3D Slicer Documentation

Slicer Community

Aug 29, 2020
## Contents

1 About 3D Slicer
   1.1 What is 3D Slicer? .......................................................... 3
   1.2 License ................................................................. 4
   1.3 How to cite ............................................................ 4
   1.4 Acknowledgments ......................................................... 6
   1.5 Commercial Use .......................................................... 7
   1.6 Contact us ............................................................. 8

2 Getting Started .......................................................... 9
   2.1 System requirements ...................................................... 9
   2.2 Installing 3D Slicer ....................................................... 10
   2.3 Using Slicer ........................................................... 11

3 User Interface .......................................................... 17
   3.1 Application overview .................................................... 17
   3.2 Interacting with views .................................................. 19
   3.3 Mouse & Keyboard Shortcuts ........................................ 23

4 Data Loading and Saving ................................................ 25

5 Image Segmentation ..................................................... 29
   5.1 Basic concepts .......................................................... 30
   5.2 3D Slicer modules for segmentation ................................ 30

6 Modules ................................................................. 33
   6.1 Overview ............................................................... 33
   6.2 Main ................................................................. 33
   6.3 Wizards ................................................................. 67
   6.4 Informatics ............................................................. 67
   6.5 Registration ........................................................... 67
   6.6 Segmentation ........................................................... 67
   6.7 Quantification ........................................................... 69
   6.8 Diffusion ............................................................... 69
   6.9 Filtering ................................................................. 69
   6.10 Utilities ................................................................. 69
   6.11 Surface models ........................................................ 71
   6.12 Converters .............................................................. 71
This is documentation is a work in progress, in preparation for the new Slicer-5.0 release.

For Slicer-4.10 documentation, refer to the 3D Slicer wiki.
1.1 What is 3D Slicer?

• A software application for visualization and analysis of medical image computing data sets. All commonly used data sets are supported, such as images, segmentations, surfaces, annotations, transformations, etc., in 2D, 3D, and 4D. Visualization is available on desktop and in virtual reality. Analysis includes segmentation, registration, and various quantifications.

• A research software platform, which allows researchers to quickly develop and evaluate new methods and distribute them to clinical users. All features are available and extensible in Python and C++. A full Python environment is provided where any Python packages can be installed and combined with built-in features. Slicer has a built-in Python console and can act as a Jupyter notebook kernel with remote 3D rendering capabilities.

• Product development platform, which allows companies to quickly prototype and release products to users. Developers can focus on developing new methods and do not need to spend time with redeveloping basic data import/export, visualization, interaction features. The application is designed to be highly customizable (with custom branding, simplified user interface, etc.). 3D Slicer is completely free and there are no restrictions on how it is used - it is up to the software distributor to ensure that the developed application is suitable for the intended use.

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

Highlights:

• Free, open-source software available on multiple operating systems: Linux, MacOSX and Windows.

• Multi organ: from head to toe.

• Support for multi-modality imaging including, MRI, CT, US, nuclear medicine, and microscopy.

• Real-time interface for medical devices, such as surgical navigation systems, imaging systems, robotic devices, and sensors.
• Highly extensible: users can easily add more capabilities by installing additional modules from the Extension manager, running custom Python scripts in the built-in Python console, run any executables from the application’s user interface, or implement custom modules in Python or C++.

• Large and active user community.

1.2 License

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.

1.3 How to cite

1.3.1 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:


1.3.2 Individual modules

To acknowledge individual modules: each module has an acknowledgment tab in the top section. Information about contributors and funding source can be found there:
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:

http://www.slicer.org

1.3. How to cite

The Volumes Module loads and adjusts display parameters of volume data.
1.4 Acknowledgments

Slicer is made possible through contributions from an international community of scientists from a multitude of fields, including engineering and biomedicine. The following sections give credit to some of the major contributors to the 3D Slicer core effort. Each 3D Slicer extension has a separate acknowledgements page with information specific to that extension.

Ongoing Slicer support depends on YOU

Please give the Slicer repository a star on github. This is an easy way to show thanks and it can help us qualify for useful services that are only open to widely recognized open projects. Don’t forget to cite our publications because that helps us get new grant funding. If you find Slicer is helpful like the community please get involved. You don’t need to be a programmer to help!

1.4.1 Major Contributors

• Ron Kikinis: Principal Investigator
• Steve Pieper: Chief Architect
• Jean-Christophe Fillion-Robin: Lead Developer
• Nicole Aucoin
• Stephen Aylward
• Andrey Fedorov
• Noby Hata
• Hans Johnson
• Tina Kapur
• Gabor Fichtinger
• Andras Lasso
• Csaba Pinter
• Jim Miller
• Sonia Pujol: Director of Training
• Junichi Tokuda
• Lauren O’Donnell
• Andinet Enquobahrie
• Beatriz Paniagua

Contributors are not only developers, but also individual helping to secure funding and move the platform forward.

1.4.2 Groups Contributing to the Core Engineering of Slicer in a Major Way

• SPL: Ron Kikinis, Nicole Aucoin, Lauren O’Donnell, Andrey Fedorov, Isaiah Norton, Sonia Pujol, Noby Hata, Junichi Tokuda
• Isomics: Steve Pieper, Alex Yarmarkovich
• Kitware: Jean-Christophe Fillion-Robin, Julien Finet, Will Schroeder, Stephen Aylward, Andinet Enquobahrie, Beatriz Paniagua, Matt McCormick, Johan Andruejol, Max Smolens, Alexis Girault, Sam Horvath
• University of Iowa: Hans Johnson
• GE: Jim Miller
• Perk Lab, Queen’s University: Andras Lasso, Tamas Ungi, Csaba Pinter, Gabor Fichtinger
• Kapteyn Astronomical Institute, University of Groningen: Davide Punzo

1.4.3 Funding Sources

Many of the activities around the Slicer effort are made possible through funding from public and private sources. The National Institutes of Health of the USA is a major contributor through a variety of competitive grants and contracts. See a selection of funding grants here.

1.5 Commercial Use

We invite commercial entities to use 3D Slicer.

1.5.1 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.

1.5.2 Commercial Partners

• Ebatinca SL is an international technology company in Las Palmas, Spain focused on technology for sustainable development.
• 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.
1.5.3 3D Slicer based products

Many companies prefer not to disclose what software components they use in their products, therefore here we can only list a few commercial products that are based on 3D Slicer:

- Allen Institute for Brain Science: Cell Locator, Desktop application for manually aligning specimens to annotated 3D spaces.

- Radiopharmaceutical Imaging and Dosimetry: RPTDose, a 3D Slicer-based application that streamlines and integrates quantitative imaging analysis and dose estimation techniques to guide and optimize the use of radiopharmaceutical therapy agents in clinical trials. See more information on this Kitware blog.

- 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. See more information on this Kitware blog.


- 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. See more information on this Kitware blog.

Listed in alphabetical order.

1.6 Contact us

It is recommended to post any questions, bug reports, or enhancement requests to the Slicer forum.

Our online issue tracker is available here.

For commercial/confidential consulting, contact one of the commercial partners.
Welcome to the 3D Slicer community. This page contains information that you need to get started with 3D Slicer, including how to install and use basic features and where to find more information.

2.1 System requirements

3D Slicer runs on any Windows, Mac, or Linux computer that was released in the last 5 years. Older computers may work (depending mainly on graphics capabilities). Slicer can also run on virtual machines and docker containers.

2.1.1 Operating system versions

Notes:

- It is always recommended to use the latest public release of Windows or macOS. For Linux, this would be the latest LTS (Long-term-support) version.
- Slicer is expected to work on the vast majority of desktop and server Linux distributions. The system is required to provide at least GLIBC 2.17 and GLIBCCC 3.4.19. For more details, read here.
- Only 64-bit Slicer installers are available to download. Developers can attempt to build 32-bit versions on their own if they need to run Slicer on a 32-bit operating system. That said, this should be carefully considered as many clinical research tasks, such as processing of large CT or MR volumetric datasets, require more memory than can be accommodated with a 32-bit program.

2.1.2 Recommended hardware configuration:

- Memory: more than 4GB (8 or more is recommended). As a general rule, have 10x more memory than the amount of data that you load.
- Display: a minimum resolution of 1024 by 768 (1280 by 1024 or better is recommended).
• Graphics: Dedicated graphics hardware (discrete GPU) memory is recommended for fast volume rendering. GPU: Graphics must support minimum OpenGL 3.2. Integrated graphics card is sufficient for basic visualization. Discrete graphics card (such as NVidia GPU) is recommended for interactive 3D volume rendering and fast rendering of complex scenes. GPU texture memory (VRAM) should be larger than your largest dataset (e.g., working with 2GB data, get VRAM > 4GB) and check that your images fit in maximum texture dimensions of your GPU hardware. Except rendering, most calculations are performed on CPU, therefore having a faster GPU will generally not impact the overall speed of the application.

• Some computations in 3D Slicer are multi-threaded and will benefit from multi core, multi CPU configurations.

• Interface device: a three button mouse with scroll wheel is recommended. Pen, multi-touchscreen, touchpad, and graphic tablet are supported. All OpenVR-compatible virtual reality headsets are supported for virtual reality display.

• Internet connection to access extensions, Python packages, online documentation, sample data sets, and tutorials.

2.2 Installing 3D Slicer

To download Slicer, click here.

Installers

<table>
<thead>
<tr>
<th></th>
<th>Windows</th>
<th>macOS</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preview Release</td>
<td>version 4.11.0 revision 28879 built 2020-03-25</td>
<td>version 4.11.0 revision 28879 built 2020-03-25</td>
<td>version 4.11.0 revision 28879 built 2020-03-24</td>
</tr>
</tbody>
</table>

The “Preview Release” of 3D Slicer is updated daily (process starts at 11pm ET and takes few hours to complete) and represents the latest development including new features and fixes. The “Stable Release” is usually updated a few times a year and is more rigorously tested.

Slicer is generally simple to install on all platforms. It is possible to install multiple versions of the application on the same user account and they will not interfere with each other. If you run into mysterious problems with your installation you can try deleting the application settings files.

Once downloaded, follow the instructions below to complete installation:

2.2.1 Windows

• Run the installer
• Run Slicer from the Windows start menu
• Use “Apps & features” in Windows settings to remove the application

2.2.2 Mac

• Drag the Slicer application (Slicer.app) to your Applications folder or other location of your choice.
• You cannot install extensions into the read-only volume so you must copy before installing extensions.

• Delete the Slicer.app folder to uninstall

Installing Preview Release: Currently, preview release packages are not signed. Therefore, when the application is started the first time the following message is displayed: “Slicer... can’t be opened because it is from an unidentified developer”. To resolve this error, locate the application in Finder and right-click (two-finger click) and click Open. When it says This app can’t be opened go ahead and hit cancel. Right click again and say Open (yes, you need to repeat the same as you did before - the outcome will be different than the first time). Click the Open (or Open anyway) button to start the application. See more explanation and alternative techniques here.

2.2.3 Linux

• Open the tar.gz archive and copy directory to the location of your choice. Run the Slicer executable.

