3D Slicer module programming

Beyond the basics

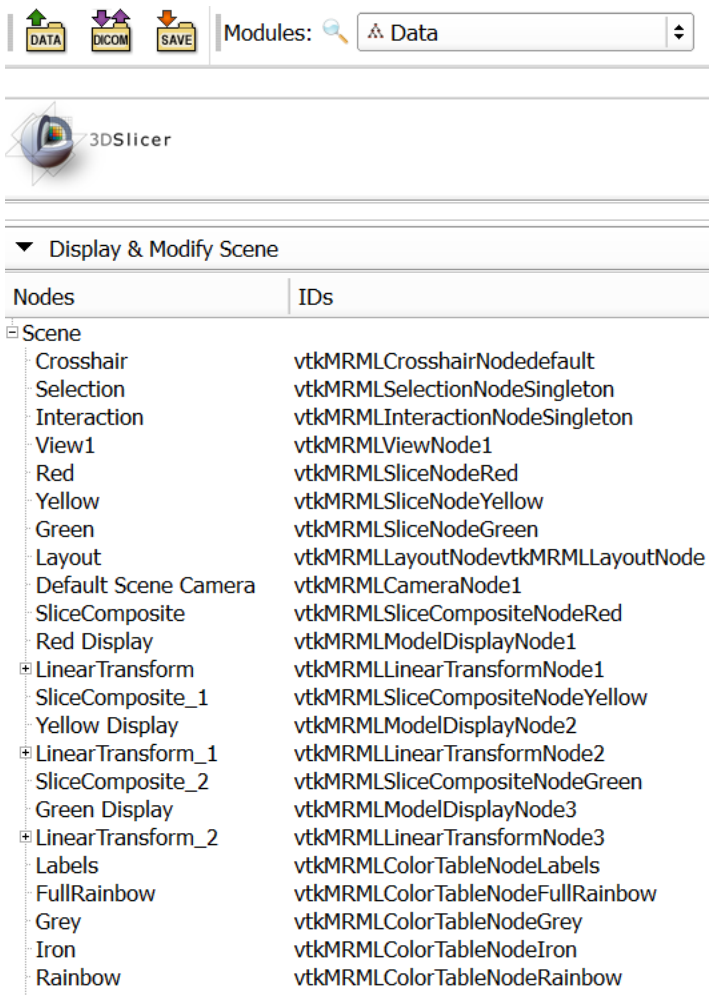
Motivation

- Can the module be used from another module (without using GUI)?
- If you save and reload the scene, are all the settings on the module user interface preserved?
- Most nodes can be transformed. Does your module work correctly if inputs or outputs are transformed?
- Does your module work correctly if the scene is closed? If the same scene is loaded twice?

If any of the answers is NO… pay attention!

Slicer data model

- Global repository for all data: MRML scene
  (MRML: Medical Reality Markup Language)
  - List of nodes, each identified by a unique string ID
  - Relationship between nodes: references, hierarchies
- **Modules do not need to know about each other!**

