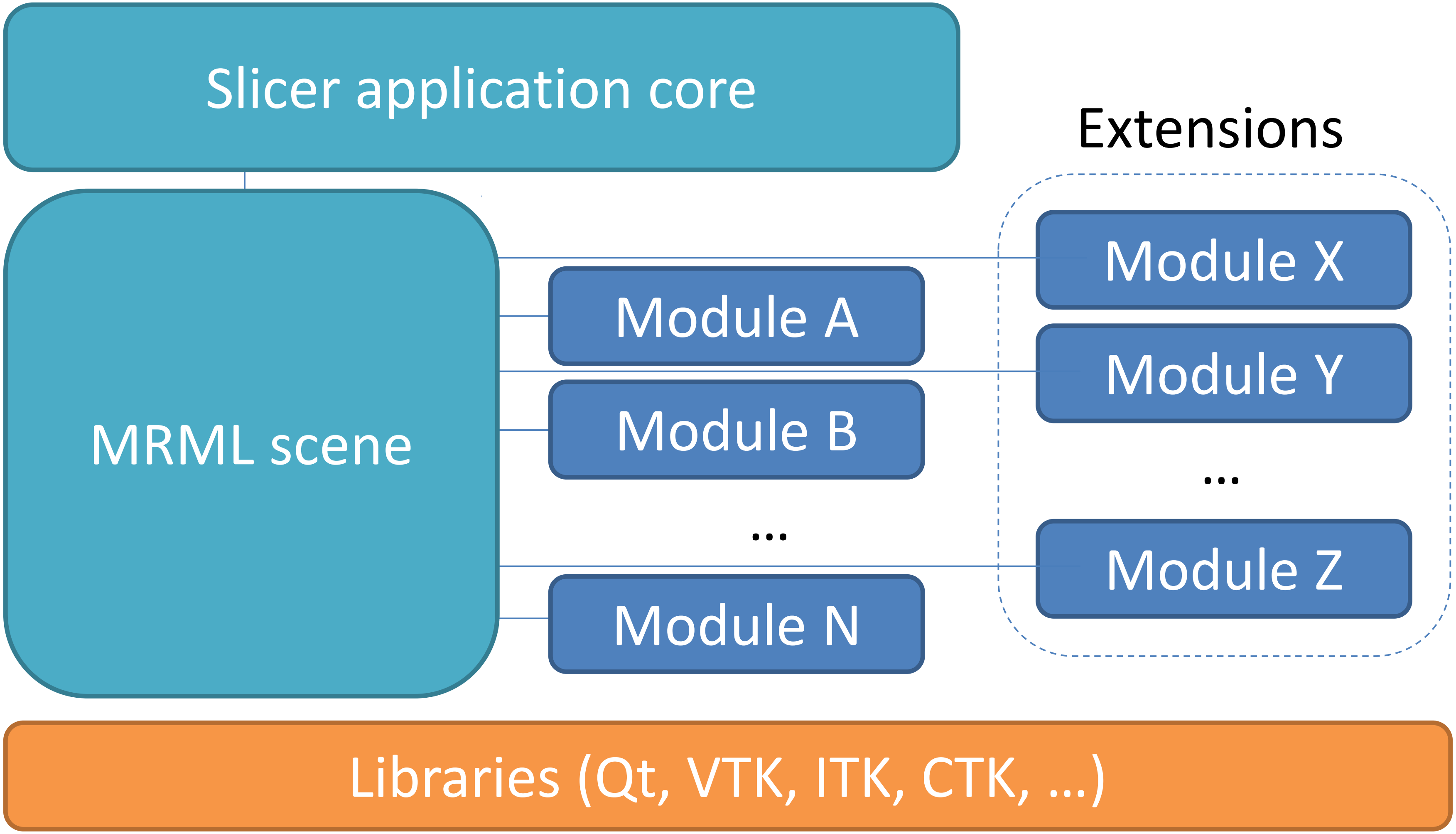
Part 2

- Use python console in Slicer
- Simple scripted module example

Part 3

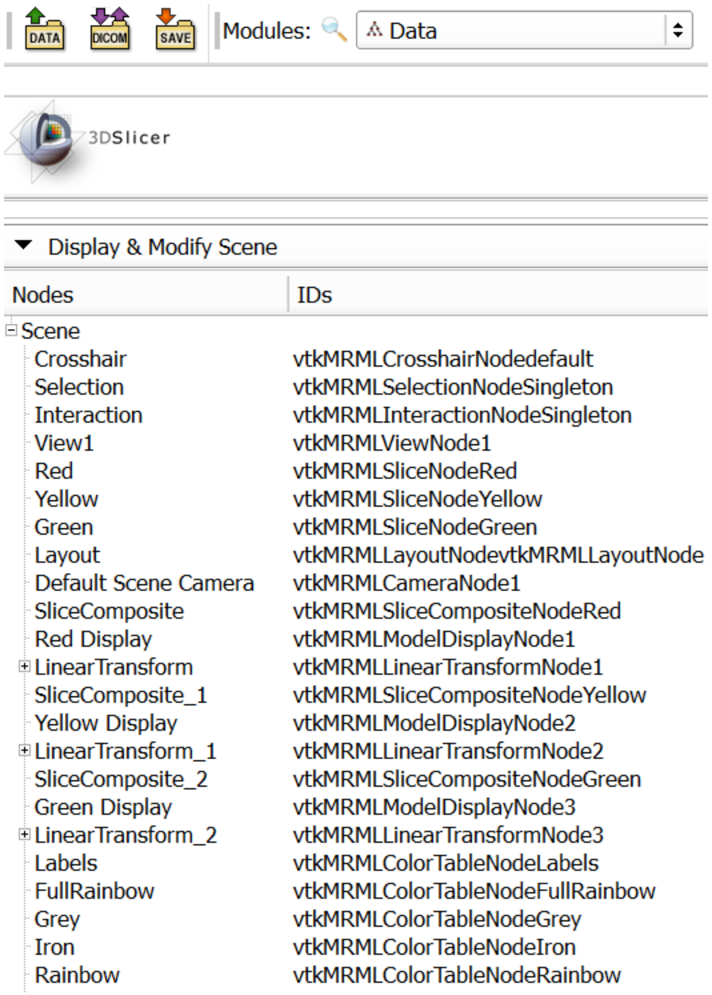
- Write simple scripted module individually

Slicer application architecture



Slicer data model

- Global repository for all data: **MRML scene**
  (MRML: Medical Reality Markup Language)
  - List of MRML nodes, each identified by a unique string ID
  - References, observations between nodes
- Modules communicate through reading/writing MRML nodes
  - Modules do not need to know about each other!

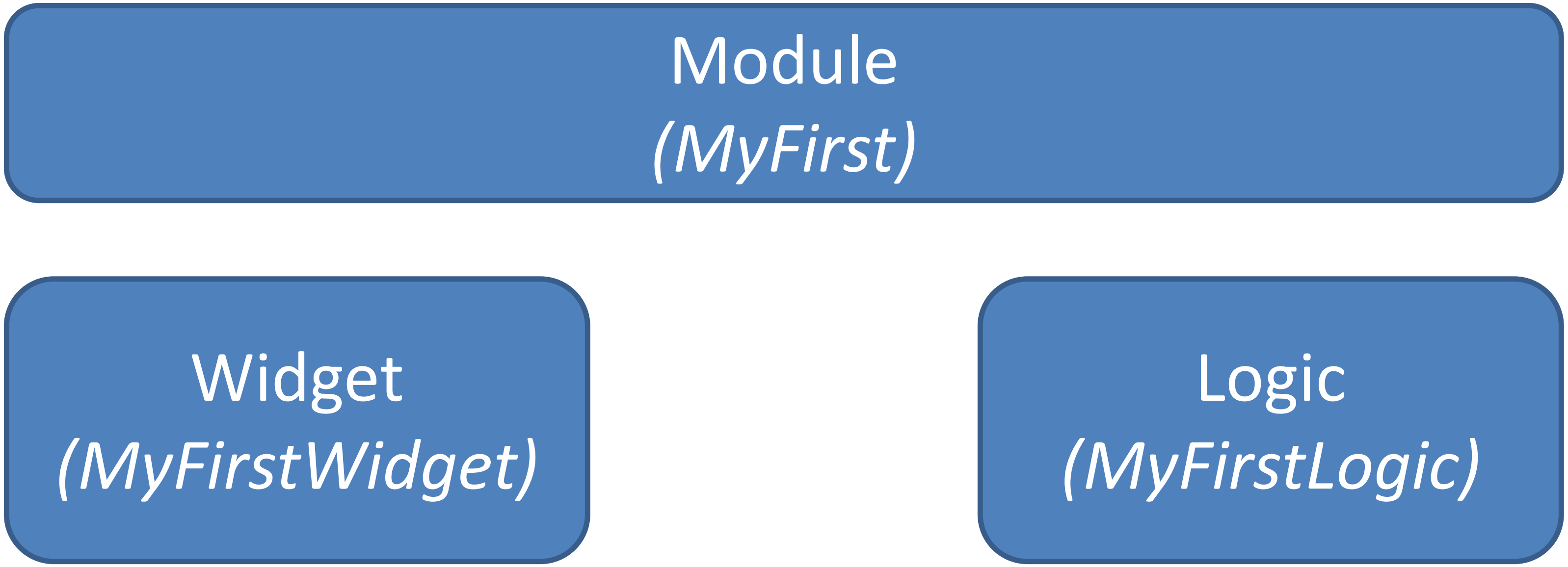


Nodes	IDs
⊟ Scene	
Crosshair	vtkMRMLCrosshairNodedefault
Selection	vtkMRMLSelectionNodeSingleton
Interaction	vtkMRMLInteractionNodeSingleton
View1	vtkMRMLViewNode1
Red	vtkMRMLSliceNodeRed
Yellow	vtkMRMLSliceNodeYellow
Green	vtkMRMLSliceNodeGreen
Layout	vtkMRMLLayoutNodevtkMRMLLayoutNode
Default Scene Camera	vtkMRMLCameraNode1
SliceComposite	vtkMRMLSliceCompositeNodeRed
Red Display	vtkMRMLModelDisplayNode1
⊞ LinearTransform	vtkMRMLLinearTransformNode1
SliceComposite_1	vtkMRMLSliceCompositeNodeYellow
Yellow Display	vtkMRMLModelDisplayNode2
⊞ LinearTransform_1	vtkMRMLLinearTransformNode2
SliceComposite_2	vtkMRMLSliceCompositeNodeGreen
Green Display	vtkMRMLModelDisplayNode3
⊞ LinearTransform_2	vtkMRMLLinearTransformNode3
Labels	vtkMRMLColorTableNodeLabels
FullRainbow	vtkMRMLColorTableNodeFullRainbow
Grey	vtkMRMLColorTableNodeGrey
Iron	vtkMRMLColorTableNodeIron
Rainbow	vtkMRMLColorTableNodeRainbow

MRML node

- Responsibilities:
  - Store data
  - Serialization to/from XML for file storage
  - No display or processing *methods*
- Basic types:
  - Data node
  - Display node: visualization options for data node content; multiple display nodes allowed
  - Storage node: what format, file name to use for persistent storage of the data node content

Scripted module implementation



Module

(MyFirst)

Widget

(MyFirstWidget)

Logic

(MyFirstLogic)

Module class

- Required. Only one global instance exists:

```
module = slicer.modules.volumes
```