• Remove the directory to uninstall

The following may be needed on fresh debian or ubuntu:

```
sudo apt-get install libpulse-dev libnss3 libglu1-mesa
```

2.3 Using Slicer

3D Slicer offers lots of features and gives users great flexibility in how to use them. As a result, new users may be overwhelmed with the number of options and have difficulty figuring out how to perform even simple operations. This is normal and many users successfully crossed this difficult stage by investing some time into learning how to use this software.

How to learn Slicer?

2.3.1 Quick start

You may try to figure out how the application works by loading data sets and explore what you can do.

Load data

Open 3D Slicer and using the Welcome panel either load your own data or download sample data to explore. Sample data is often useful for trying the features of 3D Slicer if you don’t have data of your own.
Chapter 2. Getting Started

Load Data

Welcome

Sample data

Load DICOM Data
Load Data
Customize Slicer
Download Sample Data

Feedback

Share your stories with us and let us know about how 3D Slicer has enabled your research.

We are always interested in improving 3D Slicer, and every submission will be carefully read.

See more at http://goo.gl/6BvcHm.

- About
- The Main Window
- Loading and Saving
- Display
2.3. Using Slicer

- MRHead
- CTChest
- CTACardio
- DTIBrain
- MRBrainTumor1
- MRBrainTumor2
- BaselineVolume
- DTIVolume
- DWIVolume
- CTA abdomen (Panoramix)
- CBCTDentalSurgery
- MR-US Prostate
- CT-MR Brain
View data

You can customize views by clicking on the push pin in the top left corner of viewer. In the slice viewers, the horizontal bar can be used to scroll through slices or select a slice.

Process data

3D Slicer is built on a modular architecture. Choose a module to process or analyze your data. Most important modules are the followings (complete list is available in Modules section):

- **Welcome**: The default module when 3D Slicer is started. The panel features options for loading data and customizing 3D Slicer. Below those options are drop-down boxes that contain essential information for using 3D Slicer.
- **Data**: acts as a central data-organizing hub. Lists all data currently in the scene and allows basic operations such as search, rename, delete and move.
- **DICOM**: Import and export DICOM objects, such as images, segmentations, structure sets, radiation therapy objects, etc.
- **Volumes::** Used for changing the appearance of various volume types.
- **Volume Rendering**: Provides interactive visualization of 3D image data.
- **Segmentations**: Edit display properties and import/export segmentations.
- **Segment Editor**: Segment 3D volumes using various manual, semi-automatic, and automatic tools.
- **Markups**: Allows the creation and editing of markups associated with a scene. Currently, lists of fiducially are supported as markups.
- **Models**: Loads and adjusts display parameters of models. Allows the user to change the appearance of and organize 3D surface models.
• **Transforms**: This module is used for creating and editing transformation matrices. You can establish these relations by moving nodes from the Transformable list to the Transformed list or by dragging the nodes under the Transformation nodes in the Data module.

**Extensions**

3D Slicer supports plug-ins that are called extensions. An extension could be seen as a delivery package bundling together one or more Slicer modules. After installing an extension, the associated modules will be presented to the user as built-in ones. Extensions can be downloaded from the extension manager to selectively install features that are useful for the end-user.

For details about downloading extensions, see Extension Manager documentation. Click [here](#) for a full list of extensions. The links on the page will provide documentation for each extension.

Slicer is extensible. If you are interested in customizing or adding functionality to Slicer, click [here](#).
2.3.2 Tutorials

You learn both basic concepts and highly specialized workflows from the numerous available step-by-step and video tutorials.

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

For more tutorials, visit the Tutorial page.

2.3.3 User manual

Browse the User Guide section to find quick overview of the application user interface or Modules section for detailed description of each module.

2.3.4 Ask for help

3D Slicer has been around for many years and many questions have been asked and answered about it already. If you have any questions, then you may start with a web search, for example Google `slicer load jpg` to find out how you can import a stack of jpg images.

The application has a large and very friendly and helpful user community. We have people who will happy to help with simple questions, such as how to do a specific task in Slicer, and we have a large number of engineering and medical experts who can give you advice with how to solve complex problems.

If you have any questions, go to the Slicer forum and ask us!
3.1 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 data nodes in the scene. Slicer package contains over 100 built-in modules and additional modules can be installed by using the Extension Manager.
3.1.1 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.

3.1.2 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.

3.1.3 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**.

### 3.1.4 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 drop-down button** 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.

### 3.1.5 Status bar

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

### 3.2 Interacting with views

#### 3.2.1 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.

#### 3.2.2 Mouse Modes

Slicer has multiple mouse modes: **Transform** (which allows interactive rotate, translate and zoom operations), **Window/Level** to adjust brightness/contrast of the image volumes, 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 other nodes such as 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.

Adjusting image window/level

Medical images typically contain thousands of gray levels, but regular computer displays can display only 256 gray levels, and the human eye also has limitation in what minimum contrast difference it can notice (see Kimpe 2007 for more specific information). Therefore, medical images are displayed with adjustable brightness/contrast (window/level).

By default 3D Slicer uses window/level setting that is specified in the DICOM file. If it is not available then window/level is set so that the entire intensity range of the image (except top/bottom 0.1%, to not let a very thin tail of the intensity distribution to decrease the image contrast too much).

Window/level can be manually adjusted anytime by clicking on “Adjust window/level” button on the toolbar then left-click-and-drag in any of the slice viewers. Optimal window/level can be computed for a chosen area by left-click-and-dragging while holding down Ctrl key.

3.2.3 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.

### 3.2.4 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.

- **Reset field of view** (small square) centers the slice on the current background volume
- **Link** button synchronizes properties (which volumes are displayed, zoom factor, position of parallel views, opacities, etc.) between all slice views in the same view group. Long-click on the button exposes **hot-linked** option, which controls when properties are synchronized (immediately or when the mouse button is released).
- **Eye** button in the top row can show the current slice in 3D views. Drop-down menu of the button contains advanced options to customize how this slice is rendered: “…match volume” means that the properties are taken from the full volume, while “…match 2D” means that the properties are copied from the current slice view (for example, copies zoom and pan position). Typically these differences are subtle and the settings can be left at default.
- **Orientation Selection** displays allows you to choose the orientation for this slice view.
- **Lightbox** to select a mosaic (a.k.a. contact sheet) view. Not all operations work in this mode and it may be removed in the future.
- **Reformat** allows interactive manipulation of the slice orientation.
- **Blending options** how foreground and background layers are mixed.
- **Spacing and Field of View** Spacing defines the increment for the slice offset slider. Field of view sets the zoom level for the slice.
- **Rotate to Volume Plane** changes the orientation of the slice to match the closest acquisition orientation of the displayed volume
• **Show Orientation Marker** controls display of human, cube, etc in lower right corner
• **Ruler** controls display of ruler in slice view

### 3.3 Mouse & Keyboard Shortcuts

#### 3.3.1 Generic shortcuts

#### 3.3.2 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.

#### 3.3.3 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.

**Note:** Simulation if 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 loaded in two steps: 1. import files into the application’s DICOM database (by switching to DICOM module and drag-and-dropping files to the application window), 2. load data objects (by double-clicking on items in the DICOM browser). 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.
CHAPTER 5

Image Segmentation
5.1 Basic concepts

Segmentation of images (also known as contouring or annotation) is a procedure to delineate regions in the image, typically corresponding to anatomical structures, lesions, and various other object space. It is a very common procedure in medical image computing, as it is required for visualization of certain structures, quantification (measuring volume, surface, shape properties), 3D printing, and masking (restricting processing or analysis to a specific region), etc.

Segmentation may be performed manually, for example by iterating through all the slices of an image and drawing a contour at the boundary; but often semi-automatic or fully automatic methods are used. Segment Editor module offers a wide range of segmentation methods.

Result of a segmentation is stored in segmentation node in 3D Slicer. A segmentation node consists of multiple segments.

A segment specifies region for a single structure. Each segment has a number of properties, such as name, preferred display color, content description (capable of storing standard DICOM coded entries), and custom properties. Segments may overlap each other in space.

A region can be represented in different ways, for example as a binary labelmap (value of each voxel specifies if that voxel is inside or outside the region) or a closed surface (surface mesh defines the boundary of the region). There is no one single representation that works well for everything: each representation has its own advantages and disadvantages and used accordingly.

Each segment stored in multiple representations. One representation is designated as the master representation (marked with a “gold star” on the user interface). The master representation is the only editable representation, it is the only one that is stored when saving to file, and all other representations are computed from it automatically.

Binary labelmap representation is probably the most commonly used representation because this representation is the easiest to edit. Most software that use this representation, store all segments in a single 3D array, therefore each voxel can belong to a single segment: segments cannot overlap. In 3D Slicer, overlapping between segments is allowed. To store overlapping segments in binary labelmaps, segments are organized into layers. Each layer is stored internally as a separate 3D volume, and one volume may be shared between many non-overlapping segments to conserve memory.

5.2 3D Slicer modules for segmentation

There are many modules in 3D Slicer for manipulating segmentations.

5.2.1 Segmentations module

Features:

- Create/delete segmentation nodes
- Create/delete/update representations
- Edit display properties
- Copy/move segments between segmentation nodes
- Import/export of segments to different MRML nodes
- File export of closed surface representation
5.2.2 Segment editor module

- Edit labelmap representation
- Shortcuts to common operations (show/hide in 3D, edit smoothing options, file export of closed surface representation)
5.2.3 Segment statistics module

Computes basic properties for each segment, such as volume, surface, minimum/maximum/mean intensity, shape properties.

5.2.4 Segment comparison module

Compute similarity between two segments based on metrics such as Hausdorff distance and Dice coefficient. Provided by SlicerRT extension.

5.2.5 Segment registration module

Compute rigid or deformable transform that aligns two selected segments. Provided by SegmentRegistration extension.
CHAPTER 6

Modules

6.1 Overview

6.2 Main

6.2.1 Data

Overview

Data module shows all data sets loaded into the scene and allows modification of basic properties and perform common operations on all kinds of data, without switching to other modules.

- **Subject Hierarchy** tab shows selected nodes in a freely editable folder structure.
- **Transform Hierarchy** tab shows data organized by what transforms are applied to them.
- **All Nodes** tab shows all nodes in simple list. This is intended for advanced users and troubleshooting.

In Subject Hierarchy, DICOM data is automatically added as patient-study-series hierarchy. Non-DICOM data can be parsed if loaded from a local directory structure, or can be manually organized in tree structure by creating DICOM-like hierarchy or folders.

Subject hierarchy provides features for the underlying data nodes, including cloning, bulk transforming, bulk show/hide, type-specific features, and basic node operations such as delete or rename. Additional plugins provide other type-specific features and general operations, see **Subject hierarchy labs page**.

