## Contents

1 Getting Started .................................................. 1
   1.1 What is Slicer? ................................................. 1
   1.2 Hardware Requirements ..................................... 1
   1.3 Installing 3D Slicer .......................................... 2
   1.4 Further Documentation ..................................... 2
   1.5 User Interface Overview ................................... 2
   1.6 Tutorials ...................................................... 2
   1.7 Modules ....................................................... 3
   1.8 Extensions .................................................... 3
   1.9 Use Cases ..................................................... 3

2 License ............................................................. 5

3 Citing ................................................................. 7
   3.1 3D Slicer as a Platform ................................... 7
   3.2 Individual Module .......................................... 8

4 Acknowledgments ................................................ 11

5 Commercial Use .................................................. 13
   5.1 Slicer’s License makes Commercial Use Available ....... 13
   5.2 Commercial Partners ....................................... 13
   5.3 Slicer Based Products ...................................... 14

6 User Interface ..................................................... 15
   6.1 Application overview ....................................... 15
   6.2 Interacting with views ..................................... 17
   6.3 Mouse & Keyboard Shortcuts ............................... 19

7 Data Loading and Saving ......................................... 21

8 Modules .............................................................. 25
   8.1 Main modules ................................................ 25
   8.2 Wizards ....................................................... 25
   8.3 Informatics .................................................... 26
   8.4 Registration .................................................. 26
   8.5 Segmentation ................................................ 26
Python Module Index 77
Index 79
Welcome to the 3D Slicer community. Here you will learn the basics of using Slicer including installing 3D Slicer, the basics of the main application GUI, how to use Slicer and where to find tutorials and more information.

1.1 What is Slicer?

3D Slicer is:

• A software platform for the analysis (including registration and interactive segmentation) and visualization (including volume rendering) of medical images and for research in image guided therapy.
• A free, open source software available on multiple operating systems: Linux, MacOSX and Windows
• Extensible, with powerful plug-in capabilities for adding algorithms and applications.

Features include:

• Multi organ: from head to toe.
• Support for multi-modality imaging including, MRI, CT, US, nuclear medicine, and microscopy.
• Bidirectional interface for devices.

**Important:** There is no restriction on use, but Slicer is **NOT** approved for clinical use and intended for research. Permissions and compliance with applicable rules are the responsibility of the user. For details on the license see here.

1.2 Hardware Requirements

3D Slicer is an open-source package that can be used on Mac, Linux and Windows. In order to run 3D Slicer your computer must have the graphics capabilities and memory to hold the original image data and process results. A 64-bit system is required. Click here more information.
1.3 Installing 3D Slicer

To install Slicer, click here.

Fig. 1: The Nightly version of 3D Slicer is updated nightly as groups of developers make changes. The Stable version of 3D Slicer is not updated nightly and is more rigorously tested.

Once downloaded, follow the instructions below to complete installation.

1.4 Further Documentation

If you’re interested in extending your knowledge, access the User Manual. See also the archives of the users mailing list. The archive is searchable so most answers to questions can be found there.

If you’re a developer looking for more information, access the Developer Manual. See also archives of the developer’s mailing list. Similar to the Users Mailing List archive, it is searchable.

1.5 User Interface Overview

3D Slicer is built on a modular architecture. The Main Application GUI is divided into six components: the Application Menu Bar, the Application Toolbar, the Module GUI Panel, the Data Probe Panel, the 2D Slice Viewers, and the 3D Viewer.

This section will introduce you to the basic functions on the main application’s GUI. If you require detailed information, visit this page.

Open 3D Slicer and load your own data or download sample data to explore. Go ahead and click around the user interface.

From the Welcome panel, you can load your own data or download sample data. Sample data is often useful for exploring the features of 3D Slicer if you don’t have data of your own.

Click on the push pin in the top left corner of each of the Slice Viewers or the 3D Viewer to see more options. In the Slice Viewers, the horizontal bar can be used to scroll through slices or select a slice. You can explore the various options using your loaded data or downloaded sample data.

1.6 Tutorials

The 3D Slicer documentation has an abundance of tutorials to help you familiarize yourself with the basics of 3D Slicer and with specific modules.

Try the Welcome Tutorial and the Data Loading and 3D Visualization Tutorial to learn the basics of using 3D Slicer.

- To learn about using Slicer for 3D Printing, visit this tutorial.
- To learn about Neurosurgical Planning with Slicer, visit this tutorial.
- To learn about DTI, visit this tutorial.

For more tutorials, visit the Tutorial page to see a comprehensive list. Additionally, visit our YouTube page for video tutorials.

If you would like to see a list of example cases with data sets and steps to achieve the same result, visit the Registration Library.
1.7 Modules

1.8 Extensions

1.9 Use Cases
The 3D Slicer software is distributed under a BSD-style open source license that is compatible with the Open Source Definition by The Open Source Initiative and contains no restrictions on use of the software.

To use Slicer, please read the 3D Slicer Software License Agreement before downloading any binary releases of the Slicer.
3.1 3D Slicer as a Platform

To acknowledge 3D Slicer as a platform, please cite the Slicer web site and the following publications when publishing work that uses or incorporates 3D Slicer:

3.1.1 Slicer 4


3.1.2 Slicer 3


3.1.3 Slicer 2


3.2 Individual Module

To acknowledge individual modules:

Fig. 1: Each module has an acknowledgment tab in the top section. Information about contributors and funding source can be found there.
Fig. 2: Additional information (including information about the underlying publications) can be typically found on the manual pages accessible through the help tab in the top section.
Acknowledgments
We invite commercial entities to use 3D Slicer.

### 5.1 Slicer’s License makes Commercial Use Available

- 3D Slicer is a Free Open Source Software distributed under a BSD style license.
- The license does not impose restrictions on the use of the software.
- 3D Slicer is NOT FDA approved. It is the users responsibility to ensure compliance with applicable rules and regulations.
- For details, please see the 3D Slicer Software License Agreement.

### 5.2 Commercial Partners

- **Isomics** uses 3D Slicer in a variety of academic and commercial research partnerships in fields such as planning and guidance for neurosurgery, quantitative imaging for clinical trials, clinical image informatics.
- **Kitware** Integral to continuing to support the 3D Slicer community, Kitware is also offering consulting services in response to the rapidly growing demand for the development of proprietary applications and commercial products based on 3D Slicer. Kitware has used 3D Slicer to rapidly prototype solutions in nearly every aspect of medical imaging and is also collaborating on the development of commercial pre-clinical and clinical products based on 3D Slicer.
- **Pixel Medical** builds on and contributes to 3D Slicer to develop innovative medical software from idea to clinical prototype to finished product, and to support academic research projects. Areas of expertise include radiation therapy, image guided therapy, virtual & augmented reality, hardware & device support, and machine learning & artificial intelligence.

Listed in alphabetical order.
5.3 Slicer Based Products

- **SonoVol** is developing a whole-body ultrasound imaging system for small animals. This start-up company arose from research in the Department of Biomedical Engineering at the University of North Carolina at Chapel Hill.

- **Xstrahl** is developing a Small Animal Radiation Research Platform (SARRP) that uses 3D Slicer as its front-end application for radiation therapy beam placement and system control.

Listed in alphabetical order.
6.1 Application overview

Slicer stores all loaded data in a data repository, called the “scene” (or Slicer scene or MRML scene). Each data set, such as an image volume, surface model, or point set, is represented in the scene as a “node”.

Slicer provides a large number “modules”, each implementing a specific set of functions for creating or manipulating data in the scene. Modules typically do not interact with each other directly: they just all operate on the same data, which is stored in the scene. Slicer package contains over 100 built-in modules and additional modules can be installed by using the Extension Manager.

6.1.1 Module Panel

This panel (located by default on the left side of the application main window) displays all the options and features that the current module offers to the user. Current module can be selected using the Module Selection toolbar.

Data Probe is located at the bottom of the module panel. It displays information about view content at the position of the mouse pointer.

6.1.2 Views

Slicer displays data in various views. The user can choose between a number of predefined layouts, which may contain slice, 3D, chart, and table views.

The Layout Toolbar provides a drop-down menu of layouts useful for many types of studies. When Slicer is exited normally, the selected layout is saved and restored next time the application is started.

6.1.3 Application Menu

- **File**: Functions for loading a previously saved scene or individual datasets of various types, and for downloading sample datasets from the internet. An option for saving scenes and data is also provided here. **Add Data** allows
loading data from files. **DICOM** module is recommended to import data from DICOM files and loading of imported DICOM data. **Save** opens the “Save Data” window, which offers a variety of options for saving all data or selected datasets.

- **Edit**: Contains an option for showing Application Settings, which allows users to customize appearance and behavior of Slicer, such as modules displayed in the toolbar, application font size, temporary directory location, location of additional Slicer modules to include.

- **View**: Functions for showing/hiding additional windows and widgets, such as **Extension Manager** for installing extensions from Slicer app store, **Error Log** for checking if the application encountered any potential errors, **Python Interactor** for getting a Python console to interact with the loaded data or modules, **show/hide toolbars**, or **switch view layout**.

### 6.1.4 Toolbar

Toolbar provides quick access to commonly used functions. Individual toolbar panels can be shown/hidden using menu: View / Toolbars section.

**Module Selection** toolbar is used for selecting the currently active “module”. The toolbar provides options for searching for module names (Ctrl + f or click on magnify glass icon) or selecting from a menu. **Module history** shows the list of recently used modules. **Arrow buttons** can be used for going back/to returning from previously used module.

**Favorite modules** toolbar contains a list of most frequently used modules. The list can be customized using menu: Edit / Application settings / Modules / Favorite Modules. Drag-and-drop modules from the Modules list to the Favorite Modules list to add a module.
6.1.5 Status bar

This panel may display application status, such as current operation in progress. Clicking the little X icons displays the Error Log window.

6.2 Interacting with views

6.2.1 View Cross-Reference

Holding down the Shift key while moving the mouse in any slice or 3D view will cause the Crosshair to move to the selected position in all views. By default, when the Crosshair is moved in any views, all slice views are scrolled to the same RAS position indexed by the mouse. This feature is useful when inspecting.

To show/hide the Crosshair position, click the crosshair icon .

To customize behavior and appearance of the Crosshair, click the “down arrow” button on the right side of the crosshair icon.

6.2.2 Mouse Modes

Slicer has two mouse modes: Transform (which allows interactive rotate, translate and zoom operations), and Place (which permits objects to be interactively placed in slice and 3D views).

The toolbar icons that switch between these mouse modes are shown from left to right above, respectively. Place Fiducial is the default place option as shown above; options to place both Ruler and Region of Interest Widgets are also available from the drop-down Place Mode menu.

**Note:** Transform mode is the default interaction mode. By default, Place mode persists for one “place” operation after the Place Mode icon is selected, and then the mode switches back to Transform. Place mode can be made persistent (useful for creating multiple fiducial points, rulers, etc.) by checking the Persistent checkbox shown rightmost in the Mouse Mode Toolbar.

6.2.3 3D View

Displays a rendered 3D view of the scene along with visual references to specify orientation and scale.

Default orientation axes: A = anterior, P = posterior, R = right, L = left, S = superior and I = inferior.

3D View Controls: The blue bar across any 3D View shows a pushpin icon on its left. When the mouse rolls over this icon, a panel for configuring the 3D View is displayed. The panel is hidden when the mouse moves away. For persistent display of this panel, just click the pushpin icon.