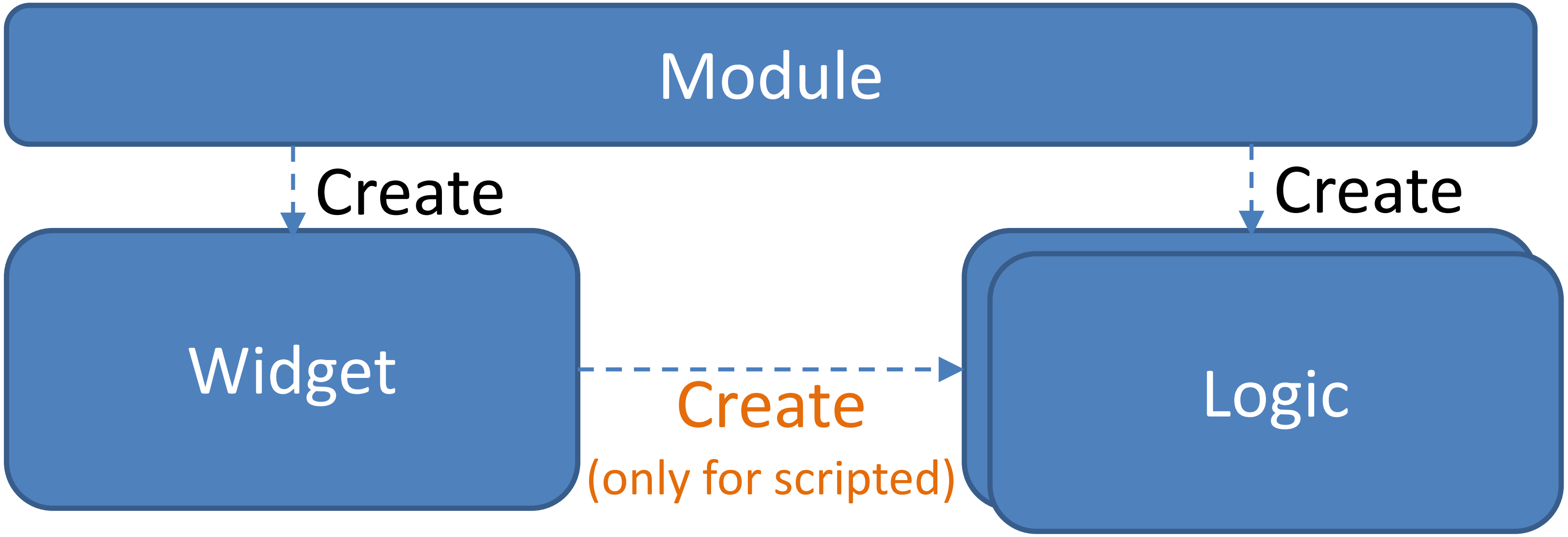
- Stores module name, description, icon, etc.
- Creates and holds a reference to logic and widget:
  - Loadable modules:

    ```
    widget = module.widgetRepresentation()
    logic = module.logic()
    ```

  - Python scripted modules:

    ```
    widget = module.widgetRepresentation().self()
    logic = widget.logic
    ```

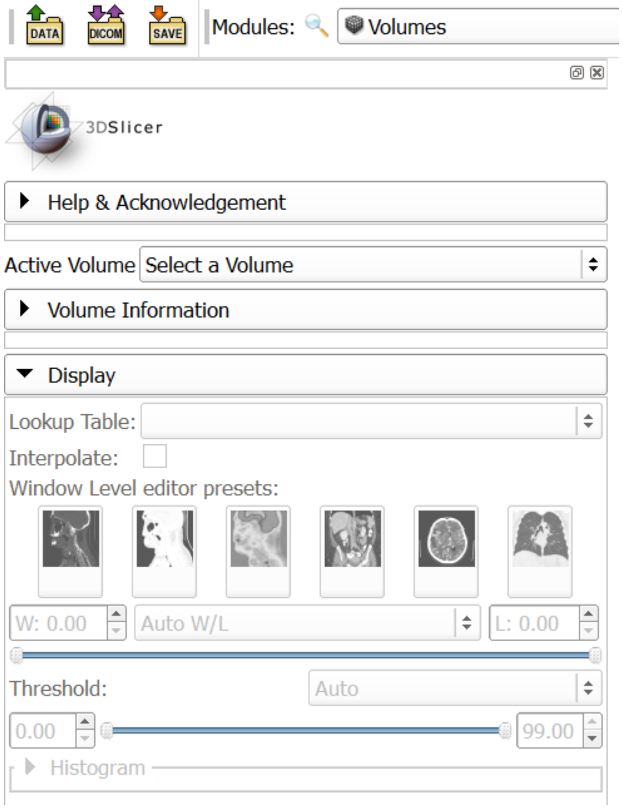
Scripted module implementation



Scripted module logic is not created automatically, it has to be instantiated in the Widget class.

Widget class

- Needed if the module has a user interface
- Typically only one global instance exists
- Defines the module's user interface
- Keeps user interface and nodes in sync (observes MRML nodes to get change notifications)
- Launches processing methods implemented in the logic class


Modules: Volumes

Active Volume: Select a Volume

Lookup Table:

Interpolate: □

Window Level editor presets:

W: 0.00 Auto W/L L: 0.00

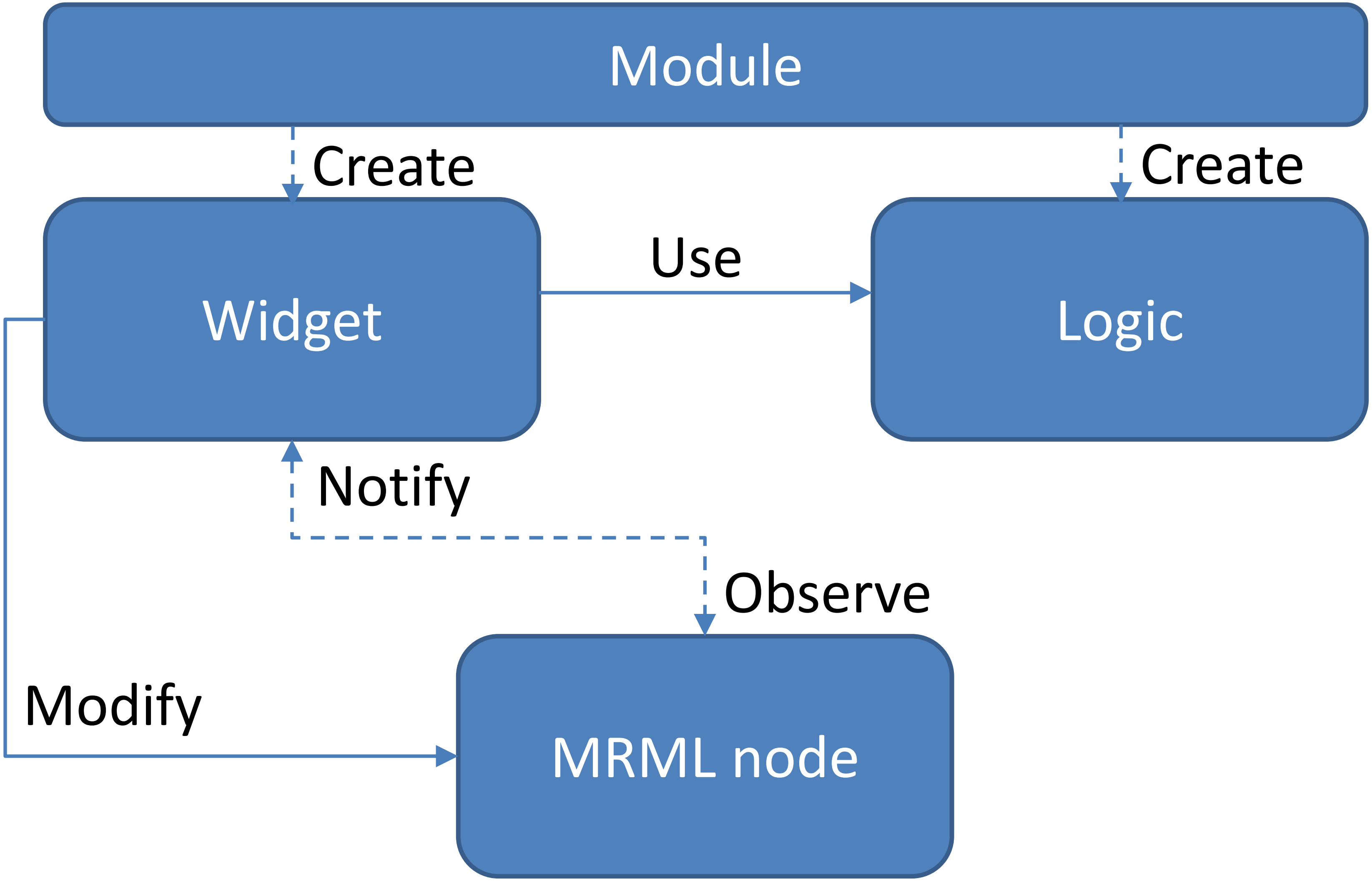
Threshold: Auto

0.00 99.00

Widget class

- Include a parameter node selector at the top (or use a singleton parameter node)
- If a parameter node is selected then add an observer to its modified events; if modified then call widget’s `updateGUIFromParameterNode()`
- If a parameter is changed in the GUI then update MRML node