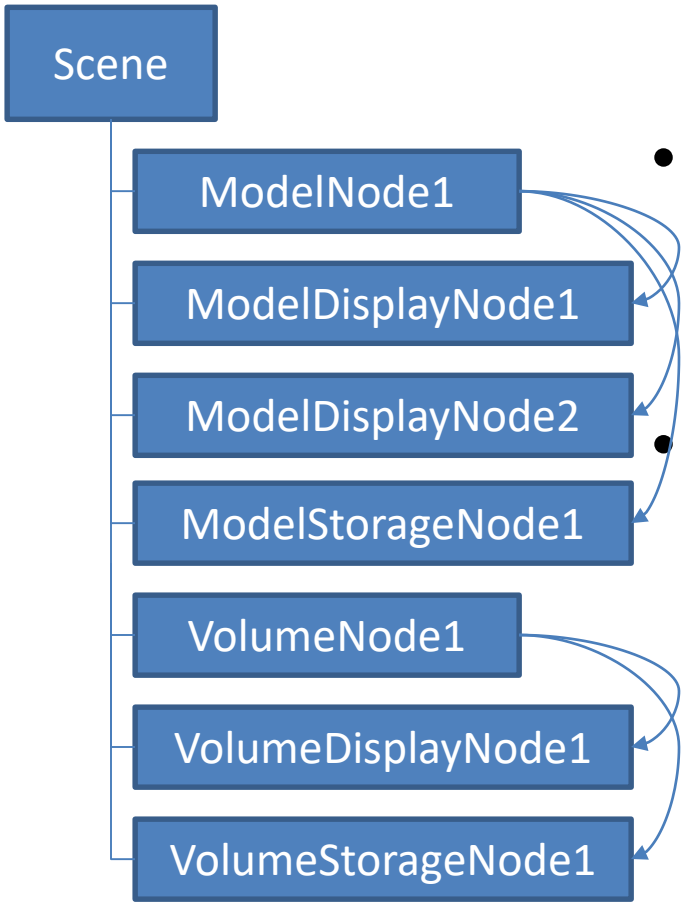


Nodes	IDs
Scene	
Crosshair	vtkMRMLCrosshairNodedefault
Selection	vtkMRMLSelectionNodeSingleton
Interaction	vtkMRMLInteractionNodeSingleton
View1	vtkMRMLViewNode1
Red	vtkMRMLSliceNodeRed
Yellow	vtkMRMLSliceNodeYellow
Green	vtkMRMLSliceNodeGreen
Layout	vtkMRMLLayoutNodevtkMRMLLayoutNode
Default Scene Camera	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 element for persistency
  - 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

Node references

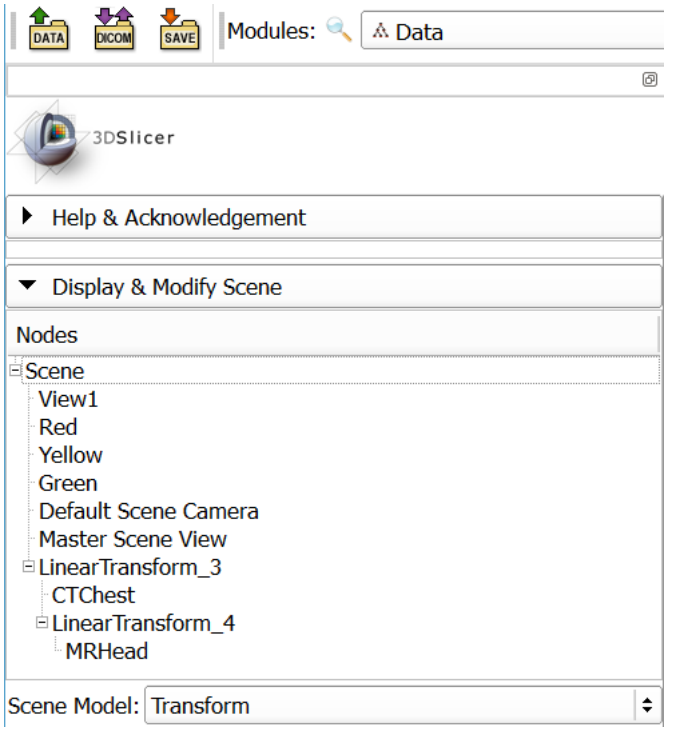


- Always use this whenever a node relies on data stored in other nodes
- Specified by role name, referenced node ID, index (multiple references with the same role is allowed)
- **Saved/restored with the scene**

  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

Node hierarchies

- Tree structure, for data **grouping**
- Same nodes can be in multiple hierarchies:
  - Transform
  - Display
  - Subject (patient/study/series)