6.2.4 Slice View

Three default slice views are provided (with Red, Yellow and Green colored bars) in which Axial, Sagittal, Coronal or Oblique 2D slices of volume images can be displayed. Additional generic slice views have a grey colored bar and an identifying number in their upper left corner.
Slice View Controls: The colored bar across any Slice View shows a pushpin icon on its left. When the mouse rolls over this icon, a panel for configuring the slice view is displayed. The panel is hidden when the mouse moves away. For persistent display of this panel, just click the pushpin icon. For more options, click the double-arrow icon.

View Controllers module provides an alternate way of displaying these controllers in the Module Panel.
6.2.5 Chart View

6.2.6 Table View

6.3 Mouse & Keyboard Shortcuts

6.3.1 Generic shortcuts

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl + f</td>
<td>find module by name (hit Enter to select)</td>
</tr>
<tr>
<td>Ctrl + a</td>
<td>add data from file</td>
</tr>
<tr>
<td>Ctrl + o</td>
<td>save data to files</td>
</tr>
<tr>
<td>Ctrl + w</td>
<td>close scene</td>
</tr>
<tr>
<td>Ctrl + 0</td>
<td>show Error Log</td>
</tr>
<tr>
<td>Ctrl + 1</td>
<td>show Application Help</td>
</tr>
<tr>
<td>Ctrl + 2</td>
<td>show Application Settings</td>
</tr>
<tr>
<td>Ctrl + 3</td>
<td>show/hide Python Interactor</td>
</tr>
<tr>
<td>Ctrl + 4</td>
<td>show Extension Manager</td>
</tr>
<tr>
<td>Ctrl + 5</td>
<td>show/hide Module Panel</td>
</tr>
<tr>
<td>Ctrl + h</td>
<td>open default startup module (configurable in Application Settings)</td>
</tr>
</tbody>
</table>

6.3.2 Slice views

The following shortcuts are available when a slice view is active. To activate a view, click inside the view: if you do not want to change anything in the view, just activate it then do right-click without moving the mouse. Note that simply hovering over the mouse over a slice view will not activate the view.

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>right-click + drag up/down</td>
<td>zoom image in/out</td>
</tr>
<tr>
<td>Ctrl + mouse wheel</td>
<td></td>
</tr>
<tr>
<td>middle-click + drag</td>
<td>pan (translate) view</td>
</tr>
<tr>
<td>Shift + left-click + drag</td>
<td>move to previous/next slice</td>
</tr>
<tr>
<td>left arrow/right arrow</td>
<td>move crosshair in all views</td>
</tr>
<tr>
<td>b/f</td>
<td></td>
</tr>
<tr>
<td>Shift + mouse move</td>
<td>move crosshair in all views</td>
</tr>
<tr>
<td>v</td>
<td>toggle slice visibility in 3D view</td>
</tr>
<tr>
<td>r</td>
<td>reset zoom and pan to default</td>
</tr>
<tr>
<td>g</td>
<td>toggle segmentation or labelmap volume visibility</td>
</tr>
<tr>
<td>t</td>
<td>toggle foreground volume visibility</td>
</tr>
<tr>
<td>[ ]</td>
<td>use previous/next volume as background</td>
</tr>
<tr>
<td>{ ]</td>
<td>use previous/next volume as foreround</td>
</tr>
</tbody>
</table>

6.3.3 3D views

The following shortcuts are available when a 3D view is active. To activate a view, click inside the view: if you do not want to change anything in the view, just activate it then do right-click without moving the mouse. Note that simply hovering over the mouse over a slice view will not activate the view.
### Shortcut

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift + mouse move</td>
<td>move crosshair in all views</td>
</tr>
<tr>
<td>left-click + drag</td>
<td>rotate view</td>
</tr>
<tr>
<td>left arrow / right arrow</td>
<td></td>
</tr>
<tr>
<td>up arrow / down arrow</td>
<td></td>
</tr>
<tr>
<td>End or Keypad 1</td>
<td>rotate to view from anterior</td>
</tr>
<tr>
<td>Shift + End or Shift + Keypad 1</td>
<td>rotate to view from posterior</td>
</tr>
<tr>
<td>Page Down or Keypad 3</td>
<td>rotate to view from left side</td>
</tr>
<tr>
<td>Shift + Page Down Shift + Keypad 3 or</td>
<td>rotate to view from right side</td>
</tr>
<tr>
<td>Home or Keypad 7</td>
<td>rotate to view from superior</td>
</tr>
<tr>
<td>Shift + Home Shift + Keypad 7 or</td>
<td>rotate to view from inferior</td>
</tr>
<tr>
<td>right-click + drag up/down</td>
<td>zoom view in / out</td>
</tr>
<tr>
<td>Ctrl + mouse wheel</td>
<td></td>
</tr>
<tr>
<td>+/-</td>
<td></td>
</tr>
<tr>
<td>middle-click + drag</td>
<td>pan (translate) view</td>
</tr>
<tr>
<td>Shift + left-click + drag</td>
<td></td>
</tr>
<tr>
<td>Shift + left arrow / Shift + right arrow</td>
<td></td>
</tr>
<tr>
<td>Shift + up arrow / Shift + down arrow</td>
<td></td>
</tr>
<tr>
<td>Shift + Keypad 2 / Shift + Keypad 4</td>
<td></td>
</tr>
<tr>
<td>Shift + Keypad 6 / Shift + Keypad 8</td>
<td></td>
</tr>
<tr>
<td>Keypad 0 or Insert</td>
<td>reset zoom and pan, rotate to nearest standard view</td>
</tr>
</tbody>
</table>

**Note:** Simulation of shortcuts not available on your device:

- One-button mouse: instead of right-click do Ctrl + click
- Trackpad: instead of right-click do two-finger click
CHAPTER 7

Data Loading and Saving

There are two major types of data that can be loaded to Slicer:

- **DICOM**, which is a widely used and sophisticated set of standards for digital radiology. DICOM data can only be loaded through the DICOM browser, after importing to the DICOM database. The DICOM browser is accessible from the toolbar using the DICOM button. More information about DICOM can be found on the Slicer wiki.

- **Non-DICOM**, covering all types of data ranging from images and models to tables and point lists.
  - Loading can happen in two ways: drag&drop file on the Slicer window, or by using the Load Data button on the toolbar.
  - Saving happens with the Save Data toolbar button.

Data available in Slicer can be reviewed in the Data module, which can be found on the toolbar or the modules list. More details about the module can be found on the Slicer wiki.

The Data module’s default Subject hierarchy tab can show the datasets in a tree hierarchy, arranged as patient/study/series as typical in DICOM, or any other folder structure:
The Subject hierarchy view contains numerous built-in functions for all types of data. These functions can be accessed by right-clicking the node in the tree. The list of actions differs for each data type, so it is useful to explore the options.

Medical imaging data comes in various forms and representations, which may confuse people just starting in the field. The following diagram gives a brief overview about the most typical data types encountered when using Slicer, especially in a workflow that involves segmentation.
8.1 Main modules

- module_annotations
- module_data
- module_datastore
- module_dicom
- module_editor
- module_markups
- module_models
- module_sceneviews
- module_segmentations
- Segment editor
- module_transforms
- module_viewcontrollers
- module_volumerendering
- module_volumes
- module_welcometoslicer

8.2 Wizards

- module_comparevolumes
8.3 Informatics

- module_annotaions
- module_colors
- module_data
- module_dicom
- module_markups
- module_sampledata
- module_tables
- module_terminologies

8.4 Registration

- module_brainsfit
- module_landmarkregistration
- module_performmetrictest
- module_brainsresample
- module_brainsresize
- module_transforms
- Specialized:
  - module_acpctransform
  - module_brainsdemonwarp
  - module_fiducialregistration
  - module_reformat
  - module_vbrainsdemonwarp

8.5 Segmentation

- module_editor
- module_emsegment
- module_emsegmentquick
- Segment editor
- Segment statistics
- module_simpleregiongrowingsegmentation
- Specialized:
  - module_emsegmentcommandline
  - module_brainsroiauto
8.6 Quantification

- module_dataprobe
- module_labelstatistics
- module_brainslabelstats
- module_petstandarduptakevaluecomputation
  - Segment statistics

8.7 Diffusion

- module_DMRIInstall
- Import and export:
  - module_DWIConvert
- Utilities:
  - module_BRAINSDWICleanup
  - module_ResampleDTIVolume
  - module_ResampleScalarVectorDWIVolume

8.8 IGT

- module_OpenIGTLinkIF

8.9 Filtering

- module_N4ITKBiasFieldCorrection
- module_CheckerBoardFilter
- module_ExtractSkeleton
- module_HistogramMatching
- module_ImageLabelCombine
- module_SimpleFilters
- module_ThresholdScalarVolume
- module_VotingBinaryHoleFillingImageFilter
- module_IslandRemoval
- Arithmetic:
  - module_AddScalarVolumes
– module_CastScalarVolume
– module_MaskScalarVolume
– module_MultiplyScalarVolumes
– module_SubtractScalarVolumes

• Denoising:
  – module_GradientAnisotropicDiffusion
  – module_CurvatureAnisotropicDiffusion
  – module_GaussianBlurImageFilter
  – module_MedianImageFilter

• Morphology:
  – module_GrayscaleFillHoleImageFilter
  – module_GrayscaleGrindPeakImageFilter

8.10 Surface models

• module_GrayscaleModelMaker
• module_LabelMapSmoothing
• module_MergeModels
• module_ModelMaker
• module_ModelToLabelMap
• module_ProbeVolumeWithModel
• module_SurfaceToolbox

8.11 Converters

• module_CreateDICOMSeries
• module_CropVolume
• module_OrientScalarVolume
• module_VectorToScalarVolume

8.12 Endoscopy

• module_Endoscopy
8.13 Utilities

- module_BRAINSStripRotation
- module_DataStore
- DICOM Patcher
- module_ScreenCapture
- module_EMSegmentTransformToNewFormat
- BRAINS:
  - module_BRAINTransformConvert

8.14 Developer Tools

- module_Cameras
- module_EventBroker
- module_ExecutionModelTour
- module_ExtensionWizard
- DICOM plugins:
  - module_DICOMDiffusionVolumePlugin
  - module_DICOMScalarVolumePlugin

8.15 Legacy

- Converters:
  - module_BSplineToDeformationField
- Filtering:
  - module_OtsuThresholdImageFilter
  - module_ResampleScalarVolume
- Registration:
  - module_ExpertAutomatedRegistration

8.16 Testing

- module_PerformanceTests
- module_SelfTests
8.17 MultiVolume Support

- module_MultiVolumeImporter
- module_MultiVolumeExplorer
CHAPTER 9

Extensions Manager
CHAPTER 10

Settings
11.1 Python

11.1.1 freesurfer module

11.1.2 mrml module

11.1.3 saferef module

11.1.4 slicer package

Submodules

slicer.ScriptedLoadableModule module

slicer.cli module

This module is a place holder for convenient functions allowing to interact with CLI.

slicer.cli.cancel (node)

slicer.cli.createNode (cliModule, parameters=None)

    Creates a new vtkMRMLCommandLineModuleNode for a specific module, with optional parameters

slicer.cli.run (module, node=None, parameters=None, wait_for_completion=False, delete_temporary_files=True, update_display=True)