- **Subject hierarchy view**
  - Overview all loaded data objects in the same place, types indicated by icons
  - Organize data in folders or patient/subject trees
  - Visualize and bulk-handle lots of data nodes loaded from disk
  - Easy show/hide of branches of displayable data
– Transform whole study (any branch)
– Export DICOM data (edit DICOM tags)
– Lots of type-specific functionality offered by the plugins

• Transform hierarchy view
  – Manage transformation chains/hierarchies
• All nodes view
  – Developer tool for debugging problems

How to

Create new Subject from scratch

Right-click on the empty area and select ‘Create new subject’

Create new folder

Right-click on an existing item or the empty area and select ‘Create new folder’. Folder type hierarchy item can be converted to Subject or Study using the context menu

Rename item

Right-click on the node and select ‘Rename’, or double-click the name of a node

Apply transform on node or branch

Double-click the cell of the node or branch to transform in the transform column (same icon as Transforms module), then set the desired transform. If the column is not visible, check the ‘Transforms’ checkbox under the tree. An example can be seen in the top screenshot at Patient 2

Panels and their use

Subject hierarchy tab

Contains all the objects in the Subject hierarchy in a tree representation.
Folder structure:

- Nodes can be drag&dropped under other nodes, thus re-arranging the tree
- New folder or subject can be added by right-clicking the empty area in the subject hierarchy box
- Data loaded from DICOM are automatically added to the tree in the right structure (patient, study, series)
- Non-DICOM data also appears automatically in Subject hierarchy. There are multiple ways to organize them in hierarchy:
  - Use *Create hierarchy from loaded directory structure* action in the context menu of the scene (right-click on empty area, see screenshot below). This organizes the nodes according to the local file structure they have been loaded from.
  - Drag&drop manually under a hierarchy node
  - Create model or other (e.g. annotation) hierarchies, and see the same structure in subject hierarchy
Operations (accessible in the context menu of the nodes by right-clicking them):

- Common for all nodes:
  - **Show/hide** node or branch: Click on the eye icon
  - **Delete**: Delete both data node and SH node
  - **Rename**: Rename both data node and SH node
  - **Clone**: Creates a copy of the selected node that will be identical in every manner. Its name will contain a _Copy postfix
  - **Edit properties**: If the role of the node is specified (i.e. its icon is not a question mark), then the corresponding module is opened and the node selected (e.g. Volumes module for volumes)
  - **Create child...**: Create a node with the specified type
  - **Transform node or branch**: Double-click the cell of the node or branch to transform in the Applied transform column, then set the desired transform. If the column is not visible, check the ‘Transforms’ checkbox under the tree. An example can be seen in the top screenshot at ‘Day 2’ study

- Operations for specific node types:
  - **Volumes**: icon, Edit properties and additional information in tooltip
    * ‘Register this...’ action to select fixed image for registration. Right-click the moving image to initiate registration
• ‘Segment this…’ action allows segmenting the volume, for example, in the Editor module
• ‘Toggle labelmap outline display’ for labelmaps
  - **Models**: icon, Edit properties and additional information in tooltip
  - **SceneViews**: icon, Edit properties and Restore scene view
  - **Transforms**: icon, additional tooltip info, Edit properties, Invert, Reset to identity

Highlights: when an item is selected, the related items are highlighted. Meaning of colors:
  - Green: Items referencing the current item directly via DICOM or node references
  - Yellow: Items referenced by the current item directly via DICOM or node references
  - Light yellow: Items referenced by the current item recursively via node references

Subject hierarchy item information section: Displays detailed information about the selected subject hierarchy item.

**Transform hierarchy tab**

- **Nodes**: The view lists all transformable nodes of the scene as a hierarchical tree that describes the relationships between nodes. Nodes are graphical objects such as volumes or models that control the displays in the different views (2D, 3D). To rename an item, double click with the left button on any item (but the scene) in the list. A right click pops up a menu containing different actions: “Insert Transform” creates an identity linear transform node and applies it on the selected node. “Edit properties” opens the module of the node (e.g. “Volumes” for volume nodes, “Models” for model nodes, etc.). “Rename” opens a dialog to rename the node. “Delete” removes the node from the scene. Internal drag-and-drops are supported in the view, while moving a node position within the same parent has no effect, changing the parent of a node has a different meaning depending on the current scene model.
- **Show MRML ID’s**: Show/hide in the tree view a second column containing the node ID of the nodes. Hidden by default
- **Show hidden nodes**: Show/hide the hidden nodes. By default, only the main nodes are shown
All nodes tab

List of all nodes in the scene. Supports Edit properties, Rename, Delete.
Common section for all tabs

- **Filter**: Hide all the nodes not matching the typed string. This can be useful to quickly search for a specific node. Please note that the search is case sensitive.

- **MRML node information**: Attribute list of the currently selected node. Attributes can be edited (double click in the “Attribute value” cell), added (with the “Add” button) or removed (with the “Remove” button).

Tutorials

- **2016**: This tutorial demonstrates the basic usage and potential of Slicer’s data manager module Subject Hierarchy using a two-timepoint radiotherapy phantom dataset.
- **2015**: Tutorial about loading and viewing data.

Information for developers

- Code snippets accessing and manipulating subject hierarchy items can be found in the script repository.
- **Implementing new plugins**: Plugins are the real power of subject hierarchy, as they provide support for data node types, and add functionality to the context menu items. To create a C++ plugin, implement a child class of qSlicerSubjectHierarchyAbstractPlugin, for Python plugin see below. Many examples can be found in Slicer core and in the SlicerRT extension, look for folders named SubjectHierarchyPlugins.
Writing plugins in **Python**:

- Child class of AbstractScriptedSubjectHierarchyPlugin which is a Python adaptor of the C++ qSlicer-SubjectHierarchyScriptedPlugin class
- Example: Annotations role plugin, function plugin

**Role** plugins: add support for new data node types

- Defines: ownership, icon, tooltip, edit properties, help text (in the yellow question mark popup), visibility icon, set/get display visibility, displayed node name (if different than name of the node object)
- Existing plugins in Slicer core: Markups, Models, SceneViews, Charts, Folder, Tables, Transforms, LabelMaps, Volumes

**Function** plugins: add feature in right-click context menu for certain types of nodes

- Defines: list of context menu actions for nodes and the scene, types of nodes for which the action shows up, functions handling the defined action
- Existing plugins in Slicer core: CloneNode, ParseLocalData, Register, Segment, DICOM, Volumes, Markups, Models, Annotations, Segmentations, Segments, etc.

**References**

- Additional information on Subject hierarchy labs page
- Manual editing of segmentations can be done in the Segment Editor module

**Contributors**

End-user advocate: Ron Kikinis (SPL, NA-MIC)

Authors:

- Csaba Pinter (PerkLab, Queen’s University)
- Julien Finet (Kitware)
- Alex Yarmarkovich (Isomics)
- Nicole Aucoin (SPL, BWH)

**Acknowledgements**

This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149. This work was also funded by An Applied Cancer Research Unit of Cancer Care Ontario with funds provided by the Ministry of Health, Canada.
6.2.2 Markups

Overview

This module is used to create and edit markups (fiducial list, line, angle, curve, plane, etc.) and adjust their display properties.

How to

Place new markups

1. Click “Create and place” button on the toolbar to activate place mode.

Click the down-arrow icon on the button to choose markups type.
Click “Persistent” checkbox to keep placing more points after the current markups is completed.

1. Left-click in the slice or 3D views to place points
2. Right-click to finish point placement

**Edit point positions in existing markups**

- Make sure that the correct markups node is selected in Markups module.
- Left-click-and drag a control point to move it
- Left-click a control point to show it in all slice viewers. This helps in adjusting its position along all axes.
- Right-click to delete or rename a point or change node properties.
- Ctrl + Left-click to place a new point on a curve.
- Enable Display / Interaction / Visible to show a widget that allows translation/rotation of the entire widget.

**Edit properties of a markups node that is picked in a view**

To pick a markups node in a viewer so that its properties can be edited in Markups module, right-click on it in a slice or 3D view and choose “Edit properties”.

![Edit properties](image)
Panels and their use

- Create: click on any of the buttons to create a new markup. Click in any of the viewers to place markup points.
- Markups list: Select a markups node to populate the GUI and allow modifications. When markup placement is activated on the toolbar, then points are added to this selected markups node.

Display section:

- Visibility: Toggle the markups visibility, which will override the individual markup visibility settings. If the eye icon is open, the list is visible, and pressing it will make it invisible. If the eye icon is closed, the list is invisible and pressing the button will make it visible.
- Opacity: overall opacity of the selected markups.
- Glyph Size: Set control point glyph size relative to the screen size (if absolute button is not pressed) or as an absolute size (if absolute button is depressed).
- Text Scale: Label size relative to screen size.
- Interaction: check Visible to enable translation of the entire markups in slice and 3D views with an interactive widget.
- Advanced:
  - View: select which views the markups is displayed in
  - Selected Color: Select the color that will be used to display the glyph and text when the markup is marked as selected.
  - Unselected Color: Select the color that will be used to display the glyph and text when the markup is not marked as selected.
  - Glyph Type: Select the symbol that will be used to mark each location. Default is the Sphere3D.
  - Point Labels: check to display label for each control point.
  - Fiducial projection: A widget that controls the projection of the fiducials in the 2D viewers onto slices around the one on which the fiducial has been placed.
    - 2D Projection: Toggle the eye icon open or closed to enable or disable visualization of projectied fiducial on 2D viewers.
    - Use Fiducial Color: If checked, color the projections the same color as the fiducials.
    - Projection Color: If not using fiducial color for the projection, use this color.
    - Outlined Behind Slice Plane: Fiducial projection is displayed filled (opacity = Projection Opacity) when on top of slice plane, outlined when behind, and with full opacity when in the plane. Outline isn’t used for some glyphs (Dash2D, Cross2D, Starburst).
    - Projection Opacity: A value between 0 (invisible) and 1 (fully visible) for displaying the fiducial projection.
    - Reset to Defaults: Reset the display properties of this markups node to the system defaults.
    - Save to Defaults: Save the display properties of this markups node to be the new system defaults.

Control points section:

- Interaction in views: toggle the markups lock state (if it can be moved by mouse interactions in the viewers), which will override the individual markup lock settings.
- Click to Jump Slices: If checked, click in name column to jump slices to that point. The radio buttons control if the slice is centered on the markup or not after the jump. Right click in the table allows jumping 2D slices
to a highlighted fiducial (uses the center/offset radio button settings). Once checked, arrow keys moving the highlighted row will also jump slice viewers to that markup position.

• **Buttons:** These buttons apply changes to markups in the selected list.
  
  − **Toggle visibility flag:** Toggle visibility flag on all markups in list. Use the drop down menu to set all to visible or invisible.

  − **Toggle selected flag:** Toggle selected flag on all markups in list. Use the drop down menu to set all to selected or deselected. Only selected markups will be passed to command line modules.

  − **Toggle lock flag:** Toggle lock flag on all markups in list, use drop down menu to set all to locked or unlocked.

  − **Delete the highlighted markups from the active list:** After highlighting rows in the table to select markups, press this button to delete them from the list.

  − **Remove all markups from the active list:** Pressing this button will delete all of the markups in the active list, leaving it with a list length of 0.