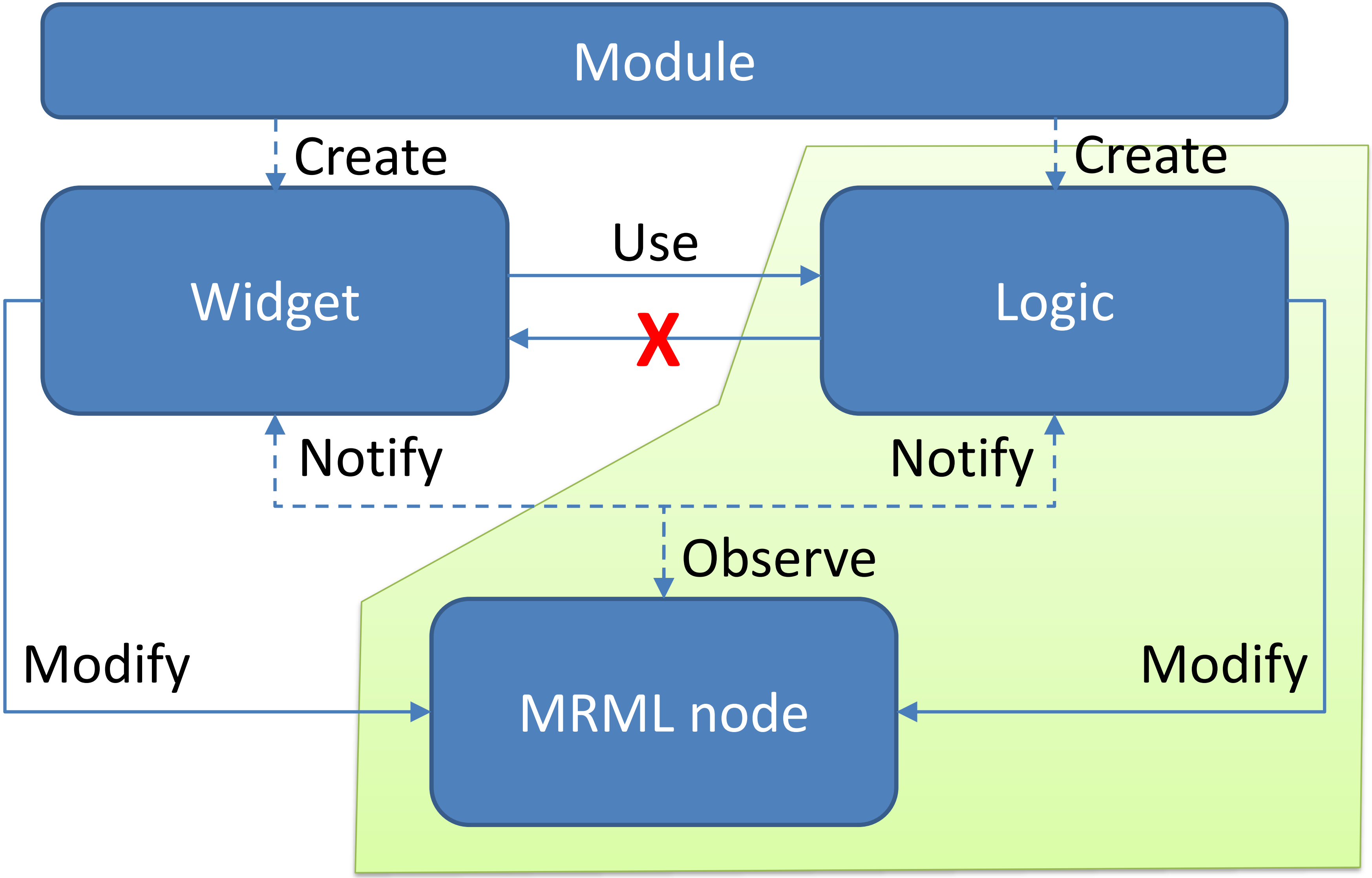
Scripted module implementation



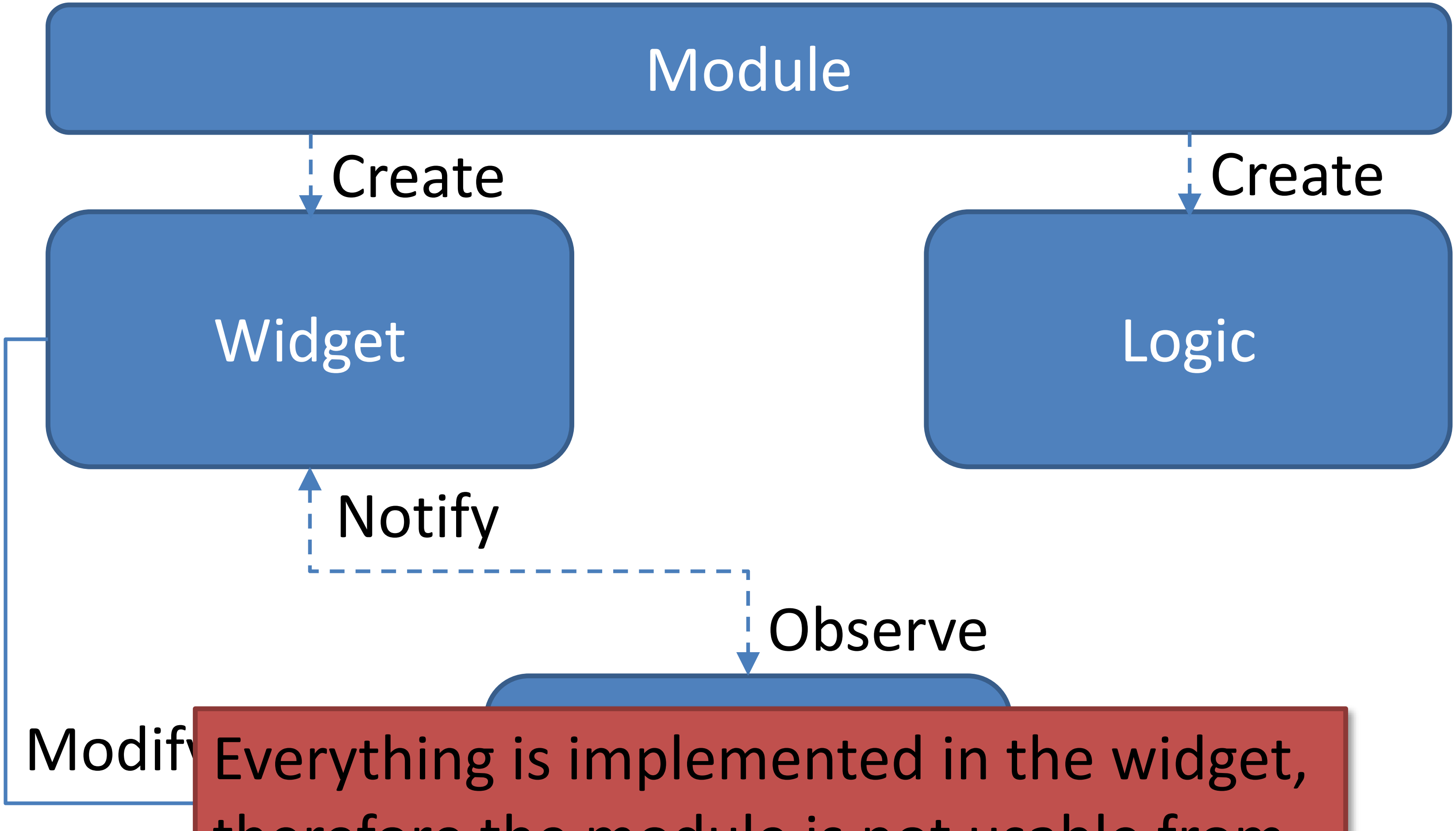
Logic class

- Needed if the module does any processing (always?)
- The module must be usable from another module, just by calling logic methods
- Must not rely on the Widget class: the module must be usable without even having a widget class
- Logic may be instantiated many times (to access utility functions inside)
- Logic may observe nodes: only if real-time background processing is needed (e.g., we observe some input nodes and update other nodes if input nodes are changed)

Scripted module implementation

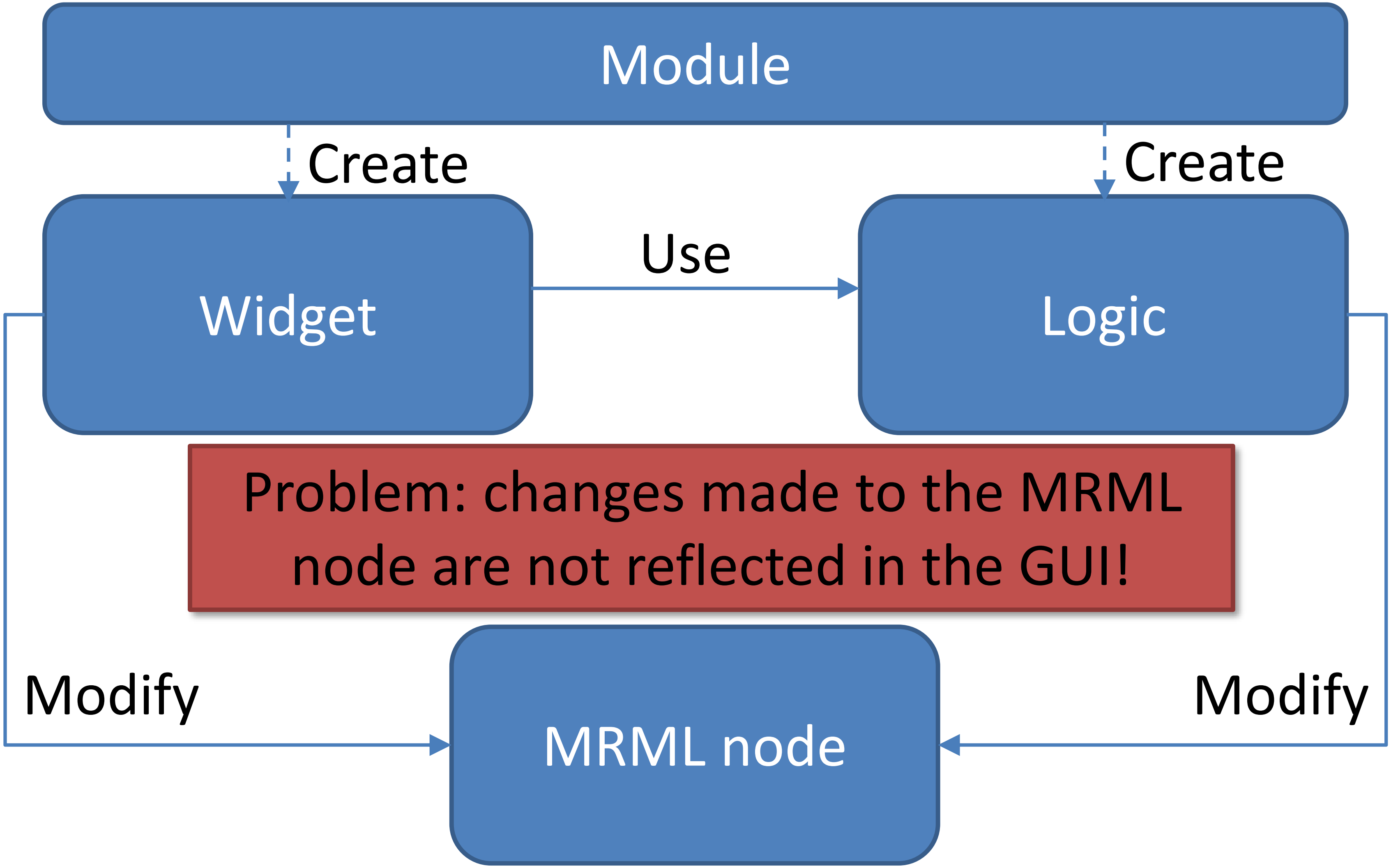


Common mistakes 1



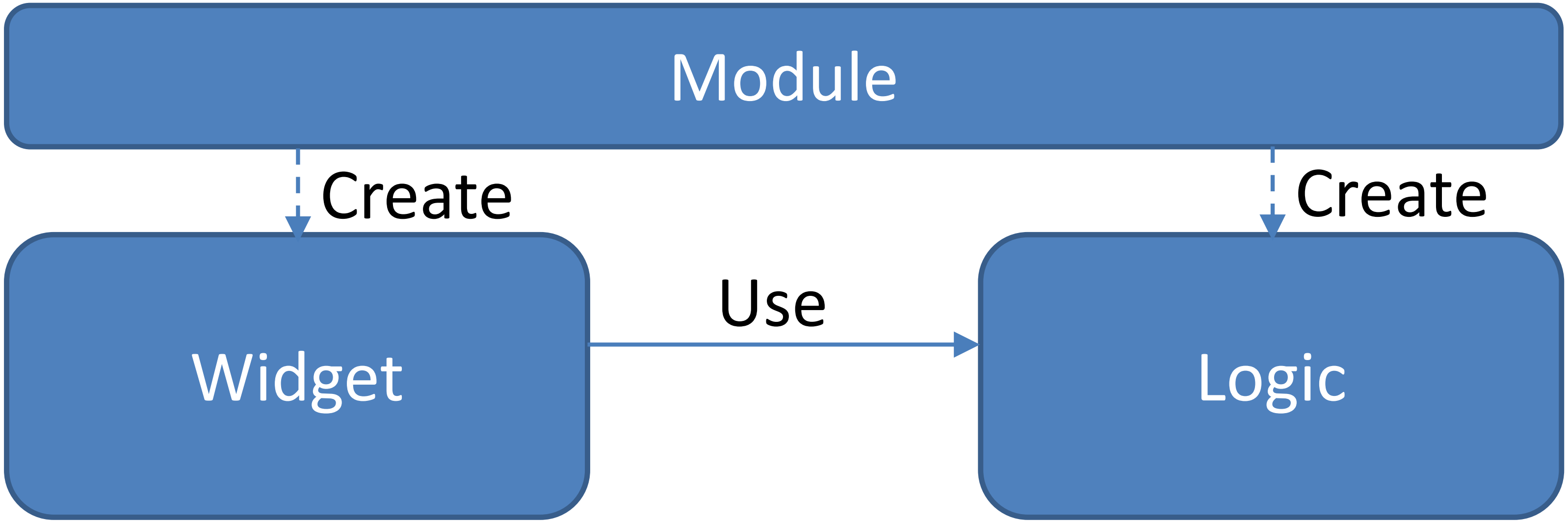
Everything is implemented in the widget, therefore the module is not usable from another module or with a custom GUI!