    Runs a CLI, optionally given a node with optional parameters, returning back the node (or the new one if created)
    node: existing parameter node (None by default) parameters: dictionary of parameters for cli (None by default)
    wait_for_completion: block if True (False by default) delete_temporary_files: remove temp files created during
    execution (True by default) update_display: show output nodes after completion
slicer.cli.runSync(module, node=None, parameters=None, delete_temporary_files=True, update_display=True)

Run a CLI synchronously, optionally given a node with optional parameters, returning the node (or the new one if created) node: existing parameter node (None by default) parameters: dictionary of parameters for cli (None by default) delete_temporary_files: remove temp files created during execution (True by default) update_display: show output nodes after completion

slicer.cli.setNodeParameters(node, parameters)

Sets parameters for a vtkMRMLCommandLineModuleNode given a dictionary of (parameterName, parameterValue) pairs For vectors: provide a list, tuple or comma-separated string For enumerations, provide the single enumeration value For files and directories, provide a string For images, geometry, points and regions, provide a vtkMRMLNode

slicer.logic module

slicer.slicerqt-with-tcl module

slicer.slicerqt module

slicer.testing module

slicer.testing.exitFailure(message="")
slicer.testing.exitSuccess()
slicer.testing.runUnitTest(path, testname)

slicer.util module

exception slicer.util.MRMLNodeNotFoundException

Bases: Exception

Exception raised when a requested MRML node was not found.

class slicer.util.NodeModify(node)

Bases: object

Context manager to conveniently compress mrml node modified event.

class slicer.util.VTKObservationMixin

Bases: object

addObserver(object, event, method, group='none', priority=0.0)

hasObserver(object, event, method)

observer(event, method)

removeObserver(object, event, method)

removeObservers(method=None)

slicer.util.addVolumeFromArray(narray, ijkToRAS=None, name=None, nodeClassName=None)

Create a new volume node from content of a numpy array and add it to the scene. Voxels values are deep-copied, therefore if the numpy array is modified after calling this method, voxel values in the volume node will not change.

Parameters

- **narray** – numpy array containing volume voxels.
• **ijkToRAS** – 4x4 numpy array or vtk.vtkMatrix4x4 that defines mapping from IJK to RAS coordinate system (specifying origin, spacing, directions)

• **name** – volume node name


**Returns**  created new volume node

Example:

```python
# create zero-filled volume
import numpy as np
volumeNode = slicer.util.addVolumeFromArray(np.zeros((30, 40, 50)))
```

Example:

```python
# create labelmap volume filled with voxel value of 120
import numpy as np
volumeNode = slicer.util.addVolumeFromArray(np.ones((30, 40, 50), 'int8') * 120,
np.diag([0.2, 0.2, 0.5, 1.0]), nodeClassName="vtkMRMLLabelMapVolumeNode")
```

`slicer.util.array(pattern=", index=0)`

Return the array you are “most likely to want” from the indexth MRML node that matches the pattern.

**Warning**: Meant to be used in the python console for quick debugging/testing.

More specific API should be used in scripts to be sure you get exactly what you want, such as `arrayFromVolume()`, `arrayFromModelPoints()`, and `arrayFromGridTransform()`.

`slicer.util.arrayFromGridTransform(gridTransformNode)`

Return voxel array from transform node as numpy array. Vector values are not copied. Values in the transform node can be modified by changing values in the numpy array. After all modifications has been completed, call `arrayFromGridTransformModified()`.

**Warning**: Important: memory area of the returned array is managed by VTK, therefore values in the array may be changed, but the array must not be reallocated. See `arrayFromVolume()` for details.

`slicer.util.arrayFromGridTransformModified(gridTransformNode)`

Indicate that modification of a numpy array returned by `arrayFromModelPoints()` has been completed.

`slicer.util.arrayFromMarkupsControlPoints(markupsNode, world=False)`

Return control point positions of a markups node as rows in a numpy array (of size Nx3).

**Parameters**  `world` – if set to True then the control points coordinates are returned in world coordinate system (effect of parent transform to the node is applied).

The returned array is just a copy and so any modification in the array will not affect the markup node.

To modify markup control points based on a numpy array, use `updateMarkupsControlPointsFromArray()`.

`slicer.util.arrayFromMarkupsCurvePoints(markupsNode, world=False)`

Return interpolated curve point positions of a markups node as rows in a numpy array (of size Nx3).
Parameters `world` – if set to True then the point coordinates are returned in world coordinate system (effect of parent transform to the node is applied).

The returned array is just a copy and so any modification in the array will not affect the markup node.

`slicer.util.arrayFromModelPointData (modelNode, arrayName)`

Return point data array of a model node as numpy array.

**Warning:** Important: memory area of the returned array is managed by VTK, therefore values in the array may be changed, but the array must not be reallocated. See `arrayFromVolume()` for details.

`slicer.util.arrayFromModelPoints (modelNode)`

Return point positions of a model node as numpy array. Point coordinates can be modified by modifying the numpy array. After all modifications has been completed, call `arrayFromModelPointsModified()`.

**Warning:** Important: memory area of the returned array is managed by VTK, therefore values in the array may be changed, but the array must not be reallocated. See `arrayFromVolume()` for details.

`slicer.util.arrayFromModelPointsModified (modelNode)`

Indicate that modification of a numpy array returned by `arrayFromModelPoints()` has been completed.

`slicer.util.arrayFromModelPolyIds (modelNode)`

Return poly id array of a model node as numpy array.

These ids are the following format: \[ n(0), i(0,0), i(0,1), \ldots \, i(0,n(0)), \ldots, n(j), i(j,0), \ldots \, i(j,n(j)), \ldots \] where \( n(j) \) is the number of vertices in polygon \( j \) and \( i(j,k) \) is the index into the vertex array for vertex \( k \) of poly \( j \).


Typically in Slicer \( n(j) \) will always be 3 because a model node’s polygons will be triangles.

**Warning:** Important: memory area of the returned array is managed by VTK, therefore values in the array may be changed, but the array must not be reallocated. See `arrayFromVolume()` for details.

`slicer.util.arrayFromSegment (segmentationNode, segmentId)`

`slicer.util.arrayFromSegmentBinaryLabelmap (segmentationNode, segmentId)`

Return voxel array of a segment’s binary labelmap representation as numpy array.

Voxels values are copied.

If binary labelmap is the master representation then voxel values in the volume node can be modified by changing values in the numpy array.

After all modifications have been completed, call:

```
segmentationNode.GetSegmentation().GetSegment(segmentID).Modified()
```

**Warning:** Important: memory area of the returned array is managed by VTK, therefore values in the array may be changed, but the array must not be reallocated. See `arrayFromVolume()` for details.

`slicer.util.arrayFromSegmentInternalBinaryLabelmap (segmentationNode, segmentId)`

Return voxel array of a segment’s binary labelmap representation as numpy array. Voxels values are not copied.
The labelmap containing the specified segment may be a shared labelmap containing multiple segments. To get and modify the array for a single segment, calling:

```csharp
segmentationNode->GetSegmentation() -> SeparateSegment(segmentId)
```

will transfer the segment from a shared labelmap into a new layer. Layers can be merged by calling:

```csharp
segmentationNode->GetSegmentation() -> CollapseBinaryLabelmaps()
```

If binary labelmap is the master representation then voxel values in the volume node can be modified by changing values in the numpy array. After all modifications has been completed, call:

```csharp
segmentationNode.GetSegmentation().GetSegment(segmentID).Modified()
```

**Warning:** Important: memory area of the returned array is managed by VTK, therefore values in the array may be changed, but the array must not be reallocated. See `arrayFromVolume()` for details.

`slicer.util.arrayFromTableColumn` *(tableNode, columnName)*

Return values of a table node’s column as numpy array. Values can be modified by modifying the numpy array. After all modifications has been completed, call `arrayFromTableColumnModified()`.

**Warning:** Important: memory area of the returned array is managed by VTK, therefore values in the array may be changed, but the array must not be reallocated. See `arrayFromVolume()` for details.

`slicer.util.arrayFromTableColumnModified` *(tableName, columnName)*

Indicate that modification of a numpy array returned by `arrayFromModelPoints()` has been completed.

`slicer.util.arrayFromTransformMatrix` *(transformNode, toWorld=False)*

Return 4x4 transformation matrix as numpy array.

**Parameters**

- toWorld – if set to True then the transform to world coordinate system is returned (effect of parent transform to the node is applied), otherwise transform to parent transform is returned.

The returned array is just a copy and so any modification in the array will not affect the transform node.

To set transformation matrix from a numpy array, use `updateTransformMatrixFromArray()`.

`slicer.util.arrayFromVTKMatrix` *(vmatrix)*

Return vtkMatrix4x4 or vtkMatrix3x3 elements as numpy array. The returned array is just a copy and so any modification in the array will not affect the input matrix. To set VTK matrix from a numpy array, use `vtkMatrixFromArray()` or `updateVTKMatrixFromArray()`.

`slicer.util.arrayFromVolume` *(volumeNode)*

Return voxel array from volume node as numpy array.

Voxels values are not copied. Voxel values in the volume node can be modified by changing values in the numpy array. After all modifications has been completed, call `arrayFromVolumeModified()`.

**Warning:** Memory area of the returned array is managed by VTK, therefore values in the array may be changed, but the array must not be reallocated (change array size, shallow-copy content from other array most likely causes application crash). To allow arbitrary numpy operations on a volume array:
1. Make a deep-copy of the returned VTK-managed array using \texttt{numpy.copy()}.
2. Perform any computations using the copied array.
3. Write results back to the image data using \texttt{updateVolumeFromArray()}.

\texttt{slicer.util.arrayFromVolumeModified(volumeNode)}

Indicate that modification of a numpy array returned by \texttt{arrayFromVolume()} has been completed.

\texttt{slicer.util.childWidgetVariables(widget)}

Get child widgets as attributes of an object.

Each named child widget is accessible as an attribute of the returned object, with the attribute name matching the child widget name. This function provides convenient access to widgets in a loaded UI file.

Example:

```python
uiWidget = slicer.util.loadUI(myUiFilePath)
s = slicer.util.childWidgetVariables(uiWidget)
s.inputSelector.setMRMLScene(slicer.mrmlScene)
s.outputSelector.setMRMLScene(slicer.mrmlScene)
```

\texttt{slicer.util.clickAndDrag(widget, button='Left', start=(10, 10), end=(10, 40), steps=20, modifiers=[])}

Send synthetic mouse events to the specified widget (qMRMLSliceWidget or qMRMLThreeDView)

Parameters

- \texttt{button} – “Left”, “Middle”, “Right”, or “None”
- \texttt{start} : window coordinates for action
- \texttt{end} : window coordinates for action
- \texttt{steps} – number of steps to move in, if <2 then mouse jumps to the end position
- \texttt{modifiers} – list containing zero or more of “Shift” or “Control”

\textbf{Hint:} For generating test data you can use this snippet of code:

```python
layoutManager = slicer.app.layoutManager()
threeDView = layoutManager.threeDWidget(0).threeDView()
style = threeDView.interactorStyle()
interactor = style.GetInteractor()

def onClick(caller, event):
    print(interactor.GetEventPosition())

interactor.AddObserver(vtk.vtkCommand.LeftButtonPressEvent, onClick)
```

\texttt{slicer.util.computeChecksum(algo, filePath)}

Compute digest of \texttt{filePath} using \texttt{algo}.