  − **Transformed:** Check to show the transformed coordinates in the table. This will apply any transform nodes to the points in the list and show that result. Keep unchecked to show the raw RAS values that are stored in MRML. If you harden the transform the transformed coordinates will be the same as the non transformed coordinates.

  − **Hide RAS:** Check to hide the coordinate columns in the table and uncheck to show them. Right click in rows to see coordinates.

• **Control points table:** Right click on rows in the table to bring up a context menu to show the full precision coordinates, distance between multiple highlighted fiducials, delete the highlighted markup, jump slice viewers to that location, refocus 3D viewers to that location, or if there are other lists, to copy or move the markup to another list.

  − **Selected:** A check box is shown in this column, it’s check state depends on if the markup is selected or not. Click to toggle the selected state. Only selected markups will be passed to command line modules.

  − **Locked:** An open or closed padlock is shown in this column, depending on if the markup is unlocked or locked. Click it to toggle the locked state.

  − **Visibility:** An open or closed eye icon is shown in this column, depending on if the markup is visible or not. Click it to toggle the visibility state.

  − **Name:** A short name for this markup, displayed in the viewers as text next to the glyphs.

  − **Description:** A longer description for this markup, not displayed in the viewers.

  − **X, Y, Z:** The RAS coordinates of this markup, 3 places of precision are shown.

• **Advanced section:**

  − **Move Up:** Move a highlighted markup up one spot in the list.

  − **Move Down:** Move a highlighted markup down one spot in the list.

  − **Add Markup:** Add a new markup to the selected list, placing it at the origin

  − **Naming:**

    * **Name Format:** Format for creating names of new markups, using sprintf format style. %N is replaced by the list name, %d by an integer.

    * **Apply:** Rename all markups in this list according to the current name format, trying to preserve numbers. A quick way to re-number all the fiducials according to their index is to use a name format with no number in it, rename, then add the number format specifier %d to the format and rename one more time.
Curve settings section:

- Curve type:
  - linear: control points are connected with straight line
  - spline, Kochanek spline: smooth interpolating curve
  - polynomial: smooth approximating curve
  - shortest distance on surface: curve points are forced to be on the selected model’s surface, connected with a minimal-cost path

- Surface: surface used for shortest distance on surface curve type and cost function that is minimized to find path connecting two control points

Resample section: Replace control points by curve points sampled at equal distances. If a model node is selected for Constrain points to surface then the resampled points will be projected to the chosen model surface.

Information for developers

Markups module can be used from other modules as demonstrated in examples in the Script repository.
The Simple Markups Widget can be integrated into slicelets. It is lightweight and can access the Markups features. An example of this use is in Gel Dosimetry. To use this, access it at GitHub.

Markups fiducial point list file format

The Markups Fiducial storage node uses a comma separated value file to store the fiducials on disk. The format is:

A leading comment line giving the Slicer version number:

```
# Markups fiducial file version = 4.11
```

A comment line giving the coordinate system. In earlier versions of Slicer, numeric codes were used: RAS = 0, LPS = 1.

```
# CoordinateSystem = LPS
```

A comment line explaining the fields in the csv

```
# columns = id,x,y,z,ow,ox,oy,oz,vis,sel,lock,label,desc,associatedNodeID
```

Then comes the fiducials, one per line, for example:

```
vtkMRMLMarkupsFiducialNode_0,1.29,-40.18,60.15,0,0,0,1,1,1,0,F-1,,
```

- id = a string giving a unique id for this fiducial, usually based on the class name
- x,y,z = the floating point coordinate of the fiducial point
- ow,ox,oy,oz = the orientation quaternion of this fiducial, angle and axis, default 0,0,0,1 (or 0,0,0,0,0,1.0)
- vis = the visibility flag for this fiducial, 0 or 1, default 1
- sel = the selected flag for this fiducial, 0 or 1, default 1
- lock = the locked flag for this fiducial, 0 or 1, default 0
3D Slicer Documentation

- label = the name for this fiducial, displayed beside the glyph, with quotes around it if there is a comma in the field
- desc = a string description for this fiducial, optional
- associatedNodeID = an id of a node in the scene with which the fiducial is associated, for example the volume or model on which the fiducial was placed, optional

Related modules

- This module will replace Annotations module.
- Endoscopy module uses fiducials

References

- History and design considerations

Contributors

Authors:
- Andras Lasso (PerkLab, Queen’s University)
- Davide Punzo (Kapteyn Astronomical Institute, University of Groningen)
- Kyle Sunderland (PerkLab, Queen’s University)
- Nicole Aucoin (SPL, BWH)

Acknowledgements

This 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. Information on NA-MIC can be obtained from the NA-MIC website.

6.2.3 Segmentations

Overview

The Segmentations module manages segmentations. Each segmentation can contain multiple segments, which correspond to one structure or ROI. Each segment can contain multiple data representations for the same structure, and the module supports automatic conversion between these representations (the default ones are: planar contour, binary labelmap, closed surface model), as well as advanced display settings and import/export features.

- Visualization of structures in 2D and 3D views
- Define regions of interest as input to further analysis (volume measurements, masking for computing radiomics features, etc.)
- Create surface meshes from images for 3D printing
• Editing of 3D closed surfaces

Motivation, features, and details of the infrastructure are explained in these slides (source).

Use cases

Edit segmentation

Segmentation can be edited using Segment Editor module.

Import an existing segmentation from volume file

3D volumes in NRRD (.nrrd or .nhdr) and Nifti (.nii or .nii.gz) file formats can be directly loaded as segmentation:

• Drag-and-drop the volume file to the application window (or use menu: File/Add Data, then select the file)
• In Description column choose Segmentation
• Click OK

Other volume file formats can be loaded as labelmap volume and then converted to segmentation node:

• Drag-and-drop the volume file to the application window (or use menu: File/Add Data, then select the file)
• Click Show Options
• In Options column check LabelMap checkbox (to indicate that the volume is a labelmap, not a grayscale image)
• Click OK
• Go to Data module, Subject hierarchy tab
• Right-click on the name of the imported volume and choose Convert labelmap to segmentation node

Tip: To show the segmentation in 3D, go to Segmentations module and in Representations section, Closed surface row, click Create. Or, during segment editing in Segment Editor module click Show 3D button to show/hide segmentation in 3D.

Import an existing segmentation from model (surface mesh) file

3D models in STL and OBJ formats can be directly loaded as segmentation:

• Drag-and-drop the volume file to the application window (or use menu: File/Add Data, then select the file)
• In Description column choose Segmentation
• Click OK

Other mesh file formats can be loaded as model and then converted to segmentation node:

• Drag-and-drop the volume file to the application window (or use menu: File/Add Data, then select the file)
• Click OK
• Go to Data module, Subject hierarchy tab
• Right-click on the name of the imported volume and choose Convert model to segmentation node
Tip: Selection of a master volume is required for editing a segmentation. If no volume is available then it can be created by:

- Go to Segment editor module
- Click Specify geometry button (on the right side of Master volume node selector)
- In the displayed Segmentation geometry window click OK

Export segmentation to model (surface mesh) file

Segments can be exported to STL or OBJ files for 3D printing or visualization/processing in external software:

- Open Export to files section in Segmentations module (or in Segment editor module: choose Export to files, in the drop-down menu of Segmentations button)
- Choose destination folder, file format, etc.
- Click Export

Other file formats:

- Go to Data module, right-click on the segmentation node, and choose Export visible segments to models (alternatively, use Segmentations module’s Export/import models and labelmaps section)
- In application menu, choose File / Save
- Select File format
- Click Save

Export segmentation to labelmap volume

If segments in a segmentation do not overlap each other then segmentation is saved as a 3D volume node by default when the scene is saved (application menu: File / Save). If the segmentation contains overlapping segments then it is saved as a 4D volume: each 3D volume containing a set of non-overlapping segments.

To force saving segmentation as a 3D volume, export it to a labelmap volume by right-clicking on the segmentation in Data module.

For advanced export options, Segmentations module’s Export/import models and labelmaps section can be used. If exported segmentation geometry (origin, spacing, axis directions, extents) must exactly match another volume’s then then choose that volume as Reference volume in Advanced section.

Create new representation in segmentation (conversion)

The supported representations are listed as rows in the Representations section. The already existing representations have a green tick, the master representation (that is the source of all conversions and the representation that can be edited) a gold star.

To create a representation that is missing, click on the Create button in its row. If custom conversion path or parameters are to be used (reference grid for labelmaps, smoothing for surfaces, etc.), long-press the button and choose “Advanced create...”. In the appearing window the available conversion paths are listed in a list, ordered by estimated time cost. After selecting the desired path, the involved parameters are listed in the bottom section of the window.

To re-convert an existing representation (to use different conversion path or parameters), click the Update button in their row.
Adjust how segments are displayed

- By right-clicking the eye icon in the segments table the display options are shown and the different display modes can be turned on or off
- Advanced display options are available in Segmentations module’s Display sections.

Managing segmentations using Python scripts

See Script repository’s Segmentations section for examples.

DICOM export

- The master representation is used when exporting into DICOM
- DICOM RT structure set export is available if SlicerRT extension is installed
- DICOM Segmentation Object export if QuantitativeReporting extension is installed

Panels and their use

- Segments table
  - Add/remove segments
  - Edit selected: takes user to Segment Editor module
  - Set visibility and per-segment display settings, opacity, color, segment name
- Display
  - Segmentations-wide display settings (not per-segment!): visibility, opacity (will be multiplied with per-segment opacity for display)
  - Views: Individual views to show the active segmentation in
  - Slice intersection thickness
  - Representation in 3D/2D views: The representation to be shown in the 3D and 2D views. Useful if there are multiple representations available, for example if we want to show the closed surface in the 3D view but the labelmap in the slice views
- Representations
  - List of supported representations and related operations
  - The already existing representations have a green tick, the master representation (that is the source of all conversions and the representation that can be edited) a gold star
  - The buttons in each row can be used to create, remove, update a representation
    - Advanced conversion is possible (to use the non-default path or conversion parameters) by long-pressing the Create or Update button
    - Existing representations can be made master by clicking Make master. The master representation is used as source for conversions, it is the one that can be edited, and saved to disk
- Copy/move (import/export)
  - Left panel lists the segments in the active segmentation
– Right panel shows the external data container
– The arrow buttons can be used to copy (with plus sign) or move (no plus sign) segments between the segmentation and the external node
– New labelmap or model can be created by clicking the appropriate button on the top of the right panel
– Multiple segments can be exported into a labelmap. In case of overlapping segments, the subsequent segments will overwrite the previous ones!

Subject hierarchy

• Segmentations are shown in subject hierarchy as any other node, with the exception that the contained segments are in a “virtual branch”.
  – The segments can be moved between segmentations, but drag&drop to anywhere other than under another segmentation is not allowed

• Special subject hierarchy features for segmentations
  – Create representation: Create the chosen representation using the default path

• Special subject hierarchy features for segments
  – Show only this segment: Useful if only one segment needs to be shown and there are many, so clicking the eye buttons would take a long time
  – Show all segments

Tutorials

• Segmentation tutorials

Information for developers

• vtkSegmentationCore on GitHub (within Slicer)
• Segmentations Slicer module on GitHub
• Segmentations Labs page
• Manipulation of segmentations from Python scripts - examples in script repository

Related modules

• Segment Editor module is for editing segments of a segmentation node
• Editor module: the legacy Editor module has been replaced by Segment Editor module.