Common mistakes 2



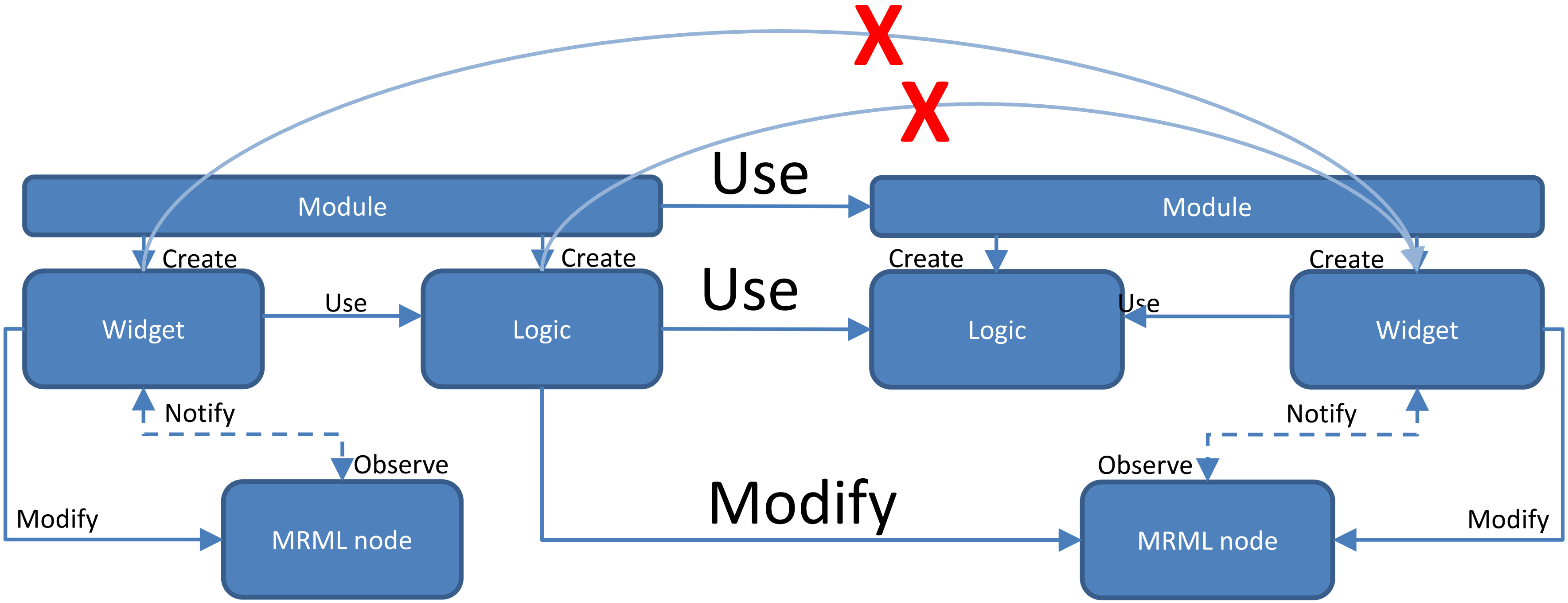
Problem: changes made to the MRML node are not reflected in the GUI!

Common mistakes 3



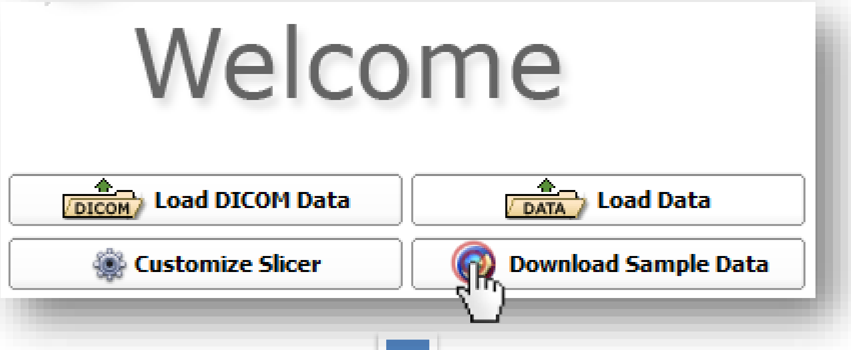
No parameter node is used. When the scene is saved and reloaded all the settings in the module user interface are lost!

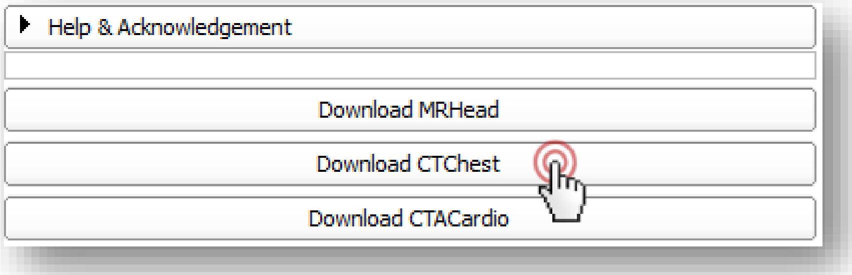
Communication between modules

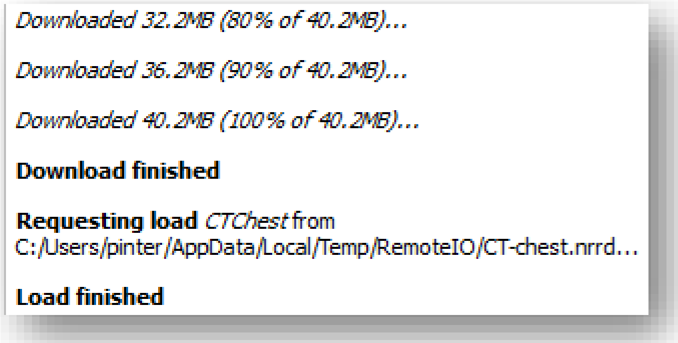


- Module logic may modify any MRML nodes – most common form of communication
- Module logic class may use another module's logic class
- Module class may use another module class (e.g., to access module logic and pass it to its own logic)

Launch Slicer and load data







Downloaded 32.2MB (80% of 40.2MB)...

Downloaded 36.2MB (90% of 40.2MB)...

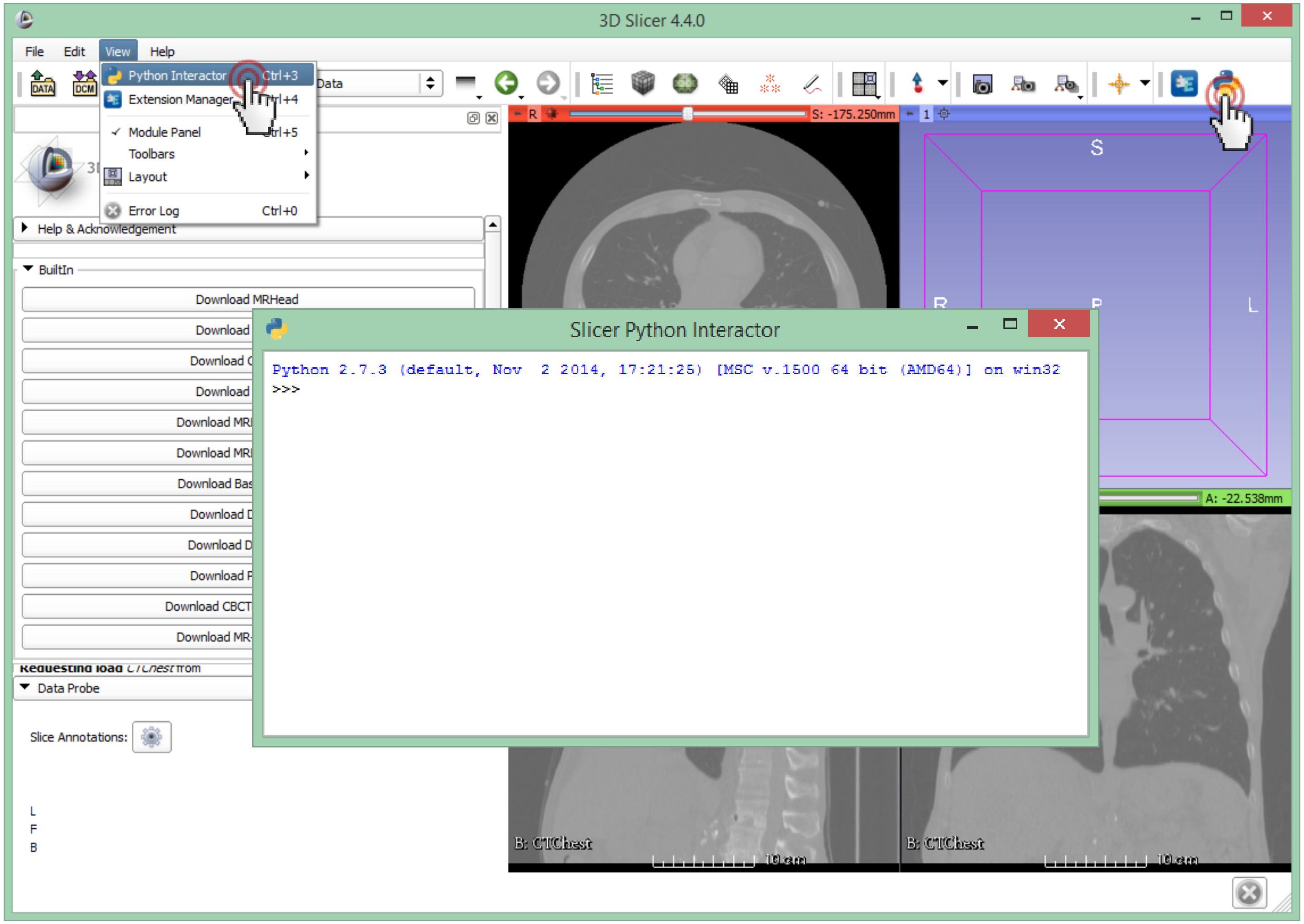
Downloaded 40.2MB (100% of 40.2MB)...

Download finished

Requesting load *CTChest* from C:/Users/pinter/AppData/Local/Temp/RemoteIO/CT-chest.nrrd...

Load finished

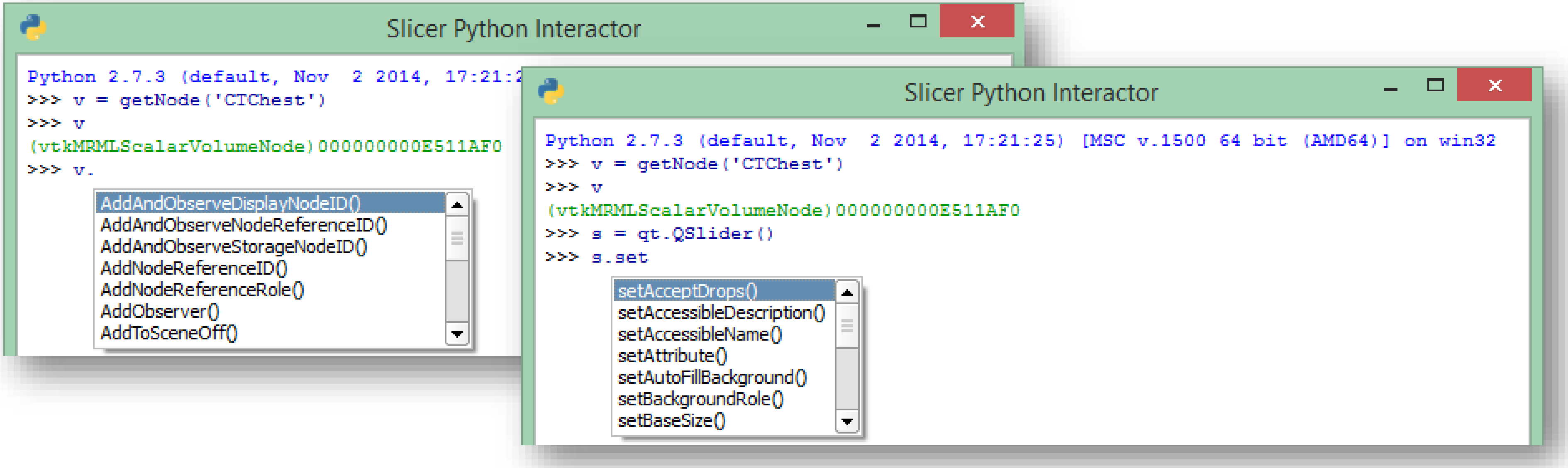
Introducing the python console



```
Python 2.7.3 (default, Nov  2 2014, 17:21:25) [MSC v.1500 64 bit (AMD64)] on win32
>>>
```