Supported hashing algorithms are SHA256, SHA512, and MD5.

It internally reads the file by chunk of 8192 bytes.

\textbf{Raises}

- \texttt{ValueError} – if \texttt{algo} is unknown.
- \texttt{IOError} – if \texttt{filePath} does not exist.
slicer.util.confirmOkCancelDisplay (text, windowTitle=None, parent=None, **kwargs)
    Display an confirmation popup. Return if confirmed with OK.

slicer.util.confirmRetryCloseDisplay (text, windowTitle=None, parent=None, **kwargs)
    Display an confirmation popup. Return if confirmed with Retry.

slicer.util.confirmYesNoDisplay (text, windowTitle=None, parent=None, **kwargs)
    Display an confirmation popup. Return if confirmed with Yes.

slicer.util.createProgressDialog (parent=None, value=0, maximum=100, labelText='", windowTitle='Processing...', **kwargs)
    Display a modal QProgressDialog.

    Go to QProgressDialog documentation to learn about the available keyword arguments.

Examples:

```python
# Prevent progress dialog from automatically closing
progressbar = createProgressIndicator(autoClose=False)

# Update progress value
progressbar.value = 50

# Update label text
progressbar.labelText = "processing XYZ"
```

slicer.util.delayDisplay (message, autoCloseMsec=1000)
    Display an information message in a popup window for a short time.

    If autoCloseMsec < 0 then the window is not closed until the user clicks on it

    If 0 <= autoCloseMsec < 400 then only slicer.app.processEvents() is called.

    If autoCloseMsec >= 400 then the window is closed after waiting for autoCloseMsec milliseconds

slicer.util.downloadAndExtractArchive (url, archiveFilePath, outputDir, expectedNumberOfExtractedFiles=None, numberOfTrials=3, checksum=None)
    Downloads an archive from url as archiveFilePath, and extracts it to outputDir.

    This combined function tests the success of the download by the extraction step, and re-downloads if extraction failed.

    If specified, the checksum is used to verify that the downloaded file is the expected one. It must be specified as <algo>:<digest>. For example, SHA256:cc211f0dfd9a05ca3841ce1141b292898b2dd2d2f08286affadf823a7e58df93.

slicer.util.downloadFile (url, targetFilePath, checksum=None, reDownloadIfChecksumInvalid=True)
    Download url to local storage as targetFilePath

    Target file path needs to indicate the file name and extension as well

    If specified, the checksum is used to verify that the downloaded file is the expected one. It must be specified as <algo>:<digest>. For example, SHA256:cc211f0dfd9a05ca3841ce1141b292898b2dd2d2f08286affadf823a7e58df93.

slicer.util.errorDisplay (text, windowTitle=None, parent=None, standardButtons=None, **kwargs)
    Display an error popup.

slicer.util.exit (status=0)
    Exits the application with the specified exit code. The method does not stops the process immediately but lets
pending events to be processed. If exit() is called again while processing pending events, the error code will be overwritten.

To make the application exit immediately, this code can be used. Note that forcing the application to exit may result in improperly released files and other resources.

```python
import sys
sys.exit(status)
```

`slicer.util.extractAlgoAndDigest(checksum)`  
Given a checksum string formatted as `<algo>:<digest>` returns the tuple `(algo, digest)`.  
`<algo>` is expected to be `SHA256`, `SHA512`, or `MD5`. `<digest>` is expected to be the full length hexadecimal digest.  

**Raises** `ValueError` – if checksum is incorrectly formatted.

`slicer.util.extractArchive(archiveFilePath, outputDir, expectedNumberOfExtractedFiles=None)`  
Extract file `archiveFilePath` into folder `outputDir`.  
Number of expected files unzipped may be specified in `expectedNumberOfExtractedFiles`. If folder contains the same number of files as expected (if specified), then it will be assumed that unzipping has been successfully done earlier.

`slicer.util.findChild(widget, name)`  
Convenience method to access a widget by its name. A `RuntimeError` exception is raised if the widget with the given name does not exist.

`slicer.util.findChildren(widget=None, name='', text='', title='', className='')`  
Return a list of child widgets that meet all the given criteria. If no criteria are provided, the function will return all widgets descendants. If no widget is provided, `slicer.util.mainWindow()` is used.  
:param widget: parent widget where the widgets will be searched  
:param name: name attribute of the widget  
:param text: text attribute of the widget  
:param title: title attribute of the widget  
:param className: className() attribute of the widget  
:return: list with all the widgets that meet all the given criteria.

`slicer.util.forceRenderAllViews()`  
Force rendering of all views.

`slicer.util.getFilesInDirectory(directory, absolutePath=True)`  
Collect all files in a directory and its subdirectories in a list.

`slicer.util.getFirstNodeByClassByName(className, name, scene=None)`  
Return the first node in the scene that matches the specified node name and node class.  
Optionally specify a classname that must also match.

`slicer.util.getFirstNodeByName(name, className=None)`  
Get the first MRML node that name starts with the specified name.

`slicer.util.getModule(moduleName)`  
`slicer.util.getModuleGui(module)`  
`slicer.util.getNewModuleGui(module)`  
`slicer.util.getNode(pattern='*', index=0, scene=None)`  
Return the indexth node where name or id matches `pattern`.  
By default, `pattern` is a wildcard and it returns the first node associated with `slicer.mrmlScene`.  

**Raises** `MRMLNodeNotFoundException` – if no node is found that matches the specified pattern.
slicer.util.getNodes(pattern='*', scene=None, useLists=False)

Return a dictionary of nodes where the name or id matches the pattern.

By default, pattern is a wildcard and it returns all nodes associated with slicer.mrmlScene.
If multiple nodes share the same name, using useLists=False (default behavior) returns only the last node with that name. If useLists=True, it returns a dictionary of lists of nodes.

slicer.util.getNodesByClass(className, scene=None)

Return all nodes in the scene of the specified class.

slicer.util.importClassesFromDirectory(directory, dest_module_name, type_info, filematch='*')

slicer.util.importModuleObjects(from_module_name, dest_module_name, type_info)

Import object of type ‘type_info’ (str or type) from module identified by ‘from_module_name’ into the module identified by ‘dest_module_name’.

slicer.util.importQtClassesFromDirectory(directory, dest_module_name, filematch='*')

slicer.util.importVTKClassesFromDirectory(directory, dest_module_name, filematch='*')

slicer.util.infoDisplay(text, windowTitle=None, parent=None, standardButtons=None, **kwargs)

Display popup with info message.

slicer.util.launchConsoleProcess(args, useStartupEnvironment=True, cwd=None)

Launch a process. Hiding the console and captures the process output. The console window is hidden when running on Windows.

:param args: executable name, followed by command-line arguments
:param useStartupEnvironment: launch the process in the original environment as the original Slicer process
:param cwd: current working directory
:return: process object.

slicer.util.loadAnnotationFiducial(filename, returnNode=False)

Load node from file.

Parameters

- **filename** – full path of the file to load.
- **returnNode** – Deprecated.

Returns

loaded node (if multiple nodes are loaded then a list of nodes). If returnNode is True then a status flag and loaded node are returned.

slicer.util.loadAnnotationROI(filename, returnNode=False)

Load node from file.

Parameters

- **filename** – full path of the file to load.
- **returnNode** – Deprecated.

Returns

loaded node (if multiple nodes are loaded then a list of nodes). If returnNode is True then a status flag and loaded node are returned.

slicer.util.loadAnnotationRuler(filename, returnNode=False)

Load node from file.

Parameters

- **filename** – full path of the file to load.
- **returnNode** – Deprecated.
Returns loaded node (if multiple nodes are loaded then a list of nodes). If returnNode is True then a status flag and loaded node are returned.

`slicer.util.loadColorTable(filename, returnNode=False)`
Load node from file.

Parameters
- `filename` – full path of the file to load.
- `returnNode` – Deprecated.

Returns loaded node (if multiple nodes are loaded then a list of nodes). If returnNode is True then a status flag and loaded node are returned.

`slicer.util.loadFiberBundle(filename, returnNode=False)`
Load node from file.

Parameters
- `filename` – full path of the file to load.
- `returnNode` – Deprecated.

Returns loaded node (if multiple nodes are loaded then a list of nodes). If returnNode is True then a status flag and loaded node are returned.

`slicer.util.loadFiducialList(filename, returnNode=False)`
Load node from file.

Parameters
- `filename` – full path of the file to load.
- `returnNode` – Deprecated.

Returns loaded node (if multiple nodes are loaded then a list of nodes). If returnNode is True then a status flag and loaded node are returned.

`slicer.util.loadLabelVolume(filename, properties={}, returnNode=False)`
Load node from file.

Parameters
- `filename` – full path of the file to load.
- `returnNode` – Deprecated.

Returns loaded node (if multiple nodes are loaded then a list of nodes). If returnNode is True then a status flag and loaded node are returned.

`slicer.util.loadMarkupsClosedCurve(filename)`
Load node from file.

Parameters `filename` – full path of the file to load.

Returns loaded node (if multiple nodes are loaded then a list of nodes).

`slicer.util.loadMarkupsCurve(filename)`
Load node from file.

Parameters `filename` – full path of the file to load.

Returns loaded node (if multiple nodes are loaded then a list of nodes).

`slicer.util.loadMarkupsFiducialList(filename, returnNode=False)`
Load node from file.
Parameters

- **filename** – full path of the file to load.
- **returnNode** – Deprecated.

Returns loaded node (if multiple nodes are loaded then a list of nodes). If returnNode is True then a status flag and loaded node are returned.

`slicer.util.loadModel` *(filename, returnNode=False)*

Load node from file.

Parameters

- **filename** – full path of the file to load.
- **returnNode** – Deprecated.

Returns loaded node (if multiple nodes are loaded then a list of nodes). If returnNode is True then a status flag and loaded node are returned.

`slicer.util.loadNodeFromFile` *(filename, filetype, properties={}, returnNode=False)*

Load node into the scene from a file.

Parameters

- **filename** – full path of the file to load.
- **filetype** – specifies the file type, which determines which IO class will load the file.
- **properties** – map containing additional parameters for the loading.
- **returnNode** – Deprecated. If set to true then the method returns status flag and node instead of signalling error by throwing an exception.

Returns loaded node (if multiple nodes are loaded then a list of nodes). If returnNode is True then a status flag and loaded node are returned.

`slicer.util.loadNodesFromFile` *(filename, filetype, properties={}, returnNode=False)*

Load nodes into the scene from a file.

It differs from `loadNodeFromFile` in that it returns loaded node(s) in an iterator.

Parameters

- **filename** – full path of the file to load.
- **filetype** – specifies the file type, which determines which IO class will load the file.
- **properties** – map containing additional parameters for the loading.

Returns loaded node(s) in an iterator object.

`slicer.util.loadScalarOverlay` *(filename, modelNodeID, returnNode=False)*

Load node from file.

Parameters

- **filename** – full path of the file to load.
- **returnNode** – Deprecated.

Returns loaded node (if multiple nodes are loaded then a list of nodes). If returnNode is True then a status flag and loaded node are returned.

`slicer.util.loadScene` *(filename, properties={})*

Load node from file.
Parameters

- `filename` – full path of the file to load.
- `returnNode` – Deprecated.

Returns loaded node (if multiple nodes are loaded then a list of nodes). If `returnNode` is True then a status flag and loaded node are returned.