References

• Development notes
6.2.4 Segment editor

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 legacy 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 Segmentations module.

Keyboard shortcuts

The following keyboard shortcuts are active when you are in the Editor module. They are intended to allow two-handed editing, where on hand is on the mouse and the other hand uses the keyboard to switch modes.

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 (extent, 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: changing the master volume does not affect the segmentation’s labelmap
representation geometry. To make changes to the geometry (make the extent larger, the resolution finer, etc.) click “Specify geometry” button next to the master volume selector, select a “Source geometry” node that will be used as a basis for the new geometry, adjust parameters, and click OK. To specify an arbitrary extent, an ROI (region of interest) node can be created and selected as source geometry.

- **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.

**Tutorials**

- **Segmentation tutorials**

**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.
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.

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

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.

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

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 and auto-complete method will be locked after initialization, therefore if either 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.

**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 and auto-complete method will be locked after initialization, therefore if either 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.
3D Slicer Documentation

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://insightjournal.org/browse/publication/977

**Threshold**

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

**Margin**

Grows or shrinks the selected segment by the specified margin.
3D Slicer Documentation

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

**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

**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.

**Logical operators**
Apply Boolean operators to selected segment or combine segments.

**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).

**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.
Related Modules

- **Segment Statistics** module computes volume, surface, mean intensity, and various other metrics for each segment.
- **Segmentations** module allows changing visualization options, exporting/importing segments to/from other nodes (models, labelmap volumes), and moving or copying segments between segmentation nodes.
- **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.
- Editor module is the predecessor of this module. Segment Editor provides all its features and many more.

Information for Developers

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

Contributors

Authors:

- Csaba Pinter (PerkLab, Queen’s University)
- Andras Lasso (PerkLab, Queen’s University)
- Kyle Sunderland (PerkLab, Queen’s University)
- Steve Pieper (Isomics Inc.)
- Wendy Plesniak (SPL, BWH)
- Ron Kikinis (SPL, BWH)
- Jim Miller (GE)

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 <http://www.na-mic.org/> (NA-MIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.
6.2.5 Volume rendering

Overview

Volume rendering (also known as volume ray casting) is a visualization technique for displaying image volumes as 3D objects directly - without requiring segmentation.

This is accomplished by specifying color and opacity for each voxel, based on its image intensity. Several presets are available for this mapping, for displaying bones, soft tissues, air, fat, etc. on CT and MR images. Users can fine-tune these presets for each image.

Use cases

Display a CT or MRI volume

- Load a data set (for example, use Sample Data module to load CT Chest data set)
- Go to Data module and right-click on the eye icon, and choose “Show in 3D views as volume rendering”
- Right-click on the eye icon and choose “Volume rendering options” to switch to edit visualization options in Volume rendering module
- Choose a different preset in Display section,
- Adjust “Offset” slider to change what image intensity range is visible
Render different volumes in two views

- Open the “Inputs” section
- Select the first volume
- Click View list and uncheck “View2” (only “View1” should be checked)
- Click the eye icon for the volume to show up in View1
- Select the second volume
- Click View list and uncheck “View1” (only “View2” should be checked)
- Click the eye icon for the volume to show up in View2

Limitations

- Only single-component scalar volumes can be used for volume rendering. Vector to Scalar Volume module can convert vector volume to scalar volume.
- To render multiple overlapping volumes, select “VTK Multi-Volume” rendering in “Display” section. Currently, no cropping can be applied in this mode.
• To reduce staircase artifacts during rendering, choose enable “Surface smoothing” in Advanced/Techniques/Advanced rendering properties section, or choose “Normal” or “Maximum” as quality.

Panels and their use

• Inputs: Contains the list of nodes required for VolumeRendering. It is unlikely that you need to interact with controllers.
  - Volume: Select the current volume to render. Note that only one volume can be rendered at a time.
  - Display: Select the current volume rendering display properties. Volume rendering display nodes contains all the information relative to volume rendering. They contain pointers to the ROI, volume property and view nodes. A new display node is automatically created if none exist for the current volume.
  - ROI: Select the current ROI to optionally crop with 6 planes the volume rendering.
  - Property: Select the current Volume Property. Volume properties contain the opacity, color and gradient transfer functions for each component.
  - View: Select the 3D views where the volume rendering must be displayed into. If no view is selected, the volume rendering is visible in all views

• Display: Main properties for the volume rendering.
  - Preset: Apply a pre-defined set of functions for the opacity, color and gradient transfer functions. The generic presets have been tuned for a combination of modalities and organs. They may need some manual tuning to fit your data.
  - Shift: Move all the inner points (first and last excluded) of the current transfer functions to the right/left (lower/higher). It can be useful when a preset defines a ramp from 0 to 200 but your data requires a ramp from 1000 to 1200.
  - Crop: Simple controls for the cropping box (ROI). More controls are available in the “Advanced…” section. Enable/Disable cropping of the volume. Show/Hide the cropping box. Reset the box ROI to the volume’s bounds.
  - Rendering: Select a volume rendering method. A default method can be set in the application settings Volume Rendering panel.
    * VTK CPU Ray Casting (default): Available on all computers, regardless of capabilities of graphics hardware. The volume rendering is entirely realized on the CPU, therefore it is slower than other options.
    * VTK GPU Ray Casting: Uses graphics hardware for rendering, typically much faster than CPU volume rendering. This is the recommended method for computers that have sufficient graphics capabilities. It supports surface smoothing to remove staircase artifacts.
    * VTK Multi-Volume: Uses graphics hardware for rendering. Can render multiple overlapping volumes. Currently does not support cropping.

• Advanced: More controls to control the volume rendering. Contains 3 tabs: “Techniques”, “Volume Properties” and “ROI”
  - Techniques: Advanced properties of the current volume rendering method.
    * Quality:
      - Adaptive (default): quality is reduced while interacting with the view (rotating, changing volume rendering settings, etc.).
      - Interactive speed: Ensure the given frame per second (FPS) is enforced in the views during interaction. The higher the FPS, the lower the resolution of the volume rendering
- Normal: fixed rendering quality, should work well for volumes that the renderer can handle without difficulties.
- Maximum: oversamples the image to achieve higher image quality, at the cost of slowing down the rendering.

* Technique:
  - Composite with shading (default): display as a shaded surface
  - Maximum intensity projection: display brightest voxel value encountered in each projection line
  - Minimum intensity projection: display darkest voxel value encountered in each projection line
* Surface smoothing: check this checkbox to reduce staircase artifacts by adding a random noise pattern (jitter) to the raycasting lines

- Volume Properties: Advanced views of the transfer functions.
  - Synchronize with Volumes module: show volume rendering with the same color mapping that is used in slice views
    - Click: Apply once the properties (window/level, threshold, lut) of the Volumes module to the Volume Rendering module.
    - Checkbox: By clicking on the checkbox, you can toggle the button. When toggled, any modification occurring in the Volumes module is continuously applied to the volume rendering
  - Control point properties: X = scalar value, O = opacity, M = mid-point, S = sharpness

* Keyboard/mouse shortcuts:
  - Left button click: Set current point or create a new point if no point is under the mouse.
  - Left button move: Move the current or selected points if any.
  - Right button click: Select/unselect point. Selected points can be moved at once
  - Right button move: Define an area to select points:
  - Middle button click: Delete point under the mouse cursor.
  - Right/Left arrow keys: Change of current point
  - Delete key: Delete the current point and set the next point as current
  - Backspace key: Delete the current point and set the previous point as current
  - ESC key: Unselect all points.

* Scalar Opacity Mapping: Opacity transfer function. Threshold mode: enabling threshold controls the transfer function using range sliders in addition to control points.
* Scalar Color Mapping: Color transfer function.
* Gradient Opacity: Gradient opacity transfer function. This controls the opacity according to how large a density gradient next to the voxel is.

* Advanced:
  - Interpolation: Linear (default for scalar volumes) or nearest neighbor (default for labelmaps) interpolation.
  - Shade: Enable/Disable shading. Shading uses light and material properties. Disable it to display X-ray-like projection.
  - Material: Material properties of the volume to compute shading effect.
– ROI: More controls for the cropping box.
  * Display Clipping box: Show hide the bounds of the ROI box.
  * Interactive mode: Control whether the cropping box is instantaneously updated when dragging the sliders or only when the mouse button is released.

**Contributors**

- Julien Finet (Kitware)
- Alex Yarmarkovich (Isomics)
- Csaba Pinter (PerkLab, Queen’s University)
- Andras Lasso (PerkLab, Queen’s University)
- Yanling Liu (SAIC-Frederick, NCI-Frederick)
- Andreas Freudling (SPL, BWH)
- Ron Kikinis (SPL, BWH)

**Acknowledgements**

This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149. Some of the transfer functions were contributed by Kitware Inc. (VolView)

---

6.2.6 Volumes
Overview

Volumes module provides basic information about volume nodes, can convert between volume types, and allows adjustment of display settings.

A volume node stores 3D array of elements (voxels) in a rectilinear grid. Grid axes are orthogonal to each other and can be arbitrarily positioned and oriented in physical space. Grid spacing (size of voxel) may be different along each axis.

Volume nodes have subtypes, based on what is stored in a voxel:

- **Scalar volume**: most common type of volume, voxels represent continuous quantity, such as a CT or MRI volume.
- **Labelmap volume**: each voxel can store a discrete value, such as an index or label; most commonly used for storing a segmentation, each label corresponds to a segment.
- **Vector volume**: each voxel stores multiple scalar values, such as RGB components of a color image, or RAS components of a displacement field.
- **Tensor volume**: each voxel stores a tensor, typically used for storing MRI diffusion tensor image.

Volumes module handles a 2D image as a single-slice 3D image. 4D volumes are represented as a sequence of 3D volumes, using Sequences extension.

Use cases

Display volume

Slice views: After loading a volume, it is displayed in slice views by default. If multiple volumes are loaded, Data module can be used to choose which one is displayed. Slice view controls allow further customization of which volume is displayed in which view and how.

3D views: Volumes can be displayed in 3D views using Volume rendering module. If structures of interest cannot be distinguished from surrounding regions then it may be necessary to segment the image using Segment Editor module and click Show 3D button.

Overlay two volumes

- Load two volumes
- Go to Data module
- Left-click on the “eye” icon of one of the volumes to show it as background volume
- Right-click on “eye” icon of the other volume and choose “Show in slice views as foreground”
- Adjust transparency of the foreground volume using the vertical slider in slice view controls. Click link button to make all changes applied to all slice views (in the same view group)

Load image file as labelmap volume

By default, a single-component image is loaded as scalar volume. To make Slicer interpret an image as labelmap volume, either of these options can be used:

- A. When the file is selected for loading, in Add data... dialog, check Show Options to see additional options, and check LabelMap checkbox in Volumes module.
• B. Before loading the file, rename it so that it contains `label` or `seg` in its name, for example: `something.label.nrrd`, `something-label.nrrd`, or `something-seg.nrrd`. This makes `LabelMap` checked by default.