Auto-completion feature

- Essential tool that provides API information
- Press TAB to bring up auto-complete window to
  - Explore available functions of a certain object
  - Save typing



```
Python 2.7.3 (default, Nov  2 2014, 17:21:2
>>> v = getNode('CTChest')
>>> v
(vtkMRMLScalarVolumeNode)000000000E511AF0
>>> v.
```

```
Python 2.7.3 (default, Nov  2 2014, 17:21:25) [MSC v.1500 64 bit (AMD64)] on win32
>>> v = getNode('CTChest')
>>> v
(vtkMRMLScalarVolumeNode)000000000E511AF0
>>> s = qt.QSlider()
>>> s.set
```

Accessing the MRML scene and nodes

- Using utility functions – in slicer.util

```
v = getNode('CTChest')
```

OR

```
v = getNode('CT*')
```

getNode: somewhat ambiguous, recommended for testing & debugging only

- Accessing MRML scene directly

```
v=slicer.mrmlScene.GetFirstNodeByName('CTChest')
```

OR

```
v=slicer.mrmlScene.GetFirstNodeByClass('vtkMRMLScalarVolumeNode')
```

Information about variables

- Get variable type and pointer: enter the variable name

```
v
(vtkMRMLScalarVolumeNode)0000008FF76243B8
```

Note: It’s always good to check the variable after you create it

- Show node content: all members and attributes inheritance tree

```
print(v)
```

- Show node API: description of all methods

```
help(v)
```

Manipulating Volumes

- Setting window/level values programmatically

```
vd = v.GetDisplayNode()
vd.SetAutoWindowLevel(0)
vd.SetWindowLevel(350,40)
```

- What methods/parameters are available:

```
help(vd)
```

OR

```
vd
(vtkCommonCorePython.vtkMRMLScalarVolumeDisplayNode)0000021AD5436588
```

http://www.slicer.org/doc/html/classes.html

Manipulating Volumes

- Accessing, changing voxels – using numpy
- Get voxel value at (100,200,30) position

```
va = slicer.util.arrayFromVolume(v)   <= get voxels as a numpy array
va[100,200,30]
-986 <= voxel value
```

- Thresholding

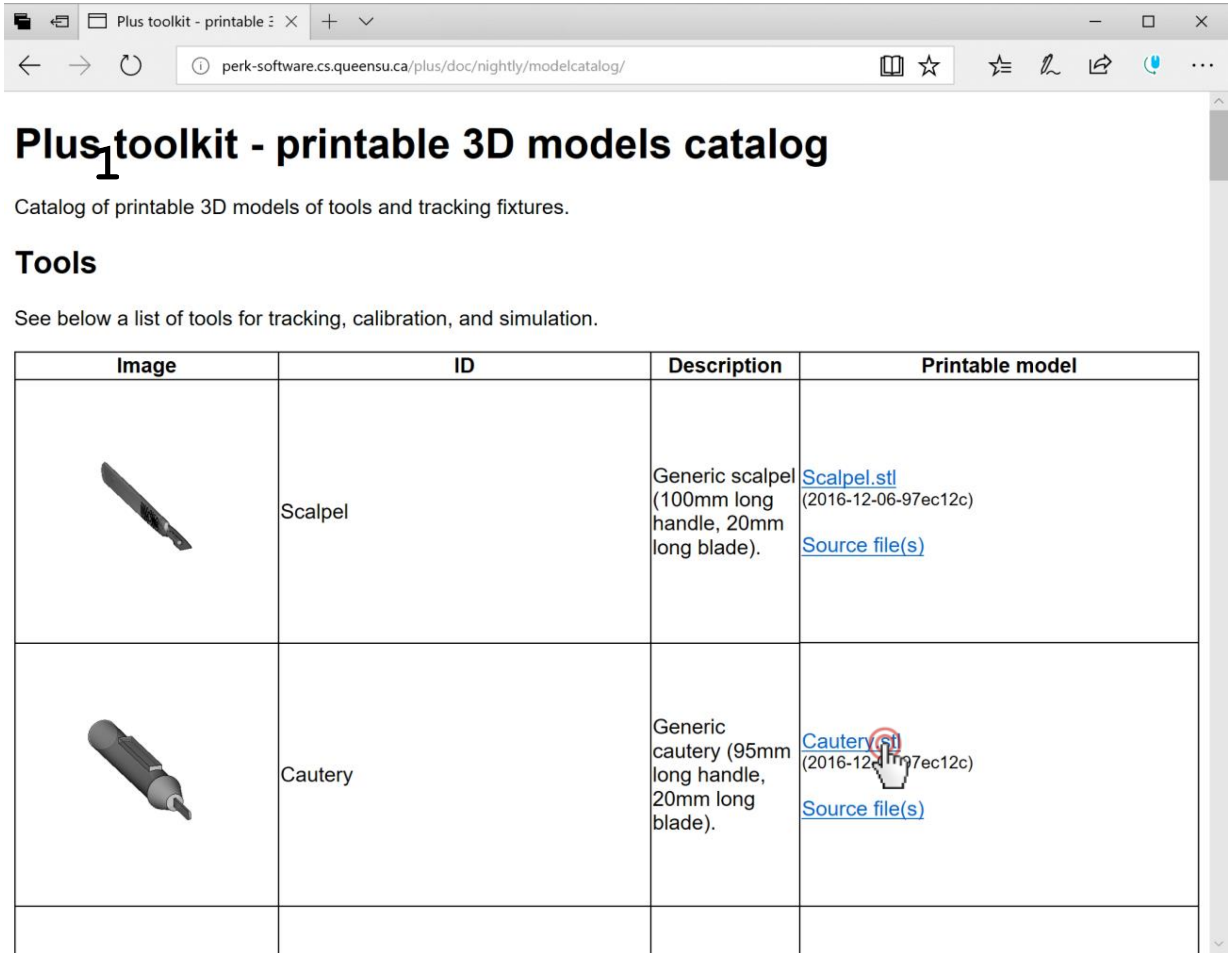
```
vaOriginal=va.copy() <= save the original voxel values
va[va<200] = -3000
va[va>200] = 2000
slicer.util.arrayFromVolumeModified(v)  <= indicate to Slicer that
                                           updates are completed
```

- Process with arbitrary function

```
va[:] = vaOriginal[:]*2.5-500; v.Modified()
```

Load model

http://perk-software.cs.queensu.ca/plus/doc/nightly/modelcatalog/



Plus toolkit - printable 3
perk-software.cs.queensu.ca/plus/doc/nightly/modelcatalog/

Plus toolkit - printable 3D models catalog

Catalog of printable 3D models of tools and tracking fixtures.

Tools

See below a list of tools for tracking, calibration, and simulation.

Image	ID	Description	Printable model
	Scalpel	Generic scalpel (100mm long handle, 20mm long blade).	Scalpel.stl (2016-12-06-97ec12c) Source file(s)
	Cautery	Generic cautery (95mm long handle, 20mm long blade).	Cautery.stl (2016-12-06-97ec12c) Source file(s)

Manipulating Models

- Setting model color programmatically

```
c = getNode('Cautery')
cd = c.GetDisplayNode()
cd.SetColor(1,0,0)
```

- Change model to a sphere

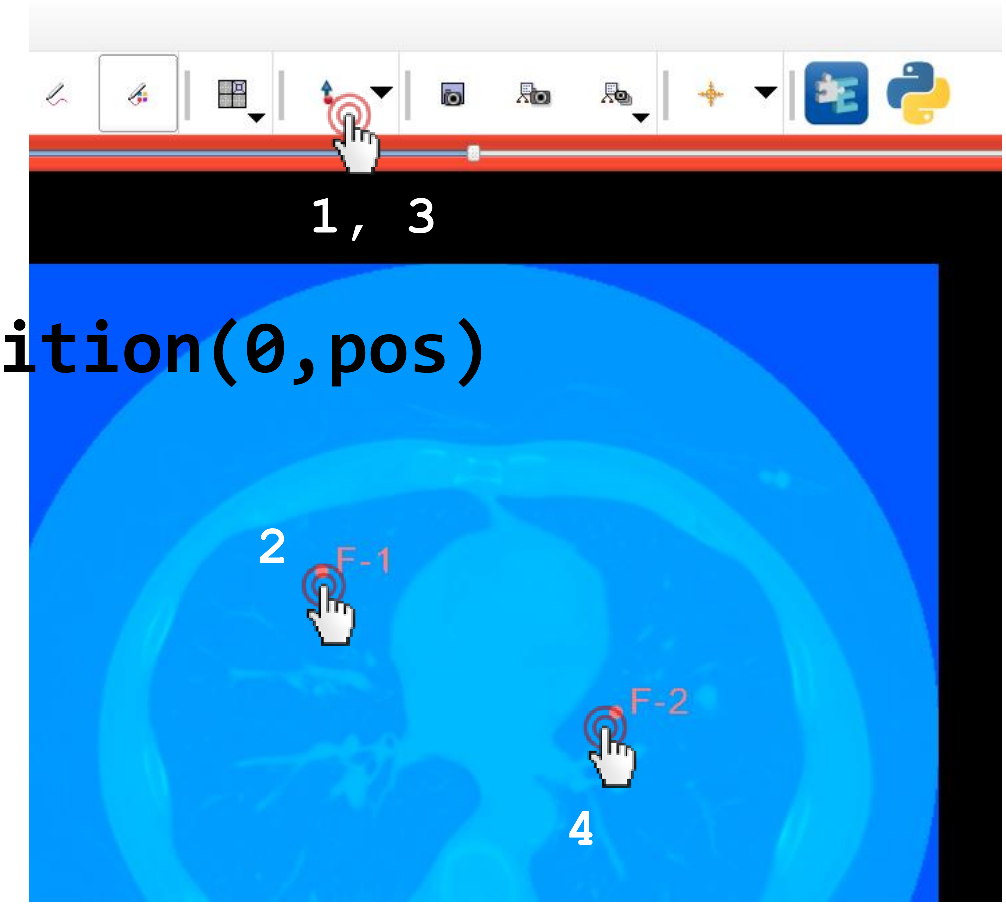
```
s = vtk.vtkSphereSource()
s.SetRadius(30)
s.SetCenter(30,40,60)
s.Update()
c.SetAndObservePolyData(s.GetOutput())
```

Manipulating Markups

- Create 2 markup points
- Get markup position

```
f = getNode('F')
pos=[0,0,0]
f.GetNthFiducialPosition(0,pos)
pos
```

```
[58.93727622783058,
 45.58082600473318,
 -170.25000000000001]
```



Observing MRML objects

```
def printPos(caller=None, event=None):
  f = getNode('F')
  pos=[0,0,0]
  f.GetNthFiducialPosition(0,pos)
  print(pos)
```

These lines start with 2 spaces!

```
printPos()
[58.93727622783058, 45.58082600473318, -170.2500000000001]
```

```
obsTag=f.AddObserver(vtk.vtkCommand.ModifiedEvent, printPos)
```

=> Drag-and-drop first fiducial

```
[20.267904116373643, 4.977985287703447, -170.2500000000001]
[21.92516292115039, 1.6634676781499849, -170.2500000000001]
[22.477582522742637, 1.1110480765577364, -170.2500000000001]
[24.68726092911574, 0.5586284749655164, -170.2500000000001]
[26.34451973388832, 0.00620887337326792, -170.2500000000001]
```

```
f.RemoveObserver(obsTag)
```

Part 2

- Use python console in Slicer
- Simple scripted module example

Python in general

- **Blocks defined by indentation: 2 spaces**
- Case sensitive
- Comments

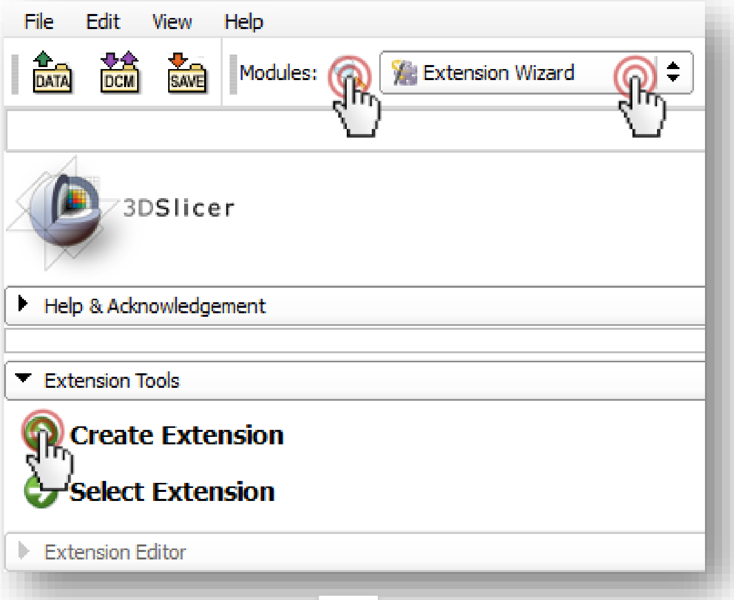
```
# This whole row is a comment
”””This is a potentially multi-line comment”””
```

- Blocks defined by indentation
- An object refers to itself as `self` (in C++: `this`)
- Namespaces: `slicer, ctk, vtk, qt`
- Blocks … indentation … spaces!