`slicer.util.loadSegmentation(filename, returnNode=False)`

Load node from file.

Parameters

- `filename` – full path of the file to load.
- `returnNode` – Deprecated.

Returns loaded node (if multiple nodes are loaded then a list of nodes). If `returnNode` is True then a status flag and loaded node are returned.

`slicer.util.loadShaderProperty(filename, returnNode=False)`

Load node from file.

Parameters

- `filename` – full path of the file to load.
- `returnNode` – Deprecated.

Returns loaded node (if multiple nodes are loaded then a list of nodes). If `returnNode` is True then a status flag and loaded node are returned.

`slicer.util.loadText(filename)`

Load node from file.

Parameters

- `filename` – full path of the text file to load.

Returns loaded text node.

`slicer.util.loadTransform(filename, returnNode=False)`

Load node from file.

Parameters

- `filename` – full path of the file to load.
- `returnNode` – Deprecated.

Returns loaded node (if multiple nodes are loaded then a list of nodes). If `returnNode` is True then a status flag and loaded node are returned.

`slicer.util.loadUI(path)`

Load UI file `path` and return the corresponding widget.

Raises `RuntimeError` – if the UI file is not found or if no widget was instantiated.

`slicer.util.loadVolume(filename, properties={}, returnNode=False)`

Load node from file.

Parameters

- `filename` – full path of the file to load.
- `properties` –
  - name: this name will be used as node name for the loaded volume
- labelmap: interpret volume as labelmap
- singleFile: ignore all other files in the directory
- center: ignore image position
- discardOrientation: ignore image axis directions
- autoWindowLevel: compute window/level automatically
- show: display volume in slice viewers after loading is completed
- fileNames: list of filenames to load the volume from

* `returnNode` - Deprecated.

**Returns** loaded node (if multiple nodes are loaded then a list of nodes). If `returnNode` is True then a status flag and loaded node are returned.

```python
slicer.util.logProcessOutput(proc)
Continuously write process output to the application log and the Python console. :param proc: process object.
```

```python
slicer.util.lookupTopLevelWidget(objectName, verbose=True)
Loop over all top level widget associated with `slicer.app` and return the one matching `objectName`
```

```python
slicer.util.mainWindow(verbose=True)
```

```python
slicer.util.messageBox(text, parent=None, **kwargs)
Displays a messagebox.

ctkMessageBox is used instead of a default qMessageBox to provide “Don’t show again” checkbox.

For example:
```
```python
slice.util.messageBox("Some message", dontShowAgainSettingsKey = "MainWindow/->DontShowSomeMessage")
```

```python
slicer.util.moduleNames()
```

```python
slicer.util.modulePath(moduleName)
```

```python
slicer.util.moduleSelector()
```

```python
slicer.util.openAddColorTableDialog()
```

```python
slicer.util.openAddDataDialog()
```

```python
slicer.util.openAddFiberBundleDialog()
```

```python
slicer.util.openAddFiducialDialog()
```

```python
slicer.util.openAddMarkupsDialog()
```

```python
slicer.util.openAddModelDialog()
```

```python
slicer.util.openAddScalarOverlayDialog()
```

```python
slicer.util.openAddSegmentationDialog()
```

```python
slicer.util.openAddShaderPropertyDialog()
```

```python
slicer.util.openAddTransformDialog()
```

```python
slicer.util.openAddVolumeDialog()
```

```python
slicer.util.openSaveDataDialog()
```
slicer.util.pip_install(req)
Install python packages. Currently, the method simply calls python -m pip install but in the future further checks, optimizations, user confirmation may be implemented, therefore it is recommended over to use this method call instead of a plain pip install. :param req: requirement specifier, same format as used by pip (https://docs.python.org/3/installing/index.html)

Example: calling from Slicer GUI

```
pip_install("tensorflow keras scikit-learn ipywidgets")
```

Example: calling from PythonSlicer console

```
from slicer.util import pip_install
pip_install("tensorflow")
```

slicer.util.plot(narray, xColumnIndex=-1, columnNames=None, title=None, show=True, nodes=None)
Create a plot from a numpy array that contains two or more columns.

**Parameters**

- **narray** – input numpy array containing data series in columns.
- **xColumnIndex** – index of column that will be used as x axis. If it is set to negative number (by default) then row index will be used as x coordinate.
- **columnNames** – names of each column of the input array.
- **title** – title of the chart. Plot node names are set based on this value.
- **nodes** – plot chart, table, and list of plot series nodes. Specified in a dictionary, with keys: ‘chart’, ‘table’, ‘series’. Series contains a list of plot series nodes (one for each table column). The parameter is used both as an input and output.

**Returns** plot chart node. Plot chart node provides access to chart properties and plot series nodes.

**Example 1**: simple plot

```
# Get sample data
import numpy as np
import SampleData
volumeNode = SampleData.downloadSample("MRHead")

# Create new plot
histogram = np.histogram(arrayFromVolume(volumeNode), bins=50)
chartNode = plot(histogram, xColumnIndex = 1)

# Change some plot properties
chartNode.SetTitle("My histogram")
chartNode.GetNthPlotSeriesNode(0).SetPlotType(slicer.vtkMRMLPlotSeriesNode.PlotTypeScatterBar)
```

**Example 2**: plot with multiple updates

```
# Get sample data
import numpy as np
import SampleData
volumeNode = SampleData.downloadSample("MRHead")

# Create variable that will store plot nodes (chart, table, series)
plotNodes = {}
```

(continues on next page)
# Create new plot
histogram = np.histogram(arrayFromVolume(volumeNode), bins=80)
plot(histogram, xColumnIndex = 1, nodes = plotNodes)

# Update plot
histogram = np.histogram(arrayFromVolume(volumeNode), bins=40)
plot(histogram, xColumnIndex = 1, nodes = plotNodes)

slicer.util.pythonShell(verbos=True)
slicer.util.quit()
slicer.util.reloadScriptedModule(moduleName)
    Generic reload method for any scripted module.
    The function performs the following:
    • Ensure sys.path includes the module path and use imp.load_module to load the associated script.
    • For the current module widget representation:
      – Hide all children widgets
      – Call cleanup() function and disconnect ScriptedLoadableModuleWidget.onModuleAboutToBeUnloaded
      – Remove layout items
    • Instantiate new widget representation
    • Call setup() function
    • Update slicer.modules.<moduleName>Widget attribute

slicer.util.resetSliceViews()
    Reset focal view around volumes

slicer.util.resetThreeDViews()
    Reset focal view around volumes

slicer.util.restart()
slicer.util.saveNode(node, filename, properties={})
    Save 'node' data into 'filename'.
    It is the user responsibility to provide the appropriate file extension.
    User has also the possibility to overwrite the fileType internally retrieved using method 'qSlicerCoreIOManager::fileWriterFileType(vtkObject*)'. This can be done by specifying a 'fileType' attribute to the optional 'properties' dictionary.

slicer.util.saveScene(filename, properties={})
    Save the current scene.
    Based on the value of 'filename', the current scene is saved either as a MRML file, MRB file or directory.
    If filename ends with '.mrml', the scene is saved as a single file without associated data.
    If filename ends with '.mrb', the scene is saved as a MRML bundle (Zip archive with scene and data files).
    In every other case, the scene is saved in the directory specified by 'filename'. Both MRML scene file and data will be written to disk. If needed, directories and sub-directories will be created.

slicer.util.selectModule(module)
slicer.util.selectedModule()
	slicer.util.setApplicationLogoVisible(visible)
    Show/hide application logo at the top of module panel.
	slicer.util.setMenuBarsVisible(visible, ignore=None)
    Show/hide all menu bars, except those listed in ignore list.
	slicer.util.setModuleHelpSectionVisible(visible)
    Show/hide Help section at the top of module panel.
	slicer.util.setModulePanelTitleVisible(visible)
    Show/hide module panel title bar at the top of module panel. If the title bar is not visible then it is not possible
to drag and dock the module panel to a different location.
	slicer.util.setPythonConsoleVisible(visible)
    Show/hide Python console.
	slicer.util.setSliceViewerLayers(background='keep-current', foreground='keep-current', label='keep-current', foregroundOpacity=None, labelOpacity=None, fit=False)
    Set the slice views with the given nodes.
    If node ID is not specified (or value is ‘keep-current’) then the layer will not be modified.

    Parameters
    • background – node or node ID to be used for the background layer
    • foreground – node or node ID to be used for the foreground layer
    • label – node or node ID to be used for the label layer
    • foregroundOpacity – opacity of the foreground layer
    • labelOpacity – opacity of the label layer
    • fit – fit slice views to their content (position&zoom to show all visible layers)
	slicer.util.setStatusBarVisible(visible)
    Show/hide status bar
	slicer.util.setToolbarsVisible(visible, ignore=None)
    Show/hide all existing toolbars, except those listed in ignore list.
	slicer.util.setViewControllersVisible(visible)
    Show/hide view controller toolbar at the top of slice and 3D views
	slicer.util.settingsValue(key, default, converter=<function <lambda>>, settings=None)
    Return settings value associated with key if it exists or the provided default otherwise.

    settings parameter is expected to be a valid `qt.Settings` object.
	slicer.util.showStatusMessage(message, duration=0)
	slicer.util.sourceDir()
    Location of the Slicer source directory.

    Type str or None

    This provides the location of the Slicer source directory, if Slicer is being run from a CMake build directory. If
the Slicer home directory does not contain a CMakeCache.txt (e.g. for an installed Slicer), the property will
have the value None.
slicer.util.startQtDesigner(args=None)
Start Qt Designer application to allow editing UI files.

slicer.util.startupEnvironment()
Returns the environment without the Slicer specific values.
Path environment variables like PATH, LD_LIBRARY_PATH or PYTHONPATH will not contain values found in the launcher settings.
Similarly key=value environment variables also found in the launcher settings are excluded. Note that if a value was associated with a key prior starting Slicer, it will not be set in the environment returned by this function.
The function excludes both the Slicer launcher settings and the revision specific launcher settings.

slicer.util.tempDirectory(key='__SlicerTemp__', tempDir=None, includeDateTime=True)
Come up with a unique directory name in the temp dir and make it and return it # TODO: switch to QTemporaryDir in Qt5. Note: this directory is not automatically cleaned up

slicer.util.toBool(value)
Convert any type of value to a boolean.
The function uses the following heuristic:
1. If the value can be converted to an integer, the integer is then converted to a boolean.
2. If the value is a string, return True if it is equal to ‘true’. False otherwise. Note that the comparison is case insensitive.
3. If the value is neither an integer or a string, the bool() function is applied.

```python
>>> [toBool(x) for x in range(-2, 2)]
[True, True, False, True]
>>> [toBool(x) for x in ['-2', '-1', '0', '1', '2', 'Hello']]
[True, True, False, True, True, False]
>>> toBool(object())
True
>>> toBool(None)
False
```

slicer.util.toLatin1String(text)
Convert string to latin1 encoding.

slicer.util.toVTKString(text)
This method is deprecated. It converted unicode string into VTK string, but since now VTK assumes that all strings are in UTF-8 and all strings in Slicer are UTF-8, too, conversion is no longer necessary. The method is only kept for backward compatibility and will be removed in the future.

slicer.util.updateMarkupsControlPointsFromArray(markupsNode, narray, world=False)
Sets control point positions in a markups node from a numpy array of size Nx3.

Parameters world – if set to True then the control point coordinates are expected in world coordinate system.