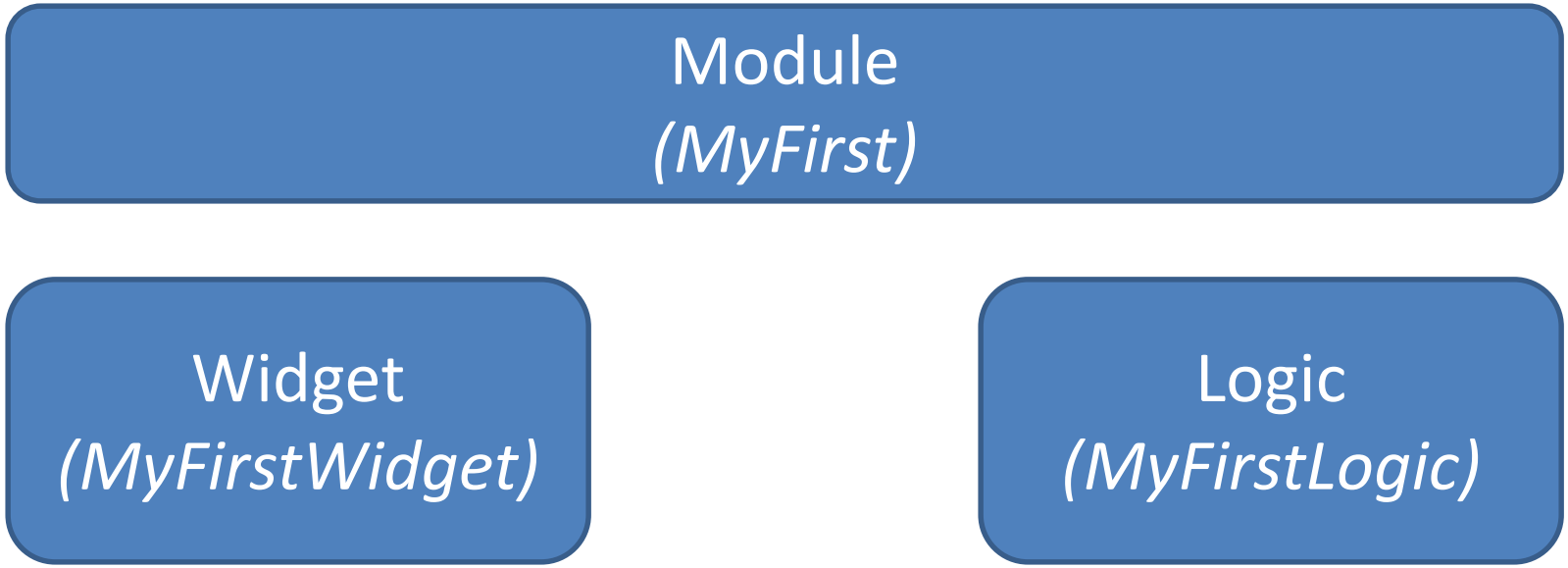
Transforms

- 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

Scripted module implementation



Module class

- Required. Only one global instance exists:

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

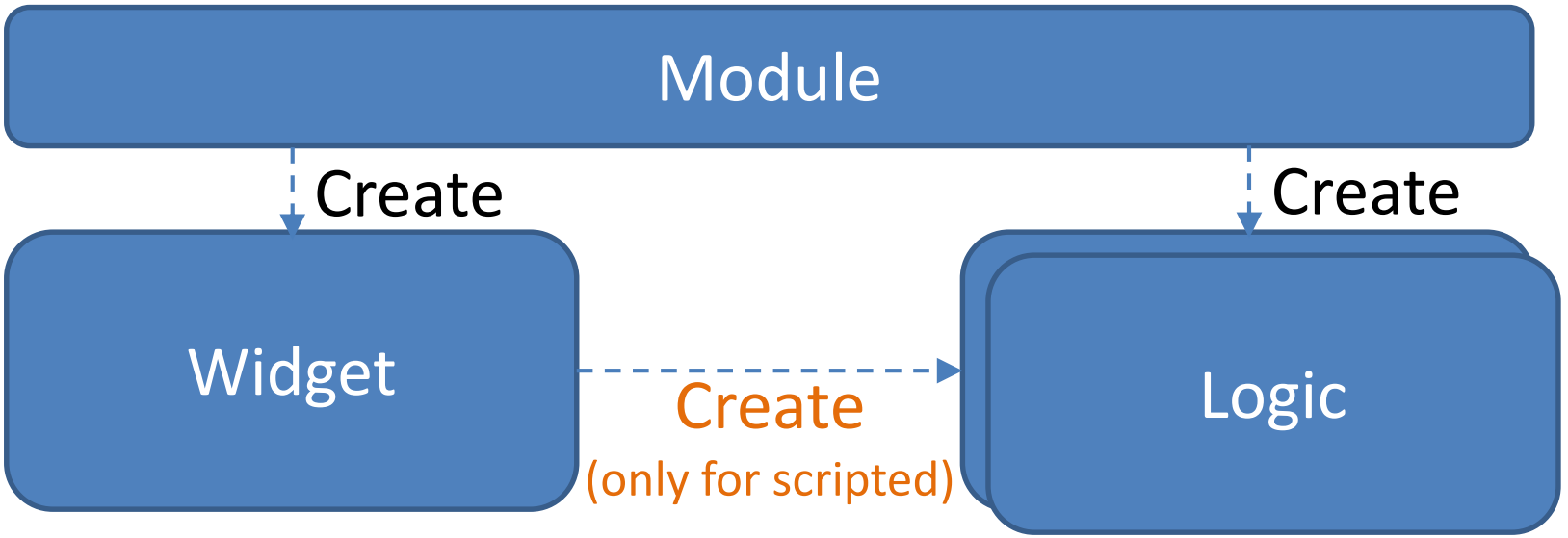
- Stores module description, icon, etc.
- Creates and holds a reference to logic and widget:
  - Loadable modules:

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

  - Python scripted modules:

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

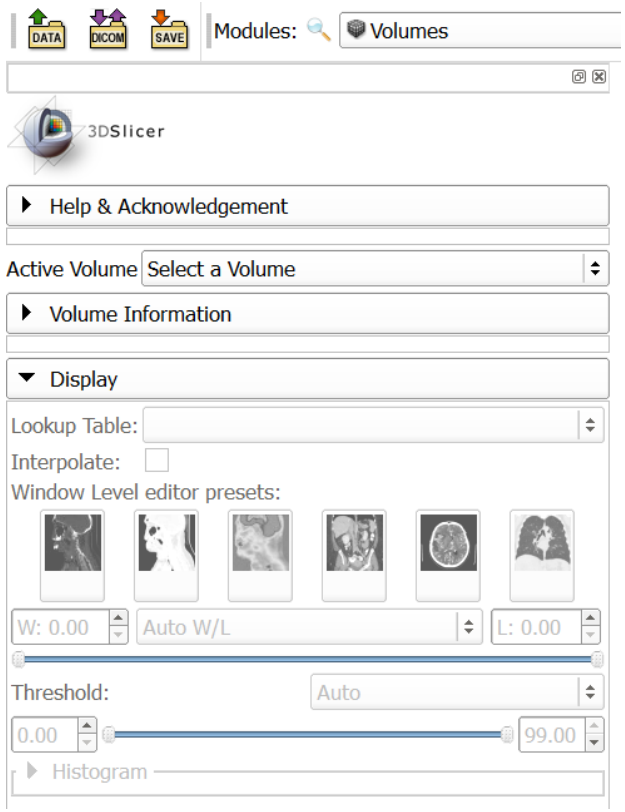
Scripted module implementation



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

Widget class

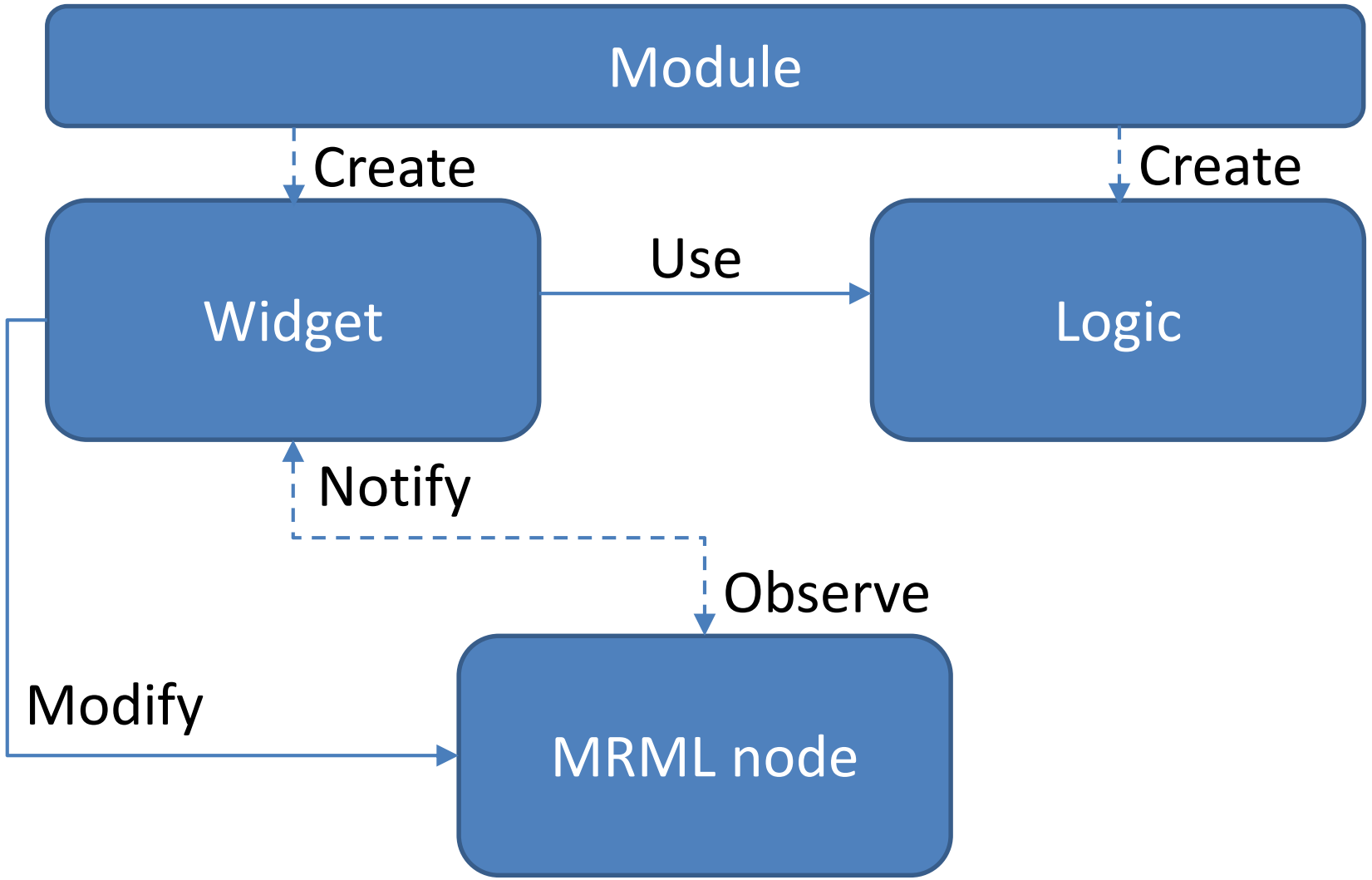
- Needed if the module has a user interface
- Typically only one global instance exists
- Defines the module's user interface
- Keeps user interface and nodes in sync (Observes MRML nodes to get change notifications)
- Launches processing methods implemented in the logic class