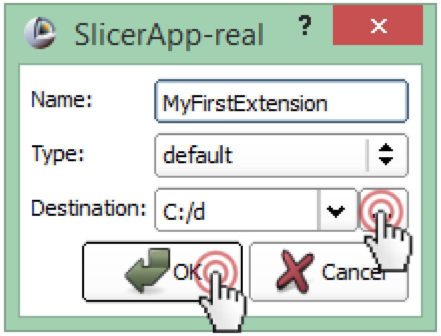
Text editor / IDE

- Using a proper text editor is essential
  - Replace all, Easy comment/uncomment, indent, …
  - Syntax highlighting
  - Keyboard shortcuts
  - Recommended:
    Windows-only: **Notepad++**, Mac-only: Xcode
    Cross-platform: Atom, Sublime Text,
- Integrated development environment:
  - Text editor + debugger, code browser, …
  - Recommended: **PyCharm**, LiClipse

Create extension



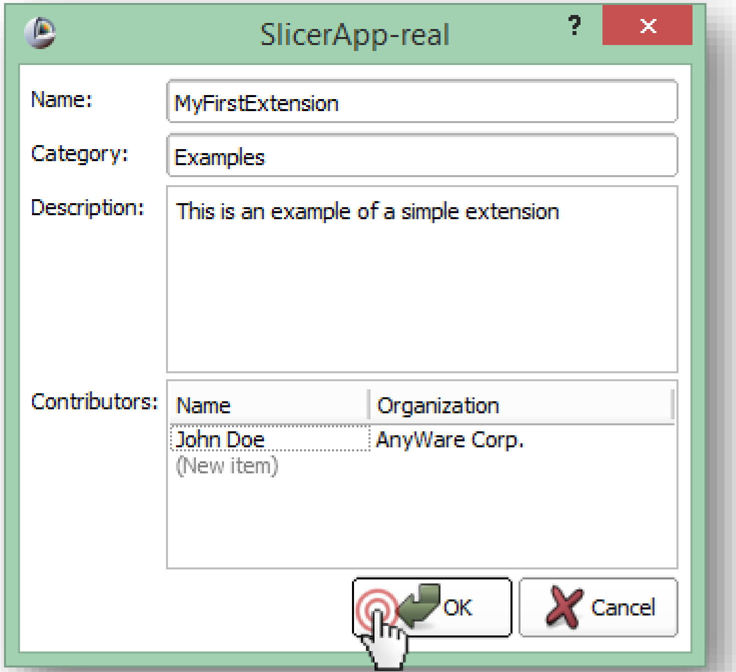
Modules: Extension Wizard



Name: MyFirstExtension

Type: default

Destination: C:/d



Name: MyFirstExtension

Category: Examples

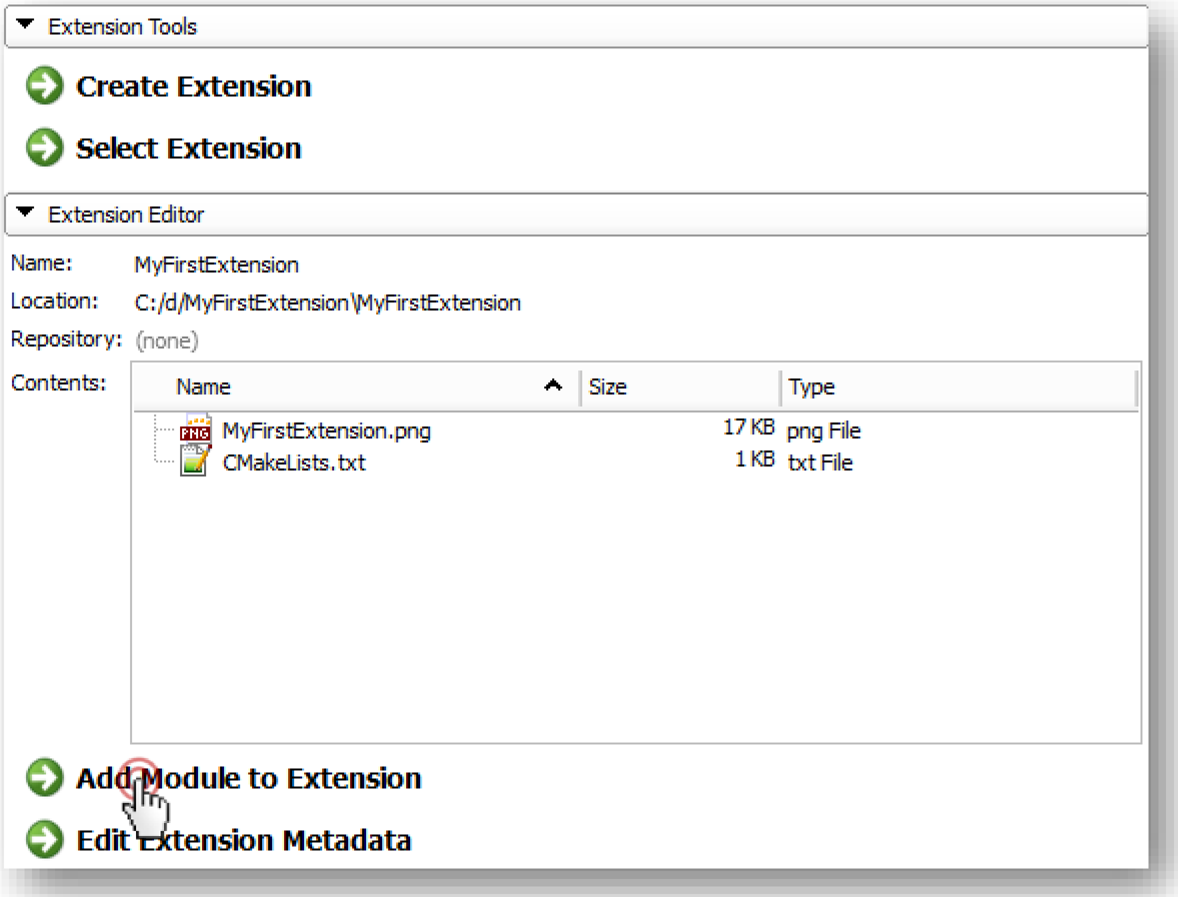
Description: This is an example of a simple extension

Contributors:

Name	Organization
John Doe	AnyWare Corp.
(New item)	

1. Enter name: *MyFirstExtension*
2. Choose destination: In your SVN folder!

Create module



▼ Extension Tools

Create Extension

Select Extension

▼ Extension Editor

Name: MyFirstExtension

Location: C:/d/MyFirstExtension\MyFirstExtension

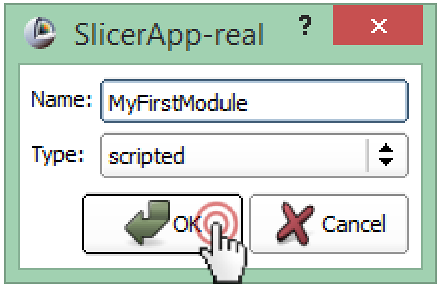
Repository: (none)

Contents:

Name	Size	Type
MyFirstExtension.png	17 KB	png File
CMakeLists.txt	1 KB	txt File

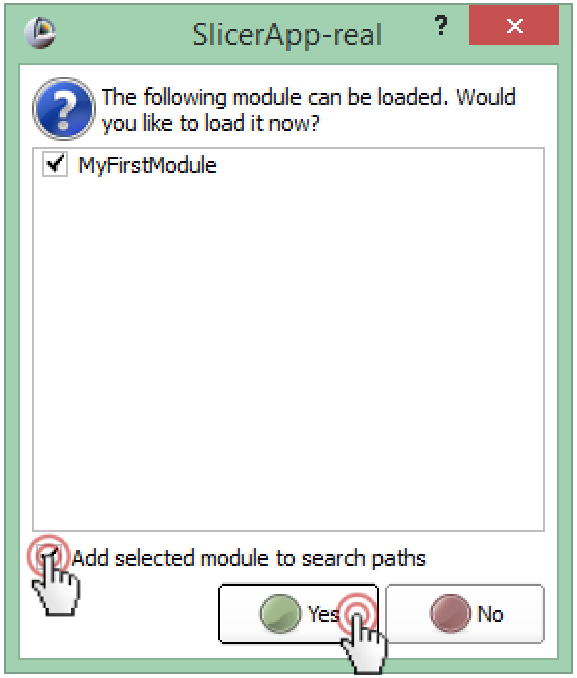
Add Module to Extension

Edit Extension Metadata



Name: MyFirstModule

Type: scripted



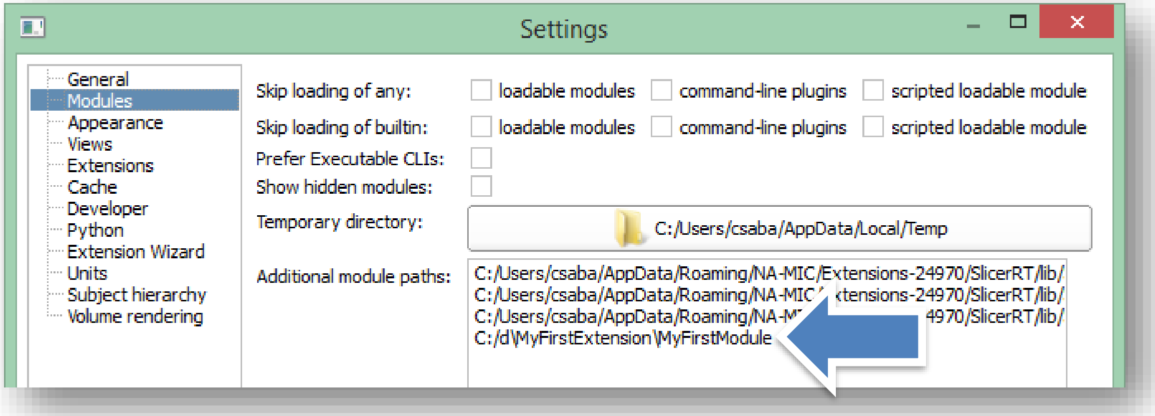
The following module can be loaded. Would you like to load it now?