All previous content of the node is deleted.

slicer.util.updateTableFromArray(tableName, narrays, columnNames=None)
Set values in a table node from a numpy array.
columnNames may contain a string or list of strings that will be used as column name(s). Values are copied, therefore if the numpy array is modified after calling this method, values in the table node will not change. All previous content of the table is deleted.

Example:
import numpy as np
histogram = np.histogram(arrayFromVolume(getNode('MRHead')))
tableNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLTableNode")
updateTableFromArray(tableNode, histogram, ["Count", "Intensity"])
slicer.util.updateTransformMatrixFromArray(transformNode, narray, toWorld=False)
    Set transformation matrix from a numpy array of size 4x4 (toParent).

    Parameters
    world – if set to True then the transform will be set so that transform to world matrix
    will be equal to narray; otherwise transform to parent will be set as narray.

slicer.util.updateVTKMatrixFromArray(vmatrix, narray)
    Update VTK matrix values from a numpy array.
    :param vmatrix: VTK matrix (vtkMatrix4x4 or vtkMatrix3x3) that will be update
    :param narray: input numpy array

To set numpy array from VTK matrix, use arrayFromVTKMatrix().

slicer.util.updateVolumeFromArray(volumeNode, narray)
    Sets voxels of a volume node from a numpy array. Voxels values are deep-copied, therefore
    if the numpy array is modified after calling this method, voxel values in the volume node
    will not change. Dimensions and data size of the source numpy array does not have to
    match the current content of the volume node.

slicer.util.vtkMatrixFromArray(narray)
    Create VTK matrix from a 3x3 or 4x4 numpy array.
    :param narray: input numpy array
    The returned matrix is just a copy and so any modification in the array will not affect
    the output matrix. To set numpy array from VTK matrix, use arrayFromVTKMatrix().

slicer.util.warningDisplay(text, windowTitle=None, parent=None, standardButtons=None, **kwargs)
    Display popup with a warning message.

Module contents

This module sets up root logging and loads the Slicer library modules into its namespace.

11.1.5 teem module
11.1.6 vtkAddon module
11.1.7 vtkITK module
CHAPTER 12

Contributing to Slicer

There are many ways to contribute to Slicer, with varying levels of effort. Do try to look through the documentation first if something is unclear, and let us know how we can do better.

- Ask a question on the Slicer forum
- Use Slicer Issues to submit a feature request or bug, or add to the discussion on an existing issue
- Submit a Pull Request to improve Slicer or its documentation

We encourage a range of Pull Requests, from patches that include passing tests and documentation, all the way down to half-baked ideas that launch discussions.

12.1 The PR Process, Circle CI, and Related Gotchas

12.1.1 How to submit a PR?

If you are new to Slicer development and you don’t have push access to the Slicer repository, here are the steps:

1. Fork and clone the repository.
2. Run the developer setup script SetupForDevelopment.sh.
3. Create a branch.
4. Push the branch to your GitHub fork.
5. Create a Pull Request.

This corresponds to the Fork & Pull Model mentioned in the GitHub flow guides.

When submitting a PR, the developers following the project will be notified. That said, to engage specific developers, you can add Cc: @<username> comment to notify them of your awesome contributions. Based on the comments posted by the reviewers, you may have to revisit your patches.
12.1.2 How to efficiently contribute?

We encourage all developers to:

- add or update tests. There are plenty of existing tests to inspire from. The testing how-tos are also resourceful.
- consider potential backward compatibility breakage and discuss these on the Slicer forum. For example, update of ITK, Python, Qt or VTK version, change to core functionality, should be carefully reviewed and integrated. Ideally, several developers would test that the changes don’t break extensions.

12.1.3 How to write commit messages?

Write your commit messages using the standard prefixes for Slicer commit messages:

- **BUG**: Fix for runtime crash or incorrect result
- **COMP**: Compiler error or warning fix
- **DOC**: Documentation change
- **ENH**: New functionality
- **PERF**: Performance improvement
- **STYLE**: No logic impact (indentation, comments)
- **WIP**: Work In Progress not ready for merge

The body of the message should clearly describe the motivation of the commit (**what**, **why**, and **how**). In order to ease the task of reviewing commits, the message body should follow the following guidelines:

1. Leave a blank line between the subject and the body. This helps `git log` and `git rebase` work nicely, and allows to smooth generation of release notes.
2. Try to keep the subject line below 72 characters, ideally 50.
3. Capitalize the subject line.
4. Do not end the subject line with a period.
5. Use the imperative mood in the subject line (e.g. **BUG**: Fix spacing not being considered.).
6. Wrap the body at 80 characters.
7. Use semantic line feeds to separate different ideas, which improves the readability.
8. Be concise, but honor the change: if significant alternative solutions were available, explain why they were discarded.
9. If the commit refers to a topic discussed on the Slicer forum, or fixes a regression test, provide the link. If it fixes a compiler error, provide a minimal verbatim message of the compiler error. If the commit closes an issue, use the GitHub issue closing keywords.

Keep in mind that the significant time is invested in reviewing commits and pull requests, so following these guidelines will greatly help the people doing reviews.

These guidelines are largely inspired by Chris Beam’s How to Write a Commit Message post.

Examples:

- **Bad**: **BUG**: Check pointer validity before dereferencing -> implementation detail, self-explanatory (by looking at the code)
- **Good**: **BUG**: Fix crash in Module X when clicking Apply button
• **Bad:** ENH: More work in qSlicerXModuleWidget -> more work is too vague, qSlicerXModuleWidget is too low level
• **Good:** ENH: Add float image outputs in module X

• **Bad:** COMP: Typo in cmake variable -> implementation detail, self-explanatory
• **Good:** COMP: Fix compilation error with Numpy on Visual Studio

### 12.1.4 How to integrate a PR?

Getting your contributions integrated is relatively straightforward, here is the checklist:

- All tests pass
- Consensus is reached. This usually means that at least two reviewers approved the changes (or added a LGTM comment) and at least one business day passed without anyone objecting. *LGMT* is an acronym for *Looks Good to Me*.
- To accommodate developers explicitly asking for more time to test the proposed changes, integration time can be delayed by few more days.
- If you do NOT have push access, a Slicer core developer will integrate your PR. If you would like to speed up the integration, do not hesitate to send a note on the Slicer forum.

#### 12.1.5 Automatic testing of pull requests

Every pull request is tested automatically using CircleCI each time you push a commit to it. The Github UI will restrict users from merging pull requests until the CI build has returned with a successful result indicating that all tests have passed.

The testing infrastructure is described in details in the [3D Slicer Improves Testing for Pull Requests Using Docker and CircleCI](https://3dslicer.org) blog post.

#### 12.1.6 Nightly tests

After changes are integrated, every evening at 10pm EST (3am UTC), Slicer build bots (aka factories) will build, test and package the Slicer application and all its extensions on Linux, MacOSX and Windows. Results are published daily on CDash (Stable & Preview) and developers that introduced changes resulting in build or test failures are notified by email.

#### 12.1.7 Decision-making process

1. Given the topic of interest, initiate discussion on the Slicer forum.
2. Identify a small circle of community members that are interested to study the topic in more depth.
3. Take the discussion off the general list, work on the analysis of options and alternatives, summarize findings on the wiki or similar. Labs page are usually a good ground for such summary.
4. Announce on the Slicer forum the in-depth discussion of the topic for the Slicer Community hangout, encourage anyone that is interested in weighing in on the topic to join the discussion. If there is someone who is interested to participate in the discussion, but cannot join the meeting due to conflict, they should notify the leaders of the given project and identify the time suitable for everyone.
5. Hopefully, reach consensus at the hangout and proceed with the agreed plan.
12.1.8 Benevolent dictators for life

The benevolent dictators can integrate changes to keep the platform healthy and help interpret or address conflict related to the contribution guidelines.

These currently include:

- Jean-Christophe Fillion-Robin
- Andras Lasso
- Steve Pieper

*Alphabetically ordered by last name.*

The Slicer community is inclusive and welcomes anyone to work to become a core developer and then a BDFL. This happens with hard work and approval of the existing BDFL.
Please see the GitHub project page at https://github.com/Slicer/Slicer/graphs/contributors
This module fixes common errors in DICOM files to make them possible to import them into Slicer. DICOM is a large and complex standard and device manufacturers and third-party software developers often make mistakes in their implementation. DICOM patcher module can recognize some common mistakes and certain known device-specific mistakes and create a modified copy of the DICOM files.

14.1 Panels and their use

- **Input DICOM directory**: folder containing the original, invalid DICOM files
- **Output DICOM directory**: folder that will contain the new, corrected DICOM files, typically this is a new, empty folder that is not a subfolder of the input DICOM directory
- **Normalize file names**: Replace file and folder names with automatically generated names. Fixes errors caused by file path containing special characters or being too long.
- **Force same patient name and ID in each directory**: Generate patient name and ID from the first file in a directory and force all other files in the same directory to have the same patient name and ID. Enable this option if a separate patient directory is created for each patched file.
- **Generate missing patient/study/series IDs**: Generate missing patient, study, series IDs. It is assumed that all files in a directory belong to the same series. Fixes error caused by too aggressive anonymization or incorrect DICOM image converters.
- **Generate slice position for multi-frame volumes**: Generate ‘image position sequence’ for multi-frame files that only have ‘SliceThickness’ field. Fixes error in Dolphin 3D CBCT scanners.
- **Partially anonymize**: If checked, then some patient identifiable information will be removed from the patched DICOM files. There are many fields that can identify a patient, this function does not remove all of them.
- **Patch**: create a fixed up copy of input files in the output folder
- **Import**: import fixed up files into Slicer DICOM database
- **Go to DICOM module**: switches to DICOM module, to see the imported DICOM files in the DICOM browser
14.2 Tutorial

- If you have already attempted to import files from the input folder then delete that from the Slicer DICOM database: go to DICOM module, right-click on the imported patient, and click Delete.
- Go to DICOM Patcher module (in Utilities category)
- Select input DICOM directory
- Select a new, empty folder as Output DICOM directory
- Click checkboxes of each fix operations that must be performed
- Click Patch button to create a fixed up copy of input files in the output folder
- Click Import button to import fixed up files into Slicer DICOM database
- Click Go to DICOM module to see the imported DICOM files in the DICOM browser

14.3 Related Modules

- module_dicom DICOM browser that lists all data sets in Slicer’s DICOM database.

14.4 Information for Developers

This is a Python scripted module. Source code is available at https://github.com/Slicer/Slicer/blob/master/Modules/Scripted/DICOMPatcher/DICOMPatcher.py.

14.5 Contributors

- Contributors: Andras Lasso (PerkLab, Queen’s University)

14.6 Acknowledgements

This module is partly funded by an Applied Cancer Research Unit of Cancer Care Ontario with funds provided by the Ministry of Health and Long-Term Care and the Ontario Consortium for Adaptive Interventions in Radiation Oncology (OCAIRO) to provide free, open-source toolset for radiotherapy and related image-guided interventions.
This is a module for segmentation of volumes. Segmentations (also known as contouring) delineate structures of interest. Some of the tools mimic a painting interface like Photoshop or GIMP, but work on 3D arrays of voxels rather than on 2D pixels. This module is a new, improved version of the old module_editor module. The Segment Editor contains many of the same functionalities and many more. New features include: overlapping segments, display in both 2D and 3D views, per-segment visualization options, editing in 3D views, create segmentation by interpolating or extrapolating segmentation on a few slices, editing on slices in any orientation.

