# 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 9

5 Commercial Use 11
  5.1 Slicer’s License makes Commercial Use Available 11
  5.2 Commercial Partners 11
  5.3 Slicer Based Products 12

6 User Interface 13
  6.1 Application overview 13
  6.2 Interacting with views 15
  6.3 Mouse & Keyboard Shortcuts 18

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
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](#) for 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.
CHAPTER 3

Citing

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.
CHAPTER 4

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 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, Saggital, 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></td>
</tr>
<tr>
<td>Ctrl + s</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></td>
</tr>
<tr>
<td>left arrow/right arrow</td>
<td>move to previous/next slice</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.
<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>rotate view</td>
</tr>
<tr>
<td>up arrow/down arrow</td>
<td>rotate view</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>pan (translate) view</td>
</tr>
<tr>
<td>+/-</td>
<td></td>
</tr>
<tr>
<td>middle-click + drag</td>
<td></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
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 be only 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
- module_segmenteditor
- module_transforms
- module_viewcontrollers
- module_volumerendering
- module_volumes
- module_welcometoslicer

8.2 Wizards

- module_comparevolumes
8.3 Informatics

- module_annotations
- 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
- module_segmenteditor
- module_simpleregiongrowingsegmentation
- Specialized:
  - module_emsegmentcommandline
  - module_brainsroiauto
  - module_robuststatisticssegmenter
8.6 Quantification

- module_dataprobe
- module_labelstatistics
- module_brainslabelstats
- module_petstandarduptakevaluecomputation
- module_segmentstatistics

8.7 Diffusion

- module_DMRIInstall
- Import and export:
  - module_DWIConvert
- Utilities:
  - module_BRAINSDDWICleanup
  - 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
• module_dicompatcher
8.14 Developer Tools

- module_ScreenCapture
- module_EMSegmentTransformToNewFormat
- BRAINS:
  - module_BRAINSTransformConvert

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

“Safe weakrefs”, originally from pyDispatcher.

Provides a way to safely weakref any function, including bound methods (which aren’t handled by the core weakref module).

```python
class saferef.BoundMethodWeakref(target, onDelete=None):
    Bases: object

    ‘Safe’ and reusable weak references to instance methods

    BoundMethodWeakref objects provide a mechanism for referencing a bound method without requiring that the
    method object itself (which is normally a transient object) is kept alive. Instead, the BoundMethodWeakref
    object keeps weak references to both the object and the function which together define the instance method.

Attributes:

    key – the identity key for the reference, calculated by the class’s calculateKey method applied to the
target instance method

    deletionMethods – sequence of callable objects taking single argument, a reference to this object
    which will be called when either the target object or target function is garbage collected (i.e. when
    this object becomes invalid). These are specified as the onDelete parameters of safeRef calls.

    weakSelf – weak reference to the target object weakFunc – weak reference to the target function

Class Attributes:
_allInstances – class attribute pointing to all live BoundMethodWeakref objects indexed by the class’s calculateKey(target) method applied to the target objects. This weak value dictionary is used to short-circuit creation so that multiple references to the same (object, function) pair produce the same BoundMethodWeakref instance.

classmethod calculateKey(target)
    Calculate the reference key for this reference
    Currently this is a two-tuple of the id()’s of the target object and the target function respectively.

class saferef.BoundNonDescriptorMethodWeakref(target, onDelete=None)
    Bases: saferef.BoundMethodWeakref
    A specialized BoundMethodWeakref, for platforms where instance methods are not descriptors.
    It assumes that the function name and the target attribute name are the same, instead of assuming that the function is a descriptor. This approach is equally fast, but not 100% reliable because functions can be stored on an attribute named differently than the function’s name such as in:
    class A: pass
def foo(self): return “foo”
A.bar = foo
    But this shouldn’t be a common use case. So, on platforms where methods aren’t descriptors (such as Jython) this implementation has the advantage of working in the most cases.

saferef.get_bound_methodWeakref(target, onDelete)
    Instantiates the appropriate BoundMethodWeakRef, depending on the details of the underlying class method implementation

saferef.safeRef(target, onDelete=None)
    Return a safe weak reference to a callable target

    target – the object to be weakly referenced, if it’s a bound method reference, will create a BoundMethodWeakref, otherwise creates a simple weakref.

    onDelete – if provided, will have a hard reference stored to the callable to be called after the safe reference goes out of scope with the reference object, (either a weakref or a BoundMethodWeakref) as argument.

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: exceptions.Exception

Exception raised when a requested MRML node was not found.

class slicer.util.NodeModify(node)

Context manager to conveniently compress mrml node modified event.

class slicer.util.VTKObservationMixin

Bases: object

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

hasObserver(object, event, method)

observer(event, method)

removeObserver(object, event, method)

removeObservers(method=None)

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

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

11.1. Python
node can be modified by changing values in the numpy array. After all modifications has been completed, call gridTransformNode.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.