☑ MyFirstModule

☑ Add selected module to search paths

Module paths

- In Application Settings



Skip loading of any: □ loadable modules □ command-line plugins □ scripted loadable module

Skip loading of builtin: □ loadable modules □ command-line plugins □ scripted loadable module

Prefer Executable CLIs: □

Show hidden modules: □

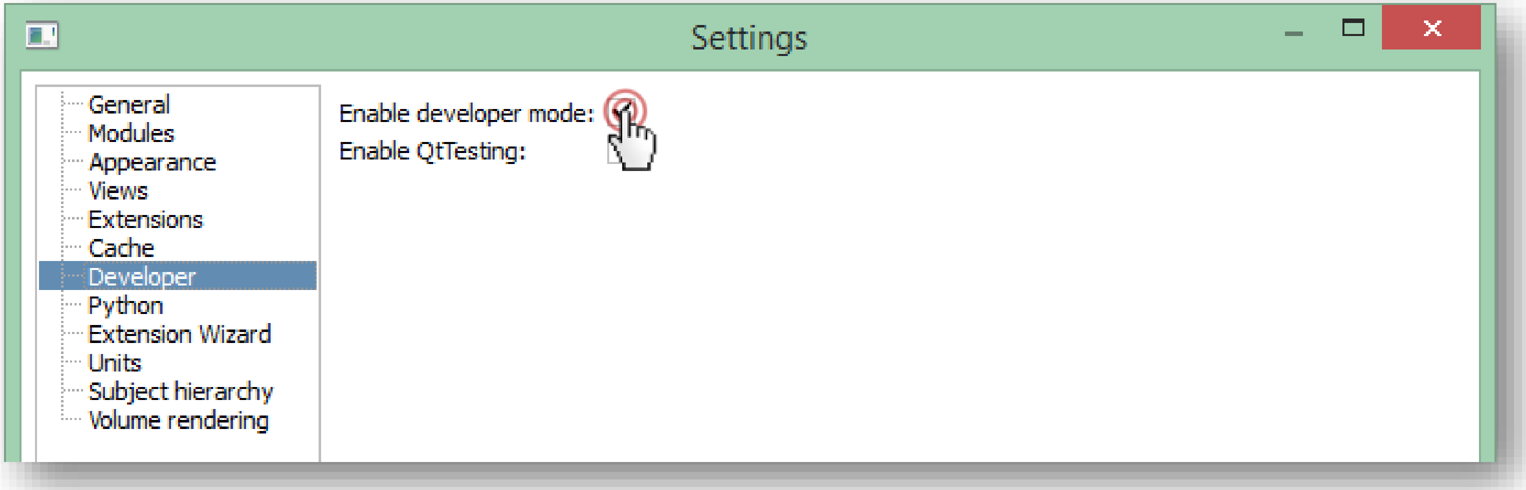
Temporary directory: C:/Users/csaba/AppData/Local/Temp

Additional module paths:
C:/Users/csaba/AppData/Roaming/NA-MIC/Extensions-24970/SlicerRT/lib/
C:/Users/csaba/AppData/Roaming/NA-MIC xtensions-24970/SlicerRT/lib/
C:/Users/csaba/AppData/Roaming/NA-M 4970/SlicerRT/lib/
C:/d\MyFirstExtension\MyFirstModule

- You can add by drag&drop
  - But you don’t need to because checking the “Add selected module to search paths” checkbox did it for you

Developer mode

- Enable features useful for development
  - Allow dynamic reload of source code etc.
- In Application Settings



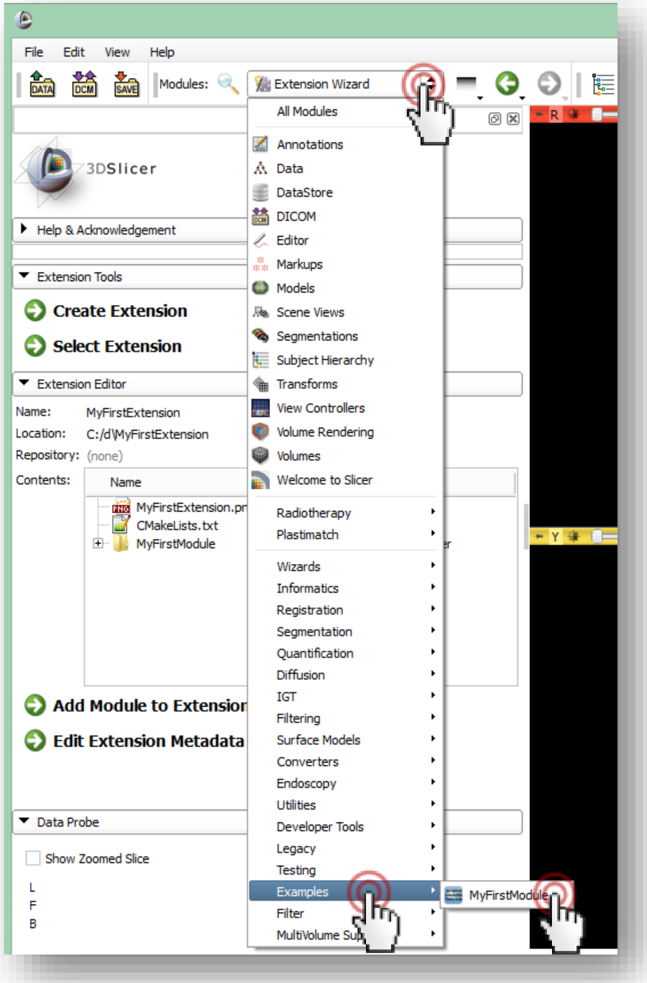
Settings

Enable developer mode: ☑

Enable QtTesting: □

- Restart Slicer

Find the new module in Slicer

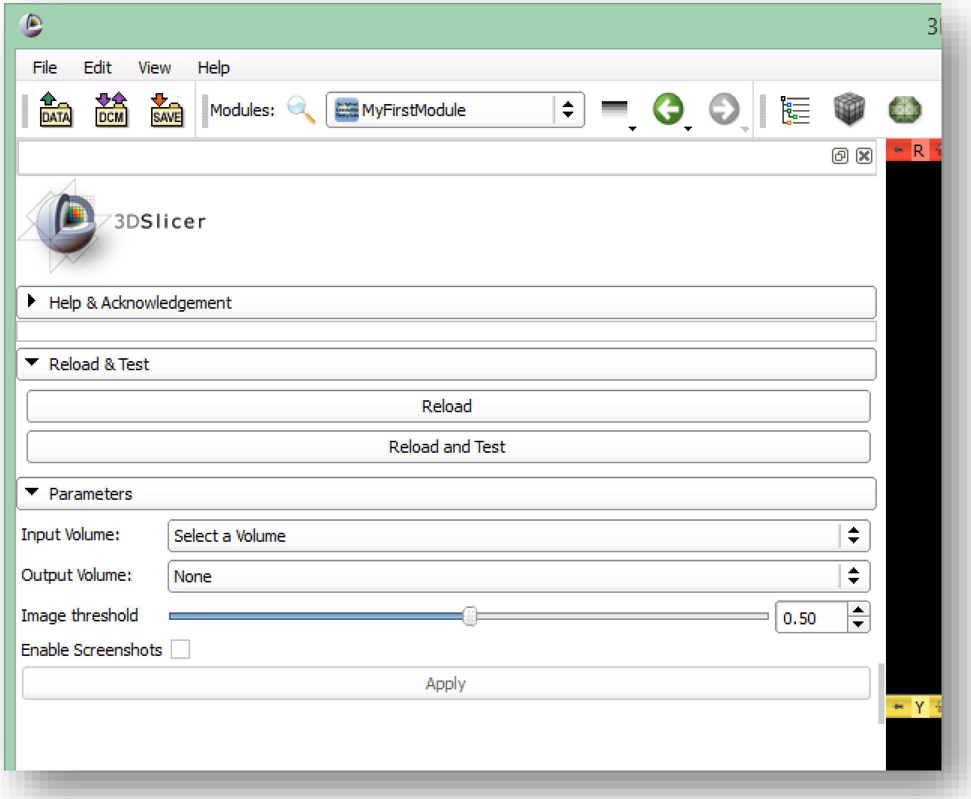


Modules: Extension Wizard

Name: MyFirstExtension

Location: C:/d\MyFirstExtension

Repository: (none)



Modules: MyFirstModule

Input Volume: Select a Volume

Output Volume: None

Image threshold 0.50

Enable Screenshots □

Commit your changes regularly

- Commit your changes
- New files need to be added explicitly
- Commit message should look like this:
  "*Re #7: Description of why I did what I did*"
  (do not describe what you did, it's obvious from the diff)
- When you think you're done
  "*Test #7: Description of why I did what I did*"
- When everybody agrees you're done
  "*Fixed #7: Description of why I did what I did*"
- Update before each commit!

Write our scripted module #1

- Open *MyFirstExtension\MyFirstModule\MyFirstModule.py*
- Rename the module

```
def __init__(self, parent):
  ScriptedLoadableModule.__init__(self, parent)
  self.parent.title = "Center of Mass"
```

- Create widgets to specify inputs:

  Under `def setup(self)` look for `input volume selector`, change

```
self.inputSelector.nodeTypes = ["vtkMRMLMarkupsFiducialNode"]
self.inputSelector.selectNodeUponCreation = False
```

- and

```
parametersFormLayout.addRow("Input Markups: ", self.inputSelector)
```

Write our scripted module #2

- Look for **`class MyFirstModuleLogic`** towards the bottom
- Insert this code above the **`Run`** function

```
def getCenterOfMass(self, markupsNode):
   centerOfMass = [0,0,0]

   import numpy as np
   sumPos = np.zeros(3)
   for i in range(markupsNode.GetNumberOfMarkups()):
     pos = np.zeros(3)
     markupsNode.GetNthFiducialPosition(i,pos)
     sumPos += pos

   centerOfMass = sumPos / markupsNode.GetNumberOfMarkups()

   logging.info('Center of mass for \'' + markupsNode.GetName() + '\': ' + repr(centerOfMass))

   return centerOfMass
```

Write our scripted module #3

- Change the **Run** function to look like this:

```
def run(self, inputMarkups, outputVolume, imageThreshold, enableScreenshots=0):
   """
   Run the actual algorithm
   """
   self.centerOfMass = self.getCenterOfMass(inputMarkups)

   return True
```

Write our scripted module #4

- Create the text field in the `setup` function under the `Apply` button, above `connections`

```
 self.centerOfMassValueLabel = qt.QLabel()
parametersFormLayout.addRow("Center of mass",self.centerOfMassValueLabel)
```

- Validating button state and displaying the output into **`MyFirstModuleWidget` – replace these functions:**

```
 def onSelect(self):
    self.applyButton.enabled = self.inputSelector.currentNode()

  def onApplyButton(self):
    logic = MyFirstModuleLogic()
    enableScreenshotsFlag = self.enableScreenshotsFlagCheckBox.checked
    imageThreshold = self.imageThresholdSliderWidget.value
    logic.run(self.inputSelector.currentNode(),
self.outputSelector.currentNode(), imageThreshold, enableScreenshotsFlag)
    self.centerOfMassValueLabel.text = str(logic.centerOfMass)
```

- Pay attention to correct indentation!

Try our scripted module

- Go to our module in *Examples / Center of Mass*
- Add a few markups
- Press *Apply*
  - 1. Display displacement center of mass position → works!
  - 2. Displays nothing → error can be seen in the Python interactor

Add auto-update

- Repurpose the checkbox:

```
# check box to trigger taking screen shots for later use in tutorials
self.enableScreenshotsFlagCheckBox = qt.QCheckBox()
self.enableScreenshotsFlagCheckBox.checked = 0
self.enableScreenshotsFlagCheckBox.setToolTip(“Enable auto-update")
parametersFormLayout.addRow("Auto-update", self.enableScreenshotsFlagCheckBox)
self.observedMarkupNode = None
self.markupsObserverTag = None
self.enableScreenshotsFlagCheckBox.connect("toggled(bool)", self.onEnableAutoUpdate)
```

- Respond to modification events: add these above onApplyButton()