• C. Load the file as scalar volume, and then convert it to labelmap volume by clicking Convert button at the bottom of Volume information section

If the goal is to load an image as labelmap volume so that it can be converted to segmentation, then there is simpler option available: choose `Segmentation` in Description column in Add data... window. This only available for nrrd and nifti images.

**Load a series of png, jpeg, or tiff images as volume**

• Choose from the menu: File / Add Data

• Click Choose File(s) to Add button and select any of the files in the sequence in the displayed dialog. Important: do not choose multiple files or the entire parent folder, just a single file of the sequence. All file names must start with a common prefix followed by a frame number (img001.tif, img002.tif,…). Number of rows and columns of the image must be the same in all files.

• Check Show Options and uncheck Single File option

• Click OK to load the volume

• Go to the Volumes module

• Choose the loaded image as Active Volume

• In the Volume Information section set the correct Image Spacing and Image Origin values

• Most modules require grayscale image as input. The loaded color image can be converted to a grayscale image by using the Vector to scalar volume module

These steps are also demonstrated in this video.

Note: Consumer file formats, such as jpg, png, and tiff are not well suited for 3D medical image storage due to the following serious limitations:

• Storage is often limited to bit depth of 8 bits per channel: this causes significant data loss, especially for CT images.

• No standard way of storing essential metadata: slice spacing, image position, orientation, etc. must be guessed by the user and provided to the software that imports the images. If the information is not entered correctly then the images may appear distorted and measurements on the images may provide incorrect results.

• No standard way of indicating slice order: data may be easily get corrupted due to incorrectly ordered or missing frames.

**Panels and their use**

• Active Volume: Select the volume to display and operate on.

• Volume Information: Information about the selected volume. Some fields can be edited to correctly describe the volume, for example, when loading incompletely specified image data such as a sequence of jpeg files. Use caution however, since changing properties such as Image Spacing will impact the physical accuracy of some calculations such as Label Statistics.

  – Image Dimensions: The number of pixels in “IJK” space - this is the way the data is arranged in memory. The IJK indices (displayed in the DataProbe) go from 0 to dimension-1 in each direction.
- Image Spacing: The physical distance between pixel centers when mapped to patient space expressed in millimeters.

- Image Origin: The location of the center of the 0,0,0 (IJK) pixel expressed with respect to patient space. Patient space is organized with respect to the subject’s Right, Anterior, and Superior anatomical directions. See coordinate systems page for more information.

- IJK to RAS Direction Matrix: The transform matrix from the IJK to RAS coordinate systems

- Center Volume: This button will apply a transform to the volume that shifts its center to the origin in patient space. Harden the transform on the volume to permanently change the image origin.

- Scan Order: Describes the image orientation (how the IJK space is oriented with respect to patient RAS.

- Number of Scalars: Most CT or MR scans have one scalar component (grayscale). Color images have three components (red, green, blue). Tensor images have 9 components. For diffusion weighted volumes this indicates the number of baseline and gradient volumes.

- Scalars Type: Tells the computer representation of each voxel. Volume module works with all types, but most modules expect scalar volumes. Vector volumes can be converted to scalar volumes using Vector to Scalar Volume module.

- Filename: Path to the file which this volume was loaded from/saved to

- Window/Level Presets: Loaded from DICOM headers defined by scanner or by technician.

- Convert to label map / Convert to scalar volume: Convert the active volume between labelmap and scalar volume.

  • Display: Set of visualization controls appropriate for the currently selected volume. Not all controls are available for all volume types.

  - Lookup Table: Select the color mapping for scalar volumes to colors.

  - Interpolate: When checked, slice views will display linearly interpolated slices through input volumes. Unchecked indicates nearest neighbor resampling

  - Window Level Presets: Predefinied shortcuts to window/level and color table combinations for common visualization requirements.

  - Window/Level Controls: Double slider with text input to define the range of input volume data that should be mapped to the display grayscale. Auto window level tries to estimate the intensity range of the foreground image data. On mouse over, a popup slides down to add support for large dynamic range by giving control over the range of the window level double slider.

  - Threshold: Controls the range of the image that should be considered transparent when used in the foreground layer of the slice display. Same parameters also control transparency of slice models displayed in the 3D viewers.

  - Histogram: Shows the number of pixels (y axis) vs the image intensity (x axis) over a background of the current window/level and threshold mapping.

  • Diffusion Weighted Volumes: The following controls show up when a DWI volume is selected

  - DWI Component: Selects the baseline or diffusion gradient direction volume to display.

  • Diffusion Tensor Volumes: The following controls show up when a DTI volume is selected

  - Scalar Mode: Mapping from tensor to scalar.

  - Slice Visibility: Allows display of graphics visualizations of tensors on one or more of the standard Red, Green, or Yellow slice views.

  - Opacity: How much of the underlying image shows through the glyphs.
– **Scalar Color Map**: How scalar measures of tensor are mapped to color.

– **Color by Scalar**: Which scalar metric is used to determine the color of the glyphs.

– **Scalar Range**: Defines the min-max range of the scalar mapping to color. When enabled, allows a consistent color mapping independent of the full range of the currently displayed item (if not selected color range will cover min-max of the currently displayed data).

– **Glyph Type**: Tubes and line show direction of eigen vector of tensor (major, middle, or minimum as selected by the Glyph Eigenvector parameter). Ellipsoid shows direction and relative scale of all three eigenvectors.

– **Scale Factor**: Controls size of glyphs. There are no physical units for this parameter.

– **Spacing**: Controls the number of glyphs on the slice view.

### Related modules

- **Volume rendering**: visualize volume in 3D views without segmentation
- **Segment editor**: delineate structures in the volume for analysis and 3D visualization
- **Vector to scalar volume**: convert vector volume to scalar volume
- **Extensions**:
  - **Image Maker**: create a volume from scratch

### Contributors

- Steve Piper (Isomics)
- Julien Finet (Kitware)
- Alex Yarmarkovich (Isomics)
- Nicole Aucoin (SPL, BWH)

### Acknowledgements

This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.
6.6 Segmentations

6.6.1 Segment statistics

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.

Labelmap statistics

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

- Voxel count: the number of voxels in the segment
- Volume mm³ the volume of the segment in mm³
- Volume cm³ the volume of the segment in cm³
- 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 mm²: the volume of the segment in mm²
- 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. See detailed definition here.
- 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. See detailed definition here.
- Elongation: the elongation of the segment. Calculated from square root of the ratio of the second largest principal moment by the second smallest. See detailed definition here.
- 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

Scalar volume statistics

• Voxel count: the number of voxels in the segment
• Volume mm³ the volume of the segment in mm³
• Volume cm³ the volume of the segment in cm³
• 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

Closed surface statistics

• Surface area mm²: the volume of the segment in mm²
• Volume mm³ the volume of the segment in mm³
• Volume cm³ the volume of the segment in cm³

Related Modules

• 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.)

Information for developers


Contributors

Authors:
• Csaba Pinter (PerkLab, Queen’s University)
• Andras Lasso (PerkLab, Queen’s University)
• Christian Bauer (University of Iowa)
• Steve Pieper (Isomics Inc.)
• Kyle Sunderland (PerkLab, Queen’s University)

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 <http://www.na-mic.org/> (NA-MIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

6.7 Quantification

6.8 Diffusion

6.9 Filtering

6.10 Utilities

6.10.1 DICOM Patcher

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.

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

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

Related Modules

- **DICOM** DICOM browser that lists all data sets in Slicer’s DICOM database.
Information for Developers

This is a Python scripted module. Source code is available here.

Contributors

Authors:

• Andras Lasso (PerkLab, Queen’s University)

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.

6.11 Surface models

6.12 Converters

6.13 Developer tools

6.14 Legacy

6.15 Testing

6.16 Sequences
CHAPTER 7

Extensions Manager
8.1 Images

Readers may support 2D, 3D, and 4D images of various types, such as scalar, vector, DWI or DTI, containing images, dose maps, displacement fields, etc.

- **DICOM** (.dcm, or any other): Slicer core supports reading and writing of some data types, while extensions add support for additional ones. Coordinate system: LPS (as defined by DICOM standard).
  - Supported DICOM information objects:
    - Slicer core: CT, MRI, PET, X-ray, some ultrasound images; secondary capture with Slicer scene (MRB) in private tag
    - **Quantitative Reporting extension**: DICOM Segmentation objects, Structured reports
    - **SlicerRT extension**: DICOM RT Structure Set, RT Dose, RT Plan, RT Image
    - **SlicerHeart extension**: 2D/3D/4D ultrasound (GE, Philips, Eigen Artemis, and other)
    - **SlicerDMRI tractography storage**
    - **SlicerDcm2nii diffusion weighted MR**
  - Notes:
    - For a number of dMRI formats we recommend use of the DICOM to NRRD converter before loading the data into Slicer.
    - Image volumes, RT structure sets, dose volumes, etc. can be exported using DICOM module’s export feature.
    - Limited support for writing image volumes in DICOM format is provided by the Create DICOM Series module.
    - Support of writing DICOM Segmentation Objects is provided by the Reporting extension
- **NRRD** (.nrrd, .nhdr): General-purpose 2D/3D/4D file format. Coordinate system: as defined in the file header (usually LPS).
– NRRD sequence (.seq.nrrd): 4D volume

• MetaImage (.mha, .mhd): Coordinate system: LPS (AnatomicalOrientation in the file header is ignored).

• VTK (.vtk): Coordinate system: LPS. Important limitation: image axis directions cannot be stored in this file format.

• Analyze (.hdr, .img, .img.gz): Image orientation is specified ambiguously in this format, therefore its use is strongly discouraged. For brain imaging, use Nifti format instead.

• Nifti (.nii, .nii.gz): File format for brain MRI. Not well suited as a general-purpose 3D image file format (use NRRD format instead).

• Tagged image file format (.tif, .tiff): can read/write single/series of frames

• PNG (.png): can read single/series of frames, can write a single frame

• JPEG (.jpg, .jpeg): can read single/series of frames, can write a single frame

• Windows bitmap (.bmp): can read single/series of frames

• BioRad (.pic)

• Brains2 (.mask)

• GIPL (.gipl, .gipl.gz)

• LSM (.ls)

• Stimulate (.spr)

• MGH-NMR (.mgz)

• MRC Electron Density (.mrc)

• SlicerRT extension
  – Vista cone beam optical scanner volume (.vff)
  – DOSXYZnrc 3D dose (.3ddose)

• SlicerHeart extension: 2D/3D/4D ultrasound (GE, Philips, Eigen Artemis, and other)
  – GE Kretz 3D ultrasound (.vol, .v01)

• RawImageGuess extension
  – RAW volume (.raw): requires manual setting of header parameters
  – Samsung 3D ultrasound (.mvl): requires manual setting of header parameters

• SlicerIGSIO extension:
  – Compressed video (.mkv, .webm)
  – IGSIO sequence metafile (.igs.mha, .igs.mhd, .igs.nrrd, .seq.mha, .seq.mhd, .mha, .mhd, .mkv, .webm): image sequence with metadata, for example for storing surgical navigation and position-tracked ultrasound data

• OpenIGTLink extension:
  – PLUS toolkit configuration file (.plus.xml): configuration file for real-time data acquisition from imaging and tracking devices and various sensors
8.2 Models

Surface or volumetric meshes:

- **VTK Polygonal Data** (.vtk, .vtp): Default coordinate system: LPS. Coordinate system (LPS/RAS) can be specified in header.