```python
slicer.util.arrayFromModelPoints(modelNode)
```

Return point positions of a model node as numpy array. Voxels values in the volume node can be modified by modifying the numpy array. After all modifications has been completed, call modelNode.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.

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

Return voxel array of a segment’s binary labelmap representation as numpy array. Voxels values are not 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 has 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.

```python
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 volumeNode.Modified().

**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 `numpy.copy()`.
2. Perform any computations using the copied array.
3. Write results back to the image data using `updateVolumeFromArray()`.

```python
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)

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

**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)
```
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 http://pyqt.sourceforge.net/Docs/PyQt4/qprogressbar.html for more keyword arguments, that could be used. E.g. progressbar = createProgressIndicator(autoClose=False) if you don’t want the progress dialog to automatically close. Updating progress value with progressbar.setValue = 50 Updating label text with 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 closed after waiting for autoCloseMsec milliseconds If autoCloseMsec=0 then the window is not closed until the user clicks on it.

slicer.util.downloadAndExtractArchive (url, archiveFilePath, outputDir, expectedNumberOfExtractedFiles=None, numberOfTrials=3)
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.

slicer.util.downloadFile (url, targetFilePath)
Download url to local storage as targetFilePath
Target file path needs to indicate the file name and extension as well

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

slicer.util.exit (status=0)

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.

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.

11.1. Python 39
slicer.util.getFirstNodeByName(name, className=None)
Get the first MRML node that name starts with the specified name. Optionally specify a classname that must also match.

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. Throws MRMLNodeNotFoundException exception 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 node 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 a info message.

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

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

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

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

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

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

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

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

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

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

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

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

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

slicer.util.loadTransform(filename, returnNode=False)
slicer.util.loadUI(path)
    Load UI file path and return the corresponding widget. Raises a RuntimeError exception if the UI file is not found or if no widget was instantiated.

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

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

slicer.util.mainWindow(verbos=True)

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: slicer.util.messageBox(“Some message”, dontShowAgainSettingsKey = “MainWindow/DontShowSomeMessage”)

slicer.util.moduleNames()

slicer.util.modulePath(moduleName)

slicer.util.moduleSelector()

slicer.util.oppenAddColorTableDialog()

slicer.util.oppenAddDataDialog()

slicer.util.oppenAddFiberBundleDialog()

slicer.util.oppenAddFiducialDialog()

slicer.util.oppenAddModelDialog()

slicer.util.oppenAddScalarOverlayDialog()

slicer.util.oppenAddSegmentationDialog()

slicer.util.oppenAddTransformDialog()

slicer.util.oppenAddVolumeDialog()

slicer.util.oppenSaveDataDialog()

slicer.util.pythonShell(verbos=True)

slicer.util.quit()

slicer.util.reloadScriptedModule(moduleName)
    Generic reload method for any scripted module.

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.

```python
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.

```python
slicer.util.selectModule(module)
```

```python
slicer.util.selectedModule()
```

```python
slicer.util.setSliceViewerLayers(background='keep-current', foreground='keep-current', label='keep-current', foregroundOpacity=None, labelOpacity=None)
```
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.

- `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

```python
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.

```python
slicer.util.showStatusMessage(message, duration=0)
```

```python
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.

```python
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.

```python
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

```python
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.
slicer.util.toVTKString(str)
    Convert unicode string into 8-bit encoded ascii string. Unicode characters without ascii equivalent will be stripped out.

slicer.util.unicodeify(s)

slicer.util.updateTableFromArray(tableNode, narrays)
    Sets values in a table node from a numpy array. Values are copied, therefore if the numpy array is modified after calling this method, values in the table node will not change.