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 the parameter node

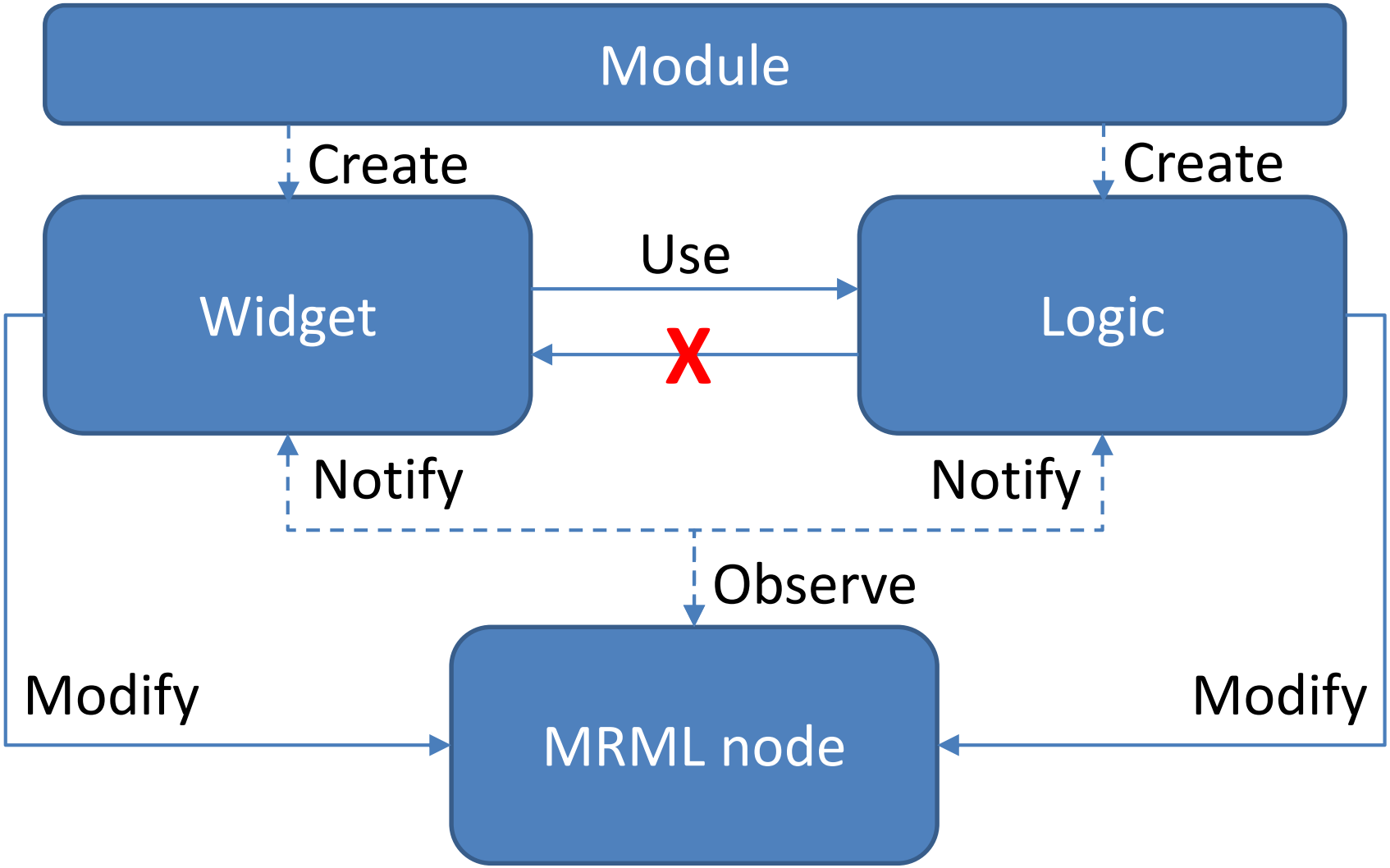
Scripted module implementation



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)