- **VTK Unstructured Grid Data** (.vtk, .vtu): Volumetric mesh. Default coordinate system: LPS. Coordinate system (LPS/RAS) can be specified in header.

- **STereoLithography** (.stl): Format most commonly used for 3D printing. Default coordinate system: LPS. Coordinate system (LPS/RAS) can be specified in header.

- **Wavefront OBJ** (.obj): Default coordinate system: LPS. Coordinate system (LPS/RAS) can be specified in header.

- **Stanford Triangle Format** (.ply): Default coordinate system: LPS. Coordinate system (LPS/RAS) can be specified in header.

- **BYU** (.byu, .g; reading only): Coordinate system: LPS.

- **UCD** (.ucd; reading only): Coordinate system: LPS.

- **ITK meta** (.meta; reading only): Coordinate system: LPS.

FreeSurfer extension:
- **Freesurfer surfaces** (.orig, .inflated, .sphere, .white, .smoothwm, .pial; read-only)

8.3 Segmentations

- **Segmentation labelmap representation** (.seg.nrrd, .nrrd, .nii, .nii.gz, .hdr): 3D volume (4D volume if there are overlapping segments) with custom fields specifying segment names, terminology, colors, etc.

- **Segmentation closed surface representation** (.vtm): saved as VTK multiblock data set, contains custom fields specifying segment names, terminology, colors, etc.

- **Labelmap volume** (.nrrd, .nii, .nii.gz, .hdr): segment names can be defined by using a color table. To write segmentation in these formats, segmentation node has to be exported to labelmap volume.

- **Closed surface** (.stl, .obj): Single segment can be read from each file. Segmentation module’s Export to files feature can be used to export directly to these formats.

SlicerOpenAnatomy extension:
- **GL Transmission Format** (.glTF, writing only)

8.4 Transforms

- **ITK HDF transform** (.h5): For linear, b-spline, grid (displacement field), thin-plate spline, and composite transforms. Coordinate system: LPS.

- **ITK TXT transform** (.tfm, .txt): For linear, b-spline, and thin-plate spline, and composite transforms. Coordinate system: LPS.

- **Matlab MAT file** (.mat): For linear and b-spline transforms. Coordinate system: LPS.

- **Displacement field** (.nrrd, .nhdr, .mha, .mhd, .nii, .nii.gz): For storing grid transform as a vector image, each voxel containing displacement vector. Coordinate system: LPS.
8.5 Markups

- **Markups JSON** (.mkp.json): fiducial list, line, curve, closed curve, plane, etc. Default coordinate system: LPS. Coordinate system (LPS/RAS) can be specified in image header.
- **Markups CSV** (.fcsv): fiducial list points legacy file format. Default coordinate system: LPS. Coordinate system (LPS/RAS) can be specified in image header.
- **Annotation CSV** (.acsv): annotation ruler, ROI

8.6 Scenes

- **MRML (Medical Reality Markup Language File)** (.mrml): MRML file is a xml-formatted text file with scene metadata and pointers to externally stored data files. See MRML overview. Coordinate system: RAS.
- **MRB (Medical Reality Bundle)** (.mrb, .zip): MRB is a binary format encapsulating all scene data (bulk data and metadata). Internally it uses zip format. Any .zip file that contains a self-contained data tree including a .mrml file can be opened. Coordinate system: RAS. Note: only .mrb file extension can be chosen for writing, but after that the file can be manually renamed to .zip if you need access to internal data.
- **Data collections in XNAT Catalog format** (.xcat; reading only)
- **Data collections in XNAT Archive format** (.xar; reading only)

8.7 Other

- **Text** (.txt, .xml, .json)
- **Table** (.csv, .tsv)
- **Color table** (.ctbl, .txt)
- **Volume rendering properties** (.vp)
- **Volume rendering shader properties** (.sp)
- **Terminology** (.term.json, .json): dictionary of standard DICOM or other terms
- **Node sequence** (.seq.mrb): sequence of any MRML node (for storage of 4D data)

8.8 What if your data is not supported?

If any of the above listed file formats cannot be loaded then report the issue on the Slicer forum.

If you have a file of binary data and you know the data is uncompressed and you know the way it is laid out in memory, then one way to load it in Slicer is to create a .nhdr file that points to the binary file. **RawImageGuess extension** can be used to explore an unknown data set, determining unknown loading parameters, and generate header file.

You can also ask about support for a particular file format on the Slicer forum. There may be extensions or scripts that can read or write additional formats (any Python package can be installed and used for data import/export).
9.1 Overview

The application settings dialog allows users to customize application behavior.
After starting Slicer, it can be accessed clicking in menu: Edit/Application Settings.

9.2 Panels

9.2.1 General

Application startup script can be used to launch any custom Python code when Slicer application is started.

9.2.2 Modules

Skip loading

Select which type of modules to not load at startup. It is also possible to start slicer by temporarily disabling those modules (not saved in settings) by passing the arguments in the command line.

For example, this command will start Slicer without any CLI loaded:

```
Slicer.exe --disable-cli-modules
```

Prefer Executable CLIs

Use the executable version of a CLI instead of its shared version. CLI modules typically come in 2 forms, as shared (dll)and as executable (exe). By default, if there is a shared version, it is the one loaded by Slicer, ignoring the executable version. Loading a shared runs the module faster but increases the memory consumption. For some
configurations (e.g. Windows 32b), memory is critical. Toggling this option to ON skips the loading of shared CLIs and loads executable version of CLIs instead. If there is no executable for a given CLI, the shared version is used.

**Show hidden modules**

Some modules don’t have a user interface, they are hidden from the module’s list. For debugging purpose, it is possible to force their display.

**Temporary directory**

Directory where modules can store their temporary outputs if needed.

**Additional module paths**

List of directories scanned at startup to load additional modules. Any CLI, Loadable or scripted modules located in these paths will be loaded. Extensions are listed in the list, to remove an extension, use the Extension Manager instead. It is also possible to start Slicer by temporarily adding module paths (not saved in settings) by passing the arguments in the command line.

For example this command will start Slicer trying to load CLIs found in the specified directory:

```
Slicer.exe --additional-module-paths C:\path\to\lib\Slicer-X.Y\cli-modules
```

**Modules**

List of modules loaded, ignored or failed to load in Slicer. An unchecked checkbox indicates that module should not be loaded (ignored) next time Slicer starts. A text color code is used to describe the state of each module:

- Black: module successfully loaded in Slicer
- Gray: module not loaded because it has been ignored (unchecked)
- Red: module failed to load. There are multiple reasons why a module can fail to load.

Look at startup log outputs to have more informations. If a module is not loaded in Slicer (ignored or failed), all dependent modules won’t be loaded. You can verify the dependencies of a module in the tooltip of the module.

You can filter the list of modules by untoggling in the advanced (>>) panel the “To Load”, “To Ignore”, “Loaded”, “Ignored” and “Failed” buttons.

**Home**

Module that is shown when Slicer starts up.

**Favorites**

List of modules that appear in the Favorites toolbar:
To add a module, drag & drop it from the Modules list above. Then use the advanced panel (>>) to reorganize/delete the modules within the toolbar.

### 9.2.3 Appearance

**Style**

The overall theme of Slicer is controlled by the selected Style:

- **Slicer** (default): it sets the style based on theme settings set by the operating system. For example, on Windows if [dark mode](https://blogs.windows.com/windowsexperience/2016/08/08/windows-10-tip-personalize-your-pc-by-enabling-the-dark-theme/) is turned on for apps, then the Dark Slicer style will be used upon launching Slicer. Currently, automatic detection of dark mode is not available on Linux, therefore use needs to manually select Dark Slicer style for a dark color scheme.

- **Light Slicer**: application window background is bright, regardless of operating system settings.

- **Dark Slicer**: application window background is dark, regardless of operating system settings.

### 9.3 Information for Advanced Users

#### 9.3.1 Settings file location

Settings are stored in a *.ini files located in directory like these ones:

- **Windows**: `%USERPROFILE%\AppData\Roaming\NA-MIC\` (typically `C:\Users\<your_user_name>\AppData\Roaming\NA-MIC\`)

- **Linux**: `~/.config/NA-MIC/`

- **Mac**: `~/.config/www.na-mic.org/`

Deleting the *.ini files restores all the settings to default.

There are two types of settings:

**User specific settings**

This file is named `Slicer.ini` and it stores settings applying to all versions of Slicer installed by the current user.

To display the exact location of this settings file, open a terminal and type:

```
./Slicer --settings-path
```

On Windows:

```
Slicer.exe --settings-path | more
```

or enter the following in the Python interactor:

```
slicer.app.slicerUserSettingsFilePath
```
User and revision specific settings

This file is named like Slicer-<REVISION>.ini and it stores settings applying to a specific revision of Slicer installed by the current user.

To display the exact location of this settings file, enter the following in the Python interactor:

```python
slicer.app.slicerRevisionUserSettingsFilePath
```
10.1 Overview

Building Slicer is the process of obtaining a copy of the source code of the project and use tools, such as compilers, project generators and build systems, to create binary libraries and executables—Slicer documentation can be also generated in this process. Users interested on using the Slicer application and its ecosystem of extensions will not typically be interested on building 3D Slicer. On the other hand, users interested on developing Slicer modules or contributing to the development of Slicer, need to have the source code of Slicer and the corresponding generated binaries.

Slicer is based on a superbuild architecture. This means that the in the building process, most of the dependencies of Slicer will be downloaded in local directories (within the Slicer build directory) and will be configured, built and installed locally, before Slicer itself is built. This helps reducing the complexity for developers.

The instructions provided in this document have been tested for Slicer in its latest version and generally will work on versions that are not too far from it in the development process.

10.1.1 Custom builds

Customized editions of Slicer can be generated without changing Slicer source code, just by modifying CMake variables:

- `SlicerApp_APPLICATION_NAME`: Custom application name to be used, instead of default “Slicer”. The name is used in installation package name, window title bar, etc.

- `Slicer_DISCLAIMER_AT_STARTUP`: String that is displayed to the user after first startup of Slicer after installation (disclaimer, welcome message, etc).

- `Slicer_DEFAULT_HOME_MODULE`: Module name that is activated automatically on application start.

- `Slicer_DEFAULT_FAVORITE_MODULES`: Modules that will be added to the toolbar by default for easy access. List contains module names, separated by space character.