It is important to remember that Segment Editor does not edit labelmap volumes, as Editor does. Segment editor creates segmentations, which can do many things that labelmap volumes cannot (overlapping contours, show/hide segments individually, show in 3D view, etc). Segmentations can be converted to labelmap volumes and models using the Import/Export section of Segment editor module.

15.1 Keyboard shortcuts

The following keyboard shortcuts are active when you are in the Editor module. They are intended to allow two-handed editing, where one hand is on the mouse and the other hand uses the keyboard to switch modes.
<table>
<thead>
<tr>
<th>Key</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>left arrow</td>
<td>move to previous slice</td>
</tr>
<tr>
<td>right arrow</td>
<td>move to next slice</td>
</tr>
<tr>
<td>Shift + mouse move</td>
<td>scroll slices to mouse location</td>
</tr>
<tr>
<td>Ctrl + mouse wheel</td>
<td>zoom image in/out</td>
</tr>
<tr>
<td>q</td>
<td>select previous segment</td>
</tr>
<tr>
<td>w</td>
<td>select next segment</td>
</tr>
<tr>
<td>z</td>
<td>undo</td>
</tr>
<tr>
<td>y</td>
<td>redo</td>
</tr>
<tr>
<td>esc</td>
<td>unselect effect</td>
</tr>
<tr>
<td>space</td>
<td>toggle between last two active effects</td>
</tr>
<tr>
<td>1, 2, ..., 0</td>
<td>select effect (1-10)</td>
</tr>
<tr>
<td>Shift + 1, 2, ..., 0</td>
<td>select effect (11-20)</td>
</tr>
<tr>
<td>1</td>
<td>toggle masking by intensity range</td>
</tr>
</tbody>
</table>

### 15.2 Panels and their use

- **Segmentation**: Choose the segmentation to edit.
- **Master volume**: Choose the volume to segment. The master volume that is selected the very first time after the segmentation is created is used to determine the segmentation’s labelmap representation geometry (resolution, axis directions, origin). The master volume is used by all editor effects that uses intensity of the segmented volume (e.g., thresholding, level tracing). The master volume can be changed at any time during the segmentation process. Note: Currently the only way to change geometry is to create a new segmentation, set its geometry, and then import segments from another segmentation.
- **Add segment**: Add a new segment to the segmentation and select it.
- **Remove segment**: Select the segment you would like to delete then click Remove segment to delete from the segmentation.
- **Create Surface**: Display your segmentation in the 3D Viewer. This is a toggle button. When turned on the surface is created and updated automatically as the user is segmenting. When turned off, the conversion is not ongoing so the segmentation process is faster. To change surface creation parameters: go to Segmentations module, click Update button in Closed surface row in Representations section, click Binary labelmap -> Closed surface line, double-click on value column to edit a conversion parameter value. Setting Smoothing factor to 0 disables smoothing, making updates much faster. Set Smoothing factor to 0.1 for weak smoothing and 0.5 or larger for stronger smoothing.
- **Segments table**: Displays list of all segments. - Eye icon: Toggle segment’s visibility. To customize visualization: either open the slice view controls (click on push-pint and double-arrow icons at the top of a slice viewer) or go to Segmentations module. - Color swatch: set color and assign segment to standardized terminology.
- **Effects**: Select the desired effect here. See below for more information about each effect.
- **Options**: Options for the selected effect will be displayed here.
- **Undo/Redo**: The module saves state of segmentation before each effect is applied. This is useful for experimentation and error correction. By default the last 10 states are remembered.
- **Masking**: These options allow you to define the editable areas and whether or not certain segments can be overwritten.
  - **Editable area**: Changes will be limited to the selected area. This can be used for drawing inside a specific region or split a segment into multiple segments.
– Editable intensity range: Changes will be limited to areas where the master volume’s voxels are in the selected intensity range. It is useful when locally an intensity threshold separates well between different regions. Intensity range can be previewed by using Threshold effect.

– Modify other segments: Select which segments will be overwritten rather than overlapped.
  * Overwrite all: Segment will not overlap (default).
  * Overwrite visible: Visible segments will not overlap with each other. Hidden segments will not be overwritten by changes done to visible segments.
  * Allow overlap: Changing one segment will not change any other.

15.3 Tutorials

• Tutorial for 3D printing: create a 3D-printable STL model from a section of the spinal column, segmented from a CT image, attached to a base designed in CAD software.

15.4 Effects

Effects operate either by clicking the Apply button in the effect options section or by clicking and/or dragging in slice or 3D views.

15.4.1 Paint

• Pick the radius (in millimeters) of the brush to apply
• Left click to apply single circle
• Left click and drag to fill a region
• A trace of circles is left which are applied when the mouse button is released
• Sphere mode applies the radius to slices above and below the current slice.

<table>
<thead>
<tr>
<th>Key</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift + mouse wheel</td>
<td>increase/decrease brush size</td>
</tr>
<tr>
<td>-</td>
<td>shrink brush radius by 20%</td>
</tr>
<tr>
<td>+</td>
<td>grow brush radius by 20%</td>
</tr>
</tbody>
</table>

15.4.2 Draw

• Left click to lay individual points of an outline
• Left drag to lay down a continuous line of points
• Right click to apply segment

<table>
<thead>
<tr>
<th>Key</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>×</td>
<td>delete the last point added</td>
</tr>
<tr>
<td>a</td>
<td>apply segment</td>
</tr>
</tbody>
</table>
15.4.3 Erase

Same as the Paint effect, but the highlighted regions are removed from the selected segment instead of added.

If Masking / Editable area is set to a specific segment then the highlighted region is removed from selected segment and added to the masking segment. This is useful when a part of a segment has to be separated into another segment.

<table>
<thead>
<tr>
<th>Key</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift + mouse wheel</td>
<td>increase/decrease brush size</td>
</tr>
<tr>
<td>-</td>
<td>shrink brush radius by 20%</td>
</tr>
<tr>
<td>+</td>
<td>grow brush radius by 20%</td>
</tr>
</tbody>
</table>

15.4.4 Level Tracing

- Moving the mouse defines an outline where the pixels all have the same background value as the current background pixel
- Clicking the left mouse button applies that outline to the label map

15.4.5 Grow from seeds

Draw segment inside each anatomical structure. This method will start from these “seeds” and grow them to achieve complete segmentation.

- Initialize: Click this button after initial segmentation is completed (by using other editor effects). Initial computation may take more time than subsequent updates. Master volume, auto-complete method, segmentation extent will be locked after initialization, therefore if any of these have to be changed then click Cancel and initialize again.
- Update: Update completed segmentation based on changed inputs.
- Auto-update: activate this option to automatically updating result preview when segmentation is changed.
- Cancel: Remove result preview. Seeds are kept unchanged, so parameters can be changed and segmentation can be restarted by clicking Initialize.
- Apply: Overwrite seeds segments with previewed results.

Notes:

- Only visible segments are used by this effect.
- At least two segments are required.
- If a part of a segment is erased or painting is removed using Undo (and not overwritten by another segment) then it is recommended to cancel and initialize. The reason is that effect of adding more information (painting more seeds) can be propagated to the complete segmentation, but removing information (removing some seed regions) will not change the complete segmentation.
- Extent of segmentation is limited to the bounding box defined by seed segments. The reason is that this makes computation faster and reduces memory usage for small segments. The extent can be made arbitrarily large by adding seeds at boundaries of the preferred extent.

### 15.4.6 Fill between slices

Create complete segmentation on selected slices using any editor effect. You can skip any number of slices between segmented slices. This method will fill the skipped slices by interpolating between segmented slices.

- **Initialize**: Click this button after initial segmentation is completed (by using other editor effects). Initial computation may take more time than subsequent updates. Master volume, auto-complete method, segmentation extent will be locked after initialization, therefore if any of these have to be changed then click Cancel and initialize again.
- **Update**: Update completed segmentation based on changed inputs.
- **Auto-update**: activate this option to automatically updating result preview when segmentation is changed.
- **Cancel**: Remove result preview. Seeds are kept unchanged, so parameters can be changed and segmentation can be restarted by clicking Initialize.
- **Apply**: Overwrite seeds segments with previewed results.

**Notes:**
- Only visible segments are used by this effect.
- The method does not use the master volume, only the shape of the specified segments.
- The method uses ND morphological contour interpolation algorithm. See details here: [http://insight-journal.org/browse/publication/977](http://insight-journal.org/browse/publication/977)

### 15.4.7 Threshold

Use Threshold to determine a threshold range and save results to selected segment or use it as Editable intensity range.

### 15.4.8 Margin

Grows or shrinks the selected segment by the specified margin.

### 15.4.9 Smoothing

Smoothes selected labelmap or all labelmaps (only for Joint smoothing method).

### 15.4.10 Scissors

Clip segments to the specified region or fill regions of a segment (typically used with masking). Regions can be drawn on both slice view or 3D views.
- **Left click** to start drawing (free-form or rubber band circle or rectangle)
• Release button to apply

15.4.11 Identify islands

Use this tool to create a unique segment for each connected region of the selected segment. Connected regions are defined as groups of pixels which touch each other but are surrounded by zero valued voxels.

• Fully connected: If checked then only voxels that share a face are counted as connected; if unchecked then voxels that touch at an edge or a corner are considered connected.

• Minimum size: All regions that have less than this number of voxels will be deleted.

15.4.12 Logical operators

Apply Boolean operators to selected segment or combine segments.

15.5 Hints

• A large radius paint brush with threshold painting is often a very fast way to segment anatomy that is consistently brighter or darker than the surrounding region, but partially connected to similar nearby structures (this happens a lot).

• Use the slice viewer menus to control the label map opacity and display mode (to show outlines only or full volume).

15.6 Limitations

• Threshold will not work with non-scalar volume background volumes.

• Mouse wheel can be used to move slice through volume, but on some platforms (mac) it may move more than one slice at a time.

15.7 Related Modules

• Segment statistics module computes volume, surface, mean intensity, and various other metrics for each segment.

• module_segmentations module allows changing visualization options, exporting/importing segments to/from other nodes (models, labelmap volumes), and moving or copying segments between segmentation nodes.

• module_data module shows all segmentations and segments in a tree structure. Commonly used operations are available by right-clicking on an item in the tree.

• module_editor is the predecessor of this module. Segment Editor will eventually replace the Editor module, therefore developers and users are encouraged to switch to Segment Editor module.
15.8 Information for Developers

See examples for creating and modifying segmentation nodes and using segment editor effects from your own modules in Slicer script repository.

15.9 Contributors

- Contributors: Csaba Pinter (PerkLab, Queen’s University), Andras Lasso (PerkLab, Queen’s University), Steve Pieper (Isomics Inc.), Wendy Plesniak (SPL, BWH), Ron Kikinis (SPL, BWH), Jim Miller (GE)
- Contact: Csaba Pinter, csaba.pinter@queensu.ca; Andras Lasso, lasso@queensu.ca

15.10 Acknowledgements

This module is partly funded by an Applied Cancer Research Unit of Cancer Care Ontario with funds provided by the Ministry of Health and Long-Term Care and the Ontario Consortium for Adaptive Interventions in Radiation Oncology (OCAIRO) to provide free, open-source toolset for radiotherapy and related image-guided interventions. The work is part of the National Alliance for Medical Image Computing (NA-MIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.
This is a module for the calculation of statistics related to the structure of segmentations, such as volume, surface area, mean intensity, and various other metrics for each segment.