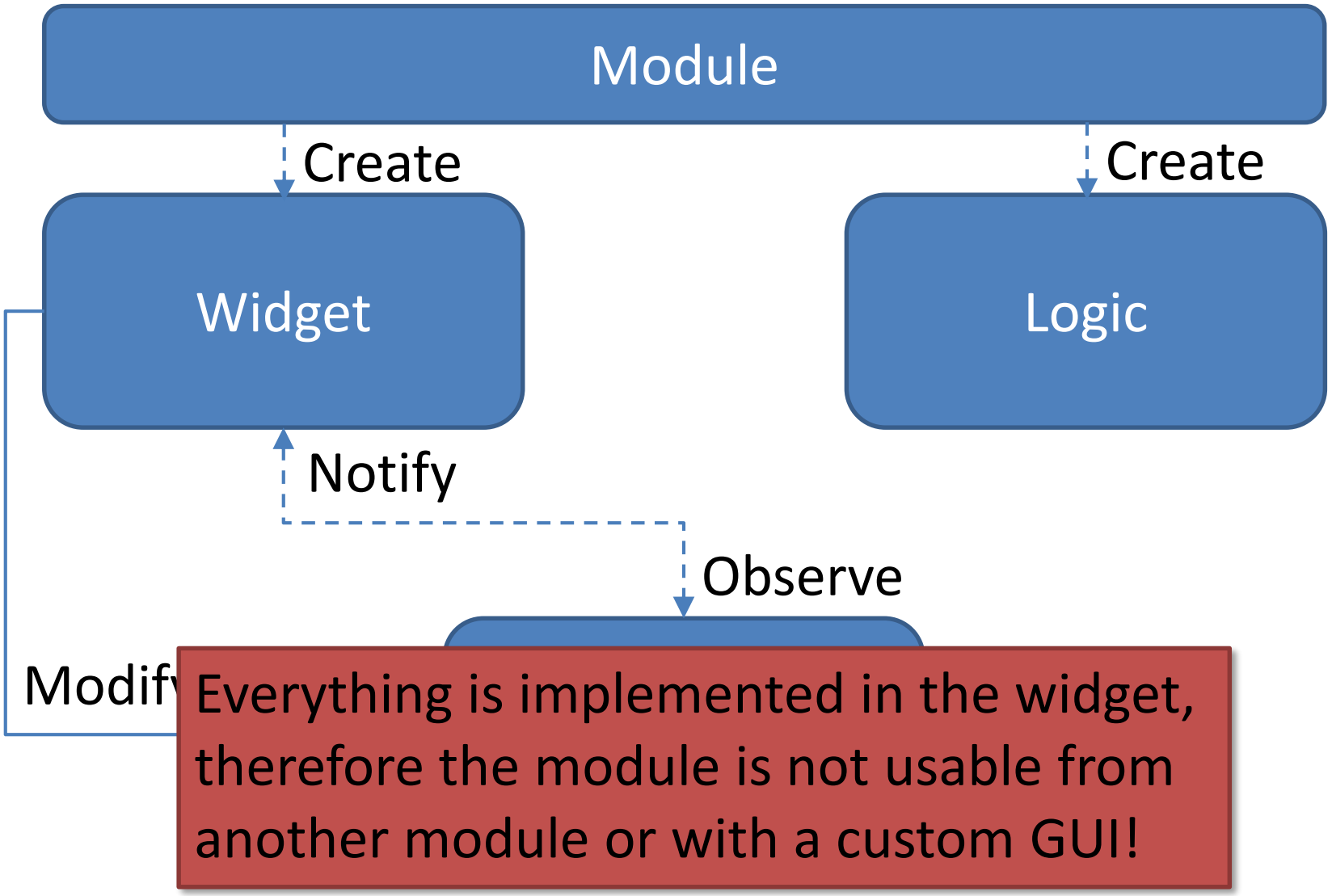
Scripted module implementation



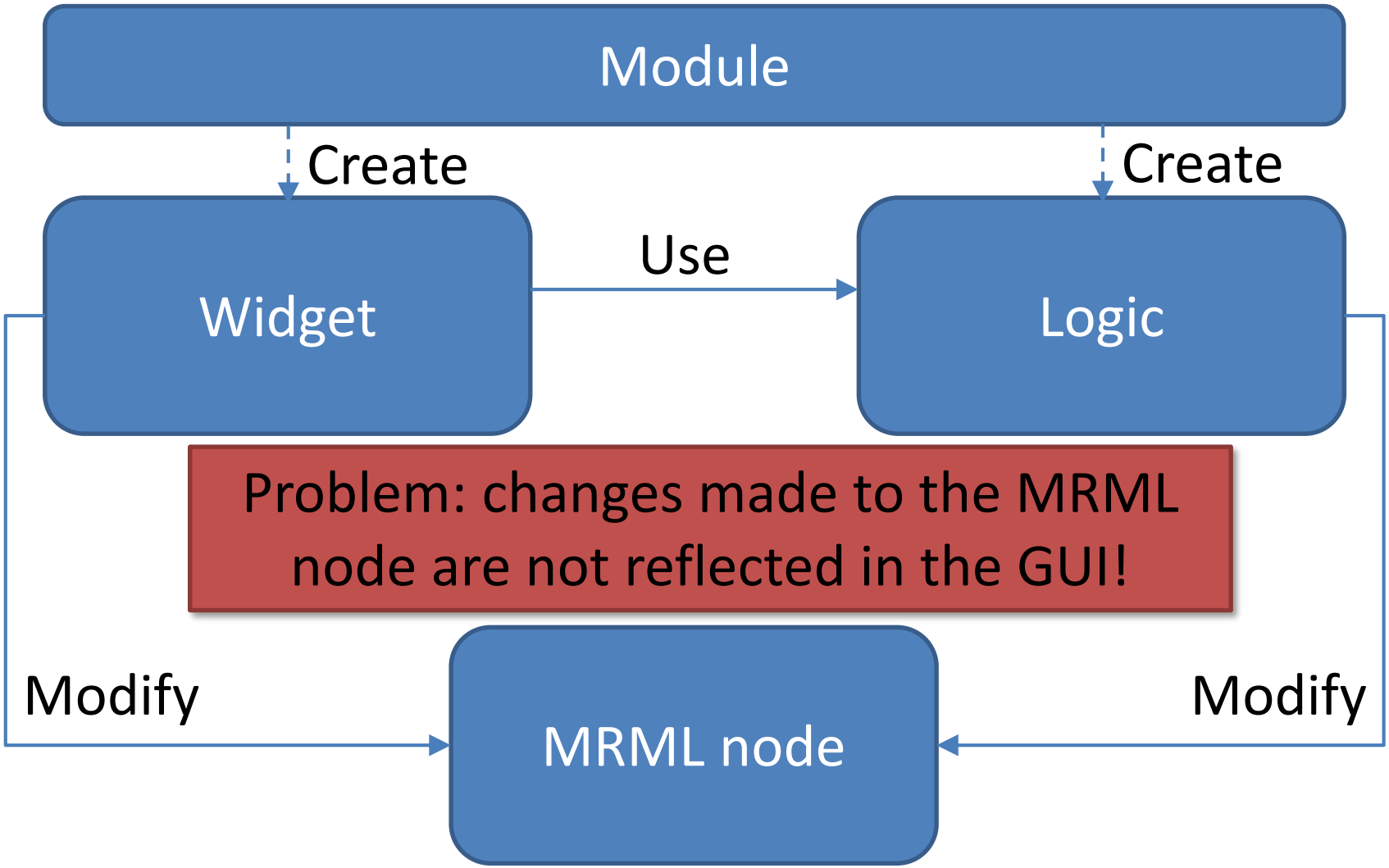
Example

https://www.assembla.com/code/slicerrt/subversion/nodes/1975/trunk/SlicerRt/sandbox/SegmentationUtils/FillRoi/FillRoi.py

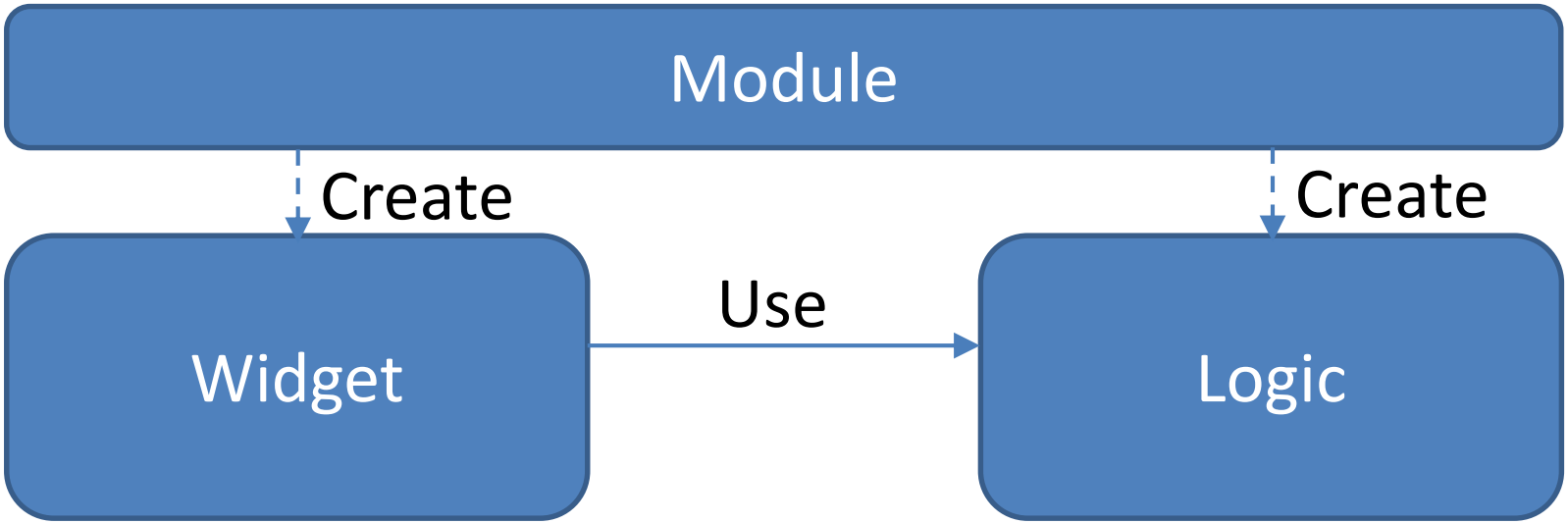
Common mistakes 1



Common mistakes 2



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!

Thanks for your attention.

Slicer data model

- MRML node
- File reader/writer
- DICOM importer/exporter
- Displayable manager
- Editor effect

- GUI from UI file: http://www.slicer.org/slicerWiki/index.php/Documentation/Nightly/Developers/Tutorials/PythonAndUIFile

Agenda

- Observers
- GetOutput/Update/GetOutputPort
- Memory management
- Scripted module
  - Module template
  - Debugging (reload&test, PyDev)
  - Slicelets