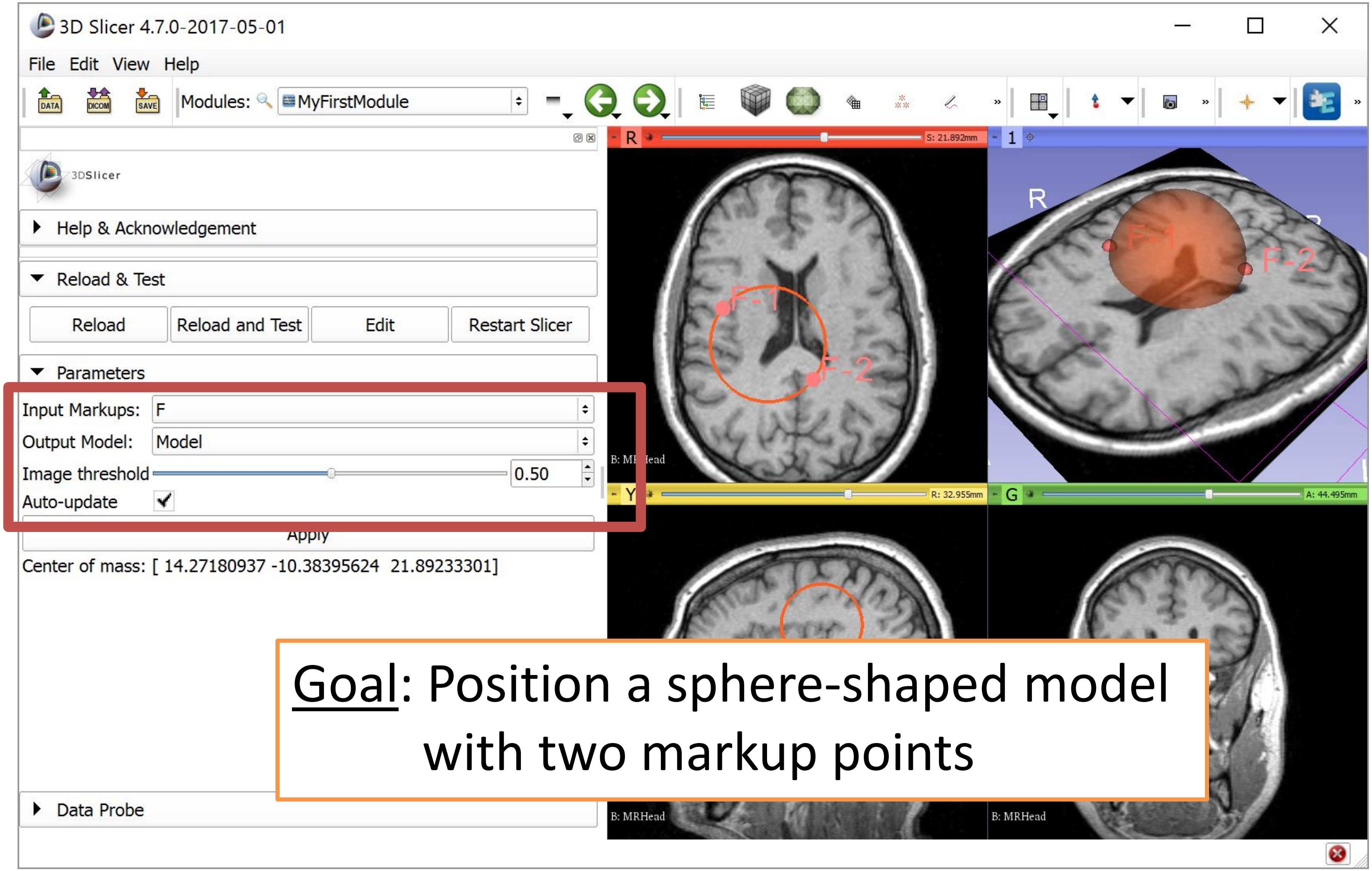
```
  def onEnableAutoUpdate(self, autoUpdate):
    if self.markupsObserverTag:
      self.observedMarkupNode.RemoveObserver(self.markupsObserverTag)
      self.observedMarkupNode = None
      self.markupsObserverTag = None
    if autoUpdate and self.inputSelector.currentNode:
      self.observedMarkupNode = self.inputSelector.currentNode()
      self.markupsObserverTag = self.observedMarkupNode.AddObserver(
        vtk.vtkCommand.ModifiedEvent, self.onMarkupsUpdated)

  def onMarkupsUpdated(self, caller=None, event=None):
    self.onApplyButton()
```

Part 3

Write simple scripted module (somewhat) independently

Task description



Input Markups: F

Output Model: Model

Image threshold 0.50

Auto-update ✓

Center of mass: [14.27180937 -10.38395624 21.89233301]

Goal: Position a sphere-shaped model with two markup points

API documentation

Generic computing and GUI libraries

Toolkit	API documentation
Python	https://docs.python.org/2/index.html
Numpy	http://docs.scipy.org/doc/numpy/reference
VTK	http://www.vtk.org/doc/release/7.1/html/classes.html
SimpleITK	http://www.itk.org/SimpleITKDoxygen/html/classes.html
Qt	http://doc.qt.io/qt-4.8/classes.html

Slicer-specific libraries

Toolkit	API documentation
Slicer core	C++: http://www.slicer.org/doc/html/classes.html Python: http://mwoehlke-kitware.github.io/Slicer/Base/slicer.html For up-to-date docs, type this into Python console: `help(slicer.util)`
CTK	http://www.commontk.org/docs/html/classes.html

Where to find examples

- Slicer core: https://github.com/Slicer/Slicer
- Extensions:
  - Index of extensions (see repository in *.s4ext): https://github.com/Slicer/ExtensionsIndex
  - SlicerIGT: https://github.com/SlicerIGT/SlicerIGT/
  - SlicerRT: https://app.assembla.com/spaces/slicerrt/subversion/source/HEAD/trunk

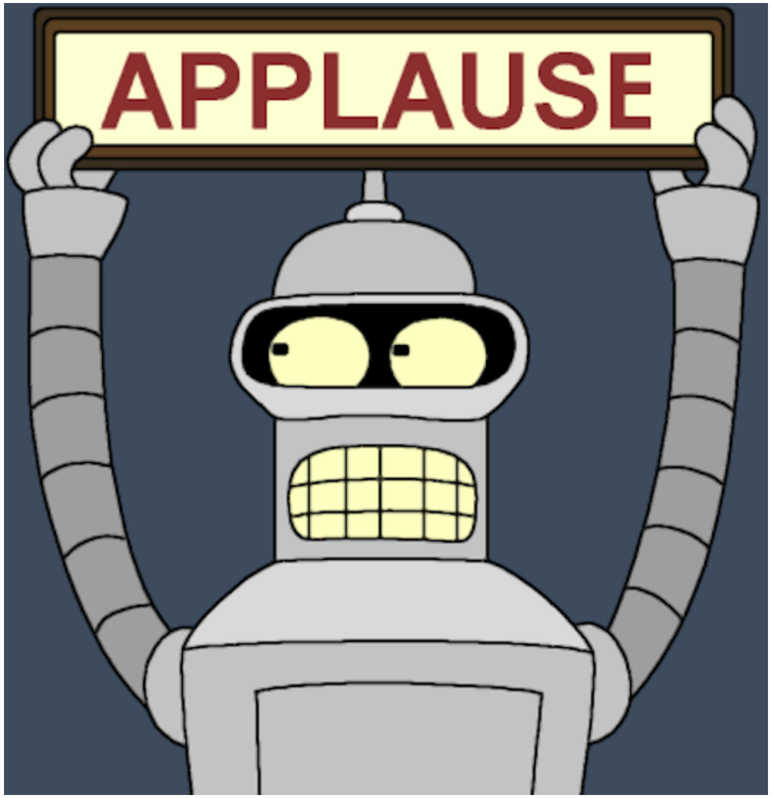
Hints

- To be able to show a model node, create display node:

  `outputModel.CreateDefaultDisplayNodes()`
- To show model intersections with a 2D slice viewer:

  `outputModel.GetDisplayNode().SetSliceIntersectionVisibility(True)`
- Remember that all the logic is implemented in the **`run`** function in the logic class, which is called in the `onApplyButton` function

Congratulations!



Thanks for attending!

Appendix

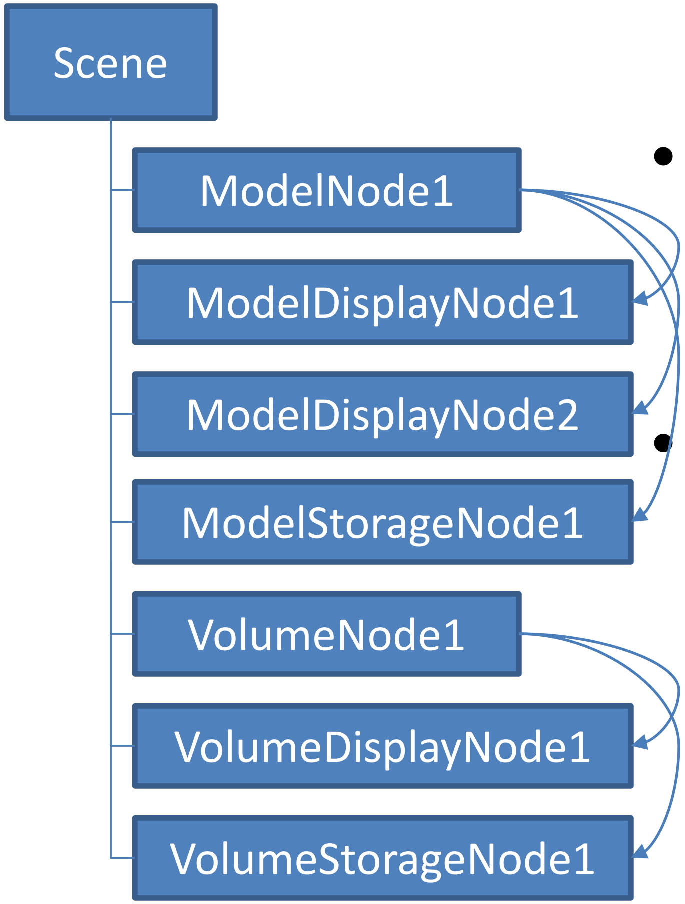
Transforms

- Slicer world coordinate system:
  RAS (right-anterior-superior)
- Get linear transform from the node to RAS:

```
nodeToRas = vtk.vtkMatrix4x4()
if node.GetTransformNodeID():
  nodeToRasNode =
    slicer.mrmlScene.GetNodeByID(node.GetTransformNodeID())
  nodeToRasNode.GetMatrixTransformToWorld(nodeToRas)
```

- Transform may be non-linear
- At least log an error if transform is present but it is ignored

Node references



- Always use this whenever a node relies on data stored in other nodes
- Specified by role name, referenced node ID, index (multiple references with the same role is allowed)
- **Saved/restored with the scene**

Not trivial: When importing a scene and a node ID is already found in the current scene, the imported node ID is automatically renamed and all references are updated

GUI design

- Qt designer can be used
- Generated UI file can be loaded to create module GUI: http://www.slicer.org/slicerWiki/index.php/Documentation/Nightly/Developers/Tutorials/PythonAndUIFile

Overview of API's accessible from python

Qt
TK
vtk Visualization Toolkit
Simple itk
NumPy
<mrml>

Qt

- Main page:
  http://www.qt.io
- Slicer uses version 4.8.6
- Features
  - User interface
  - Run-time control (signal-slot mechanism)
- Class list (bible)
  http://doc.qt.io/qt-4.8/classes.html
  Qt Assistant (desktop application)

VTK
Visualization Toolkit (by Kitware)

- Main page:
  http://www.vtk.org
- Slicer uses version 6.3.0
- Features
  - Visualization
  - Data handling
  - Simple image processing functions
- Class list (your bible when you use the API)
  http://www.vtk.org/doc/release/6.3/html/classes.html

ITK / SimpleITK
Insight Toolkit (by Kitware)

- Main page: http://www.itk.org, http://www.simpleitk.org
- Slicer uses version 4.10.0
- Features: Complex image processing functions for segmentation, registration, etc.
- Class list (bible) for SimpleITK: http://www.itk.org/SimpleITKDoxygen/html/classes.html
- Tutorial: http://www.na-mic.org/Wiki/images/a/a7/SimpleITK_with_Slicer_HansJohnson.pdf

CTK
Common Toolkit

- Main page: http://www.commontk.org
- No releases, Slicer uses trunk
- Features
  - User interface elements used in medical imaging
  - DICOM interface
- Class list (bible): http://www.commontk.org/docs/html/classes.html

NumPy

- Main page: http://www.numpy.org
- Slicer uses version 1.9.2
- Features: fundamental package for scientific computing with Python (arrays, lin. alg., Fourier, etc.)
- Reference (bible): http://docs.scipy.org/doc/numpy/reference

MRML (Slicer API)

- Features: Slicer data management and processing pipeline
- Slicer developers manual: http://www.slicer.org/slicerWiki/index.php/Documentation/Nightly/Developers
- Class list (bible): http://www.slicer.org/doc/html/classes.html