### 16.1 Labelmap statistics

Labelmap statistics are calculated using the binary labelmap representation of the segment.

- **Voxel count**: the number of voxels in the segment
- **Volume mm$^3$**: the volume of the segment in mm$^3$
- **Volume cm$^3$**: the volume of the segment in cm$^3$
- **Centroid**: the center of mass of the segment in RAS coordinates
- **Feret diameter**: the diameter of a sphere that can encompass the entire segment
- **Surface area mm$^2$**: the volume of the segment in mm$^2$
- **Roundness**: the roundness of the segment. Calculated from ratio of the area of the sphere calculated from the feret diameter by the actual area. Value of 1 represents a spherical structure. ([http://hdl.handle.net/1926/584](http://hdl.handle.net/1926/584))
- **Flatness**: the flatness of the segment. Calculated from square root of the ratio of the second smallest principal moment by the smallest. Value of 0 represents a flat structure. ([http://hdl.handle.net/1926/584](http://hdl.handle.net/1926/584))
- **Elongation**: the elongation of the segment. Calculated from square root of the ratio of the second largest principal moment by the second smallest. ([http://hdl.handle.net/1926/584](http://hdl.handle.net/1926/584))
- **Principal moments**: the principal moments of inertia for each axes of the segment
- **Principal axes**: the principal axes of rotation of the segment
- **Oriented bounding box**: the non-axis aligned bounding box that encompasses the segment
16.2 Scalar volume statistics

- Voxel count: the number of voxels in the segment
- Volume mm$^3$: the volume of the segment in mm$^3$
- Volume cm$^3$: the volume of the segment in cm$^3$
- Minimum: the minimum scalar value behind the segment
- Maximum: the maximum scalar value behind the segment
- Mean: the mean scalar value behind the segment
- Median: the median scalar value behind the segment
- Standard deviation: the standard deviation of scalar values behind the segment

16.3 Closed surface statistics

- Surface area mm$^2$: the volume of the segment in mm$^2$
- Volume mm$^3$: the volume of the segment in mm$^3$
- Volume cm$^3$: the volume of the segment in cm$^3$

16.4 Related Modules

- module_segmentations module allows changing visualization options, exporting/importing segments to/from other nodes (models, labelmap volumes), and moving or copying segments between segmentation nodes.
- Segment editor module for segmentation of volumes using tools for editing (paint, draw, erase, level tracing, grow from seeds, threshold, etc.)

16.5 Information for Developers

See examples for calculating statistics from your own modules in the Slicer script repository. Additional plugins for computation of other statistical measurements may be registered by subclassing SegmentStatisticsPluginBase.py, and registering the plugin with SegmentStatisticsLogic.

16.6 Contributors

- Contributors: Csaba Pinter (PerkLab, Queen’s University), Andras Lasso (PerkLab, Queen’s University), Christian Bauer (University of Iowa), Steve Pieper (Isomics Inc.)
- Contact: Csaba Pinter, csaba.pinter@queensu.ca; Andras Lasso, lasso@queensu.ca
16.7 Acknowledgements

This module is partly funded by an Applied Cancer Research Unit of Cancer Care Ontario with funds provided by the Ministry of Health and Long-Term Care and the Ontario Consortium for Adaptive Interventions in Radiation Oncology (OCAIRO) to provide free, open-source toolset for radiotherapy and related image-guided interventions. The work is part of the National Alliance for Medical Image Computing (NA-MIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.
CHAPTER 17

Indices and tables

- genindex
- modindex
- search
Python Module Index

S
slicer.52
slicer.cli.35
slicer.testing.36
slicer.util.36
Index

A
addObserver() (slicer.util.VTKObservationMixin method), 36
addVolumeFromArray() (in module slicer.util), 36
array() (in module slicer.util), 37
arrayFromGridTransform() (in module slicer.util), 37
arrayFromGridTransformModified() (in module slicer.util), 37
arrayFromMarkupsControlPoints() (in module slicer.util), 37
arrayFromMarkupsCurvePoints() (in module slicer.util), 37
arrayFromModelPointData() (in module slicer.util), 38
arrayFromModelPoints() (in module slicer.util), 38
arrayFromModelPointsModified() (in module slicer.util), 38
arrayFromModelPolyIds() (in module slicer.util), 38
arrayFromSegment() (in module slicer.util), 38
arrayFromSegmentBinaryLabelmap() (in module slicer.util), 38
arrayFromSegmentInternalBinaryLabelmap() (in module slicer.util), 38
arrayFromTableColumn() (in module slicer.util), 39
arrayFromTableColumnModified() (in module slicer.util), 39
arrayFromTransformMatrix() (in module slicer.util), 39
arrayFromVolume() (in module slicer.util), 39
arrayFromVolumeModified() (in module slicer.util), 40
arrayFromVTKMatrix() (in module slicer.util), 39
childWidgetVariables() (in module slicer.util), 40
clickAndDrag() (in module slicer.util), 40
computeChecksum() (in module slicer.util), 40
confirmOkCancelDisplay() (in module slicer.util), 40
confirmRetryCloseDisplay() (in module slicer.util), 41
confirmYesNoDisplay() (in module slicer.util), 41
createNode() (in module slicer.cli), 35
createProgressDialog() (in module slicer.util), 41
delayDisplay() (in module slicer.util), 41
downloadAndExtractArchive() (in module slicer.util), 41
downloadFile() (in module slicer.util), 41
derrorDisplay() (in module slicer.util), 41
exit() (in module slicer.util), 41
exitFailure() (in module slicer.testing), 36
exitSuccess() (in module slicer.testing), 36
extractAlgoAndDigest() (in module slicer.util), 42
extractArchive() (in module slicer.util), 42
findChild() (in module slicer.util), 42
findChildren() (in module slicer.util), 42
forceRenderAllViews() (in module slicer.util), 42
getFilesInDirectory() (in module slicer.util), 42
getFirstNodeByClassByName() (in module slicer.util), 42
getFirstNodeByName() (in module slicer.util), 42
getModule() (in module slicer.util), 42

C
cancel() (in module slicer.cli), 35
getModuleGui() (in module slicer.util), 42
getNewModuleGui() (in module slicer.util), 42
getNode() (in module slicer.util), 42
getNodes() (in module slicer.util), 42
getNodesByClass() (in module slicer.util), 43
hasObserver() (slicer.util.VTKObservationMixin method), 36
importClassesFromDirectory() (in module slicer.util), 43
importModuleObjects() (in module slicer.util), 43
importQtClassesFromDirectory() (in module slicer.util), 43
importVTKClassesFromDirectory() (in module slicer.util), 43
infoDisplay() (in module slicer.util), 43
launchConsoleProcess() (in module slicer.util), 43
loadAnnotationFiducial() (in module slicer.util), 43
loadAnnotationROI() (in module slicer.util), 43
loadAnnotationRuler() (in module slicer.util), 43
loadColorTable() (in module slicer.util), 44
loadFiberBundle() (in module slicer.util), 44
loadFiducialList() (in module slicer.util), 44
loadLabelVolume() (in module slicer.util), 44
loadMarkupsClosedCurve() (in module slicer.util), 44
loadMarkupsCurve() (in module slicer.util), 44
loadMarkupsFiducialList() (in module slicer.util), 44
loadModel() (in module slicer.util), 45
loadNodeFromFile() (in module slicer.util), 45
loadNodesFromFile() (in module slicer.util), 45
loadScalarOverlay() (in module slicer.util), 45
loadScene() (in module slicer.util), 45
loadSegmentation() (in module slicer.util), 46
loadShaderProperty() (in module slicer.util), 46
loadText() (in module slicer.util), 46
loadTransform() (in module slicer.util), 46
loadUI() (in module slicer.util), 46
loadVolume() (in module slicer.util), 46
logProcessOutput() (in module slicer.util), 47
lookupTopLevelWidget() (in module slicer.util), 47
mainWindow() (in module slicer.util), 47
messageBox() (in module slicer.util), 47
moduleNames() (in module slicer.util), 47
modulePath() (in module slicer.util), 47
moduleSelector() (in module slicer.util), 47
MRMLNodeNotFoundException, 36
NodeModify (class in slicer.util), 36
hasObserver() (slicer.util.VTKObservationMixin method), 36
openAddColorTableDialog() (in module slicer.util), 47
openAddDataDialog() (in module slicer.util), 47
openAddFiberBundleDialog() (in module slicer.util), 47
openAddFiducialDialog() (in module slicer.util), 47
openAddMarkupsDialog() (in module slicer.util), 47
openAddModelDialog() (in module slicer.util), 47
openAddScalarOverlayDialog() (in module slicer.util), 47
openAddSegmentationDialog() (in module slicer.util), 47
openAddShaderPropertyDialog() (in module slicer.util), 47
openAddTransformDialog() (in module slicer.util), 47
openAddVolumeDialog() (in module slicer.util), 47
openSaveDataDialog() (in module slicer.util), 47
pip_install() (in module slicer.util), 47
plot() (in module slicer.util), 48
pythonShell() (in module slicer.util), 49
quit() (in module slicer.util), 49
reloadScriptedModule() (in module slicer.util), 49
removeObserver() (slicer.util.VTKObservationMixin method), 36
removeObservers() (slicer.util.VTKObservationMixin method), 36
resetSliceViews() (in module slicer.util), 49
resetThreeDViews() (in module slicer.util), 49
restart() (in module slicer.util), 49
run() (in module slicer.cli), 35
runSync() (in module slicer.cli), 35
runUnitTest () (in module slicer.testing), 36

S
saveNode () (in module slicer.util), 49
saveScene () (in module slicer.util), 49
selectedModule () (in module slicer.util), 49
selectModule () (in module slicer.util), 49
setApplicationLogoVisible () (in module slicer.util), 50
setDataProbeVisible () (in module slicer.util), 50
setMenuBarsVisible () (in module slicer.util), 50
setModuleHelpSectionVisible () (in module slicer.util), 50
setModulePanelTitleVisible () (in module slicer.util), 50
setNodeParameters () (in module slicer.cli), 36
setPythonConsoleVisible () (in module slicer.util), 50
setSliceViewerLayers () (in module slicer.util), 50
setStatusBarVisible () (in module slicer.util), 50
settingsValue () (in module slicer.util), 50
setToolbarsVisible () (in module slicer.util), 50
setViewControllersVisible () (in module slicer.util), 50
showStatusMessage () (in module slicer.util), 50
slicer (module), 52
slicer.cli (module), 35
slicer.testing (module), 36
slicer.util (module), 36
sourceDir () (in module slicer.util), 50
startQtDesigner () (in module slicer.util), 50
startupEnvironment () (in module slicer.util), 51

T
tempDirectory () (in module slicer.util), 51
toBool () (in module slicer.util), 51
toLatin1String () (in module slicer.util), 51
toVTKString () (in module slicer.util), 51

U
updateMarkupsControlPointsFromArray () (in module slicer.util), 51
updateTableFromArray () (in module slicer.util), 51
updateTransformMatrixFromArray () (in module slicer.util), 52
updateVolumeFromArray () (in module slicer.util), 52
updateVTKMatrixFromArray () (in module slicer.util), 52

V
vtkMatrixFromArray () (in module slicer.util), 52