Example:
    import numpy as np
    histogram = np.histogram(arrayFromVolume(getNode('MRHead')))
    tableNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLTableNode")
    updateTableFromArray(tableNode, histogram)

slicer.util.updateVolumeFromArray(volumeNode, narray)
    Sets voxels of a volume node from a numpy array. Voxel 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.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
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-devel email list
- Submit a feature request or bug, or add to the discussion on the Slicer issue tracker
- 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. Create a branch.
3. Push the branch to your GitHub fork.
4. 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:

- review and follow the style guidelines described on the wiki.
- 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 mailing list. 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 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. LGTM 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.

Next, there are two scenarios:

• 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 mailing list.

• You have push access: Follow Integrating topic from external contributor instructions on the wiki.

12.1.4 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 blog post.

12.1.5 Nightly tests

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

12.1.6 Decision-making process

1. Given the topic of interest, initiate discussion on the mailing list.

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 mailing list 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.

The initial version of these guidelines was established during the [winter project week 2017](http://www.namic.org/Wiki/index.php/2017_Winter_Project_Week/UpdatingCommunityForums).
12.1.7 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 welcome 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
CHAPTER 14

Indices and tables

- genindex
- modindex
- search
S
saferef, 35
slicer, 43
slicer.cli, 36
slicer.testing, 37
slicer.util, 37
loadAnnotationROI() (in module slicer.util), 40
loadAnnotationRuler() (in module slicer.util), 40
loadColorTable() (in module slicer.util), 40
loadFiberBundle() (in module slicer.util), 40
loadFiducialList() (in module slicer.util), 40
loadLabelVolume() (in module slicer.util), 40
loadMarkupsFiducialList() (in module slicer.util), 40
loadModel() (in module slicer.util), 40
loadNodeFromFile() (in module slicer.util), 40
loadScalarOverlay() (in module slicer.util), 40
loadScene() (in module slicer.util), 40
loadSegmentation() (in module slicer.util), 40
loadTransform() (in module slicer.util), 40
loadUI() (in module slicer.util), 40
loadVolume() (in module slicer.util), 40
lookupTopLevelWidget() (in module slicer.util), 41
MainWindow() (in module slicer.util), 41
messageBox() (in module slicer.util), 41
moduleNames() (in module slicer.util), 41
modulePath() (in module slicer.util), 41
moduleSelector() (in module slicer.util), 41
MRMLNodeNotFoundException, 37
NodeModify (class in slicer.util), 37
observer() (slicer.util.VTKObservationMixin method), 37
openAddColorTableDialog() (in module slicer.util), 41
openAddDataDialog() (in module slicer.util), 41
openAddFiberBundleDialog() (in module slicer.util), 41
openAddFiducialDialog() (in module slicer.util), 41
openAddModelDialog() (in module slicer.util), 41
openAddScalarOverlayDialog() (in module slicer.util), 41
openAddSegmentationDialog() (in module slicer.util), 41
openAddTransformDialog() (in module slicer.util), 41
openAddVolumeDialog() (in module slicer.util), 41
openSaveDataDialog() (in module slicer.util), 41
pythonShell() (in module slicer.util), 41
quit() (in module slicer.util), 41
reloadScriptedModule() (in module slicer.util), 41
removeObserver() (slicer.util.VTKObservationMixin method), 37
removeObservers() (slicer.util.VTKObservationMixin method), 37
resetSliceViews() (in module slicer.util), 41
resetThreeDViews() (in module slicer.util), 41
restart() (in module slicer.util), 41
run() (in module slicer.cli), 36
runSync() (in module slicer.cli), 36
runUnitTest() (in module slicer.testing), 37
saferef (module), 35
safeRef() (in module saferef), 36
saveNode() (in module slicer.util), 41
saveScene() (in module slicer.util), 42
selectedModule() (in module slicer.util), 42
selectModule() (in module slicer.util), 42
setNodeParameters() (in module slicer.cli), 37
setSliceViewerLayers() (in module slicer.util), 42
settingsValue() (in module slicer.util), 42
showStatusMessage() (in module slicer.util), 42
slicer (module), 43
slicer.cli (module), 36
slicer.testing (module), 37
slicer.util (module), 37
sourceDir() (in module slicer.util), 42
startupEnvironment() (in module slicer.util), 42
tempDirectory() (in module slicer.util), 42
toBool() (in module slicer.util), 42
toVTKString() (in module slicer.util), 43
unicodeify() (in module slicer.util), 43
updateTableFromArray() (in module slicer.util), 43
updateVolumeFromArray() (in module slicer.util), 43
VTKObservationMixin (class in slicer.util), 37
warningDisplay() (in module slicer.util), 43