- `Slicer_CLIMODULES_DISABLED`: Built-in CLI modules that will be removed from the application. List contains module names, separated by semicolon character.
• **Slicer_QTLOADABLEMODULES_DISABLED**: Built-in Qt loadable modules that will be removed from the application. List contains module names, separated by semicolon character.

• **Slicer_QTSCRIPTEDMODULES_DISABLED**: Built-in scripted loadable modules that will be removed from the application. List contains module names, separated by semicolon character.

• **Slicer_USE_PYTHONQT_WITH_OPENSSL**: enable/disable building the application with SSL support (ON/OFF)

• **Slicer_USE_SimpleITK**: enable/disable SimpleITK support (ON/OFF)

• **Slicer_BUILD_SimpleFilters**: enable/disable building SimpleFilters. Requires SimpleITK. (ON/OFF)

• **Slicer_USE_PYTHONQT_WITH_TCL**: TCL support (ON/OFF)

• **Slicer_EXTENSION_SOURCE_DIRS**: Defines additional extensions that will be included in the application package as built-in modules. Full paths of extension source directories has to be specified, separated by semicolons.

More customization is available by using SlicerCustomAppTemplate project maintained by Kitware.

### 10.1.2 Runtime environment variables

The following environment variables can be set in the environment where the application is started to fine-tune its behavior:

• **PYTHONNOUSERSITE**: if it is set to 1 then import of user site packages is disabled.

• **QT_ENABLE_HIGHDPI_SCALING**: see Qt documentation

• **QT_SCALE_FACTOR_ROUNDING_POLICY**: see Qt documentation

• **QTWEBENGINE_REMOTE_DEBUGGING**: port number for Qt webengine remote debugger. Default value is 1337.

• **SLICER_OPENGL_PROFILE**: Requested OpenGL profile. Valid values are no (no profile), core (core profile), and compatibility (compatibility profile). Default value is compatibility on Windows systems.

• **SLICER_BACKGROUND_THREAD_PRIORITY**: Set priority for background processing tasks. On Linux, it may affect the entire process priority. An integer value is expected, default = 20 on Linux and macOS, and -1 on Windows.

### 10.2 Windows

#### 10.2.1 Install prerequisites

• CMake **>= 3.15.1**

• Git **>= 1.7.10**
  
  – Note CMake must be able to find git.exe and patch.exe. If git is installed in the default location then they may be found there, but if they are not found then either add the folder that contains them to PATH environment variable; or set GIT_EXECUTABLE and Patch_EXECUTABLE as environment variables or as CMake variables at configure time.

• NSIS (optional): Needed if packaging Slicer. Make sure you install the language packs.

• Qt5: Download Qt universal installer and install Qt 5.15 components: MSVC2019 64-bit, Qt Script, Qt WebEngine. Installing Sources and Qt Debug Information Files are recommended for debugging (they allow stepping into Qt files with the debugger in debug-mode builds).
3D Slicer Documentation

• **Visual Studio**
  – any edition can be used (including the free Community edition)
  – when configuring the installer, enable Desktop development with C++ and in installation details, check MSVC v142 - VS2019 C++ x64... (Visual Studio 2019 v142 toolset with 64-bit support) - in some distributions, this option is not enabled by default

Other versions:
• Visual Studio 2017 (v141) toolset is not tested anymore but probably still works. Qt-5.15 requires v142 redistributables, so either these extra DLL files need to be added to the installation package or each user may need to install “Microsoft Visual C++ Redistributable” package.
• Visual Studio 2015 (v140) toolset is not tested anymore and probably does not work. Requires Qt 5.10.x to build due to QtWebEngine.
• Cygwin: not tested and not recommended. Building with cygwin gcc not supported, but the cygwin shell environment can be used to run git, svn, etc.

**10.2.2 Set up source and build folders**

• Create source folder. This folder will be referred to as `<Slicer_SOURCE>` in the followings. Recommended path: C:\D\S4
  – Due to maximum path length limitations during build the build process, source and build folders must be located in a folder with very short (couple of characters) total path length.
  – While it is not enforced, we strongly recommend you to avoid the use of spaces for both the source directory and the build directory.

• Create build folder. This folder will be referred to as `<Slicer_BUILD>` in the followings. Recommended path: C:\D\S4R for release-mode build, C:\D\S4D for debug-mode build.
  – You cannot use the same build tree for both release or debug mode builds. If both build types are needed, then the same source directory can be used, but a separate build directory must be created and configured for each build type.

• Download source code into Slicer source folder from GitHub: https://github.com/Slicer/Slicer.git
  – The following command can be executed in Slicer source folder to achieve this: git clone https://github.com/Slicer/Slicer.git .

• Configure the repository for developers (optional): Needed if changes need to be contributed to Slicer repository.
  – Right-click on `<Slicer_SOURCE>/Utilities` folder in Windows Explorer and select Git bash here
  – Execute this command in the terminal (and answer all questions): ./SetupForDevelopment.sh
  – Note: more information about how to use git in Slicer can be found on this page

**10.2.3 Configure and build Slicer**

Using command-line (recommended)

Specify source, build, and Qt location and compiler version and start the build using the following commands (these can be put into a .bat file so that they can be executed again easily), assuming default folder locations:

Release mode:
mkdir C:\D\S4R
"C:\Program Files\CMake\bin\cmake.exe" --G "Visual Studio 16 2019 Win64" -DQt5_
--DIR:PATH=C:\Qt\5.15.0\msvc2019_64\lib\cmake\Qt5 C:\D\S4
"C:\Program Files\CMake\bin\cmake.exe" --build . --config Release

Debug mode:

mkdir C:\D\S4D
"C:\Program Files\CMake\bin\cmake.exe" --G "Visual Studio 16 2019 Win64" -DQt5_
--DIR:PATH=C:\Qt\5.15.0\msvc2019_64\lib\cmake\Qt5 C:\D\S4
"C:\Program Files\CMake\bin\cmake.exe" --build . --config Debug

Using graphical user interface (alternative solution)

- Run CMake (cmake-gui) from the Windows Start menu
- Set Where is the source code to <Slicer_SOURCE> location
- Set Where to build the binaries to <Slicer_BUILD> location. Do not configure yet!
- Add Qt5_DIR variable pointing to Qt5 folder: click Add entry button, set Name to Qt5_DIR, set Type to PATH, and set Value to the Qt5 folder, such as C:\Qt\5.15.0\msvc2019_64\lib\cmake\Qt5.
- Click Configure
- Select your compiler: Visual Studio 16 2019, and click Finish
- Click Generate and wait for project generation to finish (may take a few minutes)
- Click Open Project
- If building in release mode:
  - Open the top-level Slicer.sln file in the build directory in Visual Studio
  - Set active configuration to Release. Visual Studio will select Debug build configuration by default when you first open the solution in the Visual Studio GUI. If you build Slicer in release mode and accidentally forget to switch the build configuration to Release then the build will fail. Note: you can avoid this manual configuration mode selection by setting CMAKE_CONFIGURATION_TYPES to Release in cmake-gui.
- Build the ALL_BUILD project

### 10.2.4 Run Slicer

Run `<Slicer_BUILD>/Slicer-build/Slicer.exe` application.

Note: Slicer.exe is a "launcher", which sets environment variables and launches the real executable: `<Slicer_BUILD>/Slicer-build/bin\Release\SlicerApp-real.exe` (use Debug instead of Release for debug-mode builds).

### 10.2.5 Test Slicer

- Start Visual Studio with the launcher:

  Slicer.exe --VisualStudioProject
• Select build configuration. Usually Release or Debug.

• In the “Solution Explorer”, right click on RUN_TESTS project (in the CMakePredefinedTargets folder) and then select Build.

10.2.6 Package Slicer (create installer package)

• Start Visual Studio with the launcher:

```bash
Slicer.exe --VisualStudioProject
```

• Select Release build configuration.

• In the “Solution Explorer”, right click on PACKAGE project (in the CMakePredefinedTargets folder) and then select Build.

10.2.7 Debug Slicer

1. To run Slicer, the launcher needs to set certain environment variables. The easiest is to use the launcher to set these and start Visual Studio in this environment. All these can be accomplished by running the following command in `<Slicer_BUILD>/c:\D\S4R\Slicer-build` folder:

```bash
Slicer.exe --VisualStudioProject
```

Notes:

• If you just want to start VisualStudio with the launcher (and then load project file manually), run: Slicer.exe --VisualStudio

• To debug an extension that builds third-party DLLs, also specify --launcher-additional-settings option.

• While you can launch debugger using Slicer’s solution file, it is usually more convenient to load your extension’s solution file (because your extension solution is smaller and most likely you want to have that solution open anyway for making changes in your code). For example, you can launch Visual Studio to debug your extension like this:

```bash
\S4D\Slicer-build\Slicer.exe --VisualStudio --launcher-no-splash --launcher-
--additional-settings ./SlicerRT_D/inner-build/AdditionalLauncherSettings.ini c:\d\_
 Extensions\SlicerRT_D\inner-build\SlicerRT.sln
```

1. In Solution Explorer window in Visual Studio, expand App-Slicer, right-click on SlicerApp (NOT qSlicerApp) and select “Set as Startup Project”

To debug in an extension’s solution: set ALL_BUILD project as startup project and in project settings, set Debugging / Command field to the full path of SlicerApp-real.exe - something like .../Slicer-build/bin/Debug/SlicerApp-real.exe

1. Run Slicer in debug mode by Start Debugging command (in Debug menu).

Note that because CMake re-creates the solution file from within the build process, Visual Studio will sometimes need to stop and reload the project, requiring manual button pressing on your part (just press Yes or OK whenever you are asked). To avoid this, you can use a script to complete the build process and then re-start the Visual Studio.

For more debugging tips and tricks, check out this page.
Debug a test

Once VisualStudio is open with the Slicer environment loaded, it is possible to run and debug tests. To run all tests, build the RUN_TESTS project.

- To debug a test, find its project:
  - Libs/MRML/Core tests are in the MRMLCoreCxxTests project
  - CLI module tests are in <CLI_NAME>Test project (e.g. ThresholdScalarVolumeTest)
  - Loadable module tests are in qSlicer<LOADABLE_NAME>CxxTests project (e.g. qSlicerVolumeRenderingCxxTests)
  - Module logic tests are in <MODULE_NAME>LogicCxxTests project (e.g. VolumeRenderingLogicCxxTests)
  - Module widgets tests are in <MODULE_NAME>WidgetsCxxTests project (e.g. VolumesWidgetsCxxTests)

- Go to the project debugging properties (right click -> Properties, then Configuration Properties/Debugging)

- In Command Arguments, type the name of the test (e.g. vtkMRMLSceneImportTest for project MRMLCoreCxxTests)

- If the test takes argument(s), enter the argument(s) after the test name in Command Arguments (e.g. vtkMRMLSceneImportTest C:\Path\To\Slicer4\Libs\MRML\Core\Testing\vol_and_cube.mrml)

  - You can see what arguments are passed by the dashboards by looking at the test details in CDash.
  - Most VTK and Qt tests support the -I argument, it allows the test to be run in “interactive” mode. It doesn’t exit at the end of the test.

- Make the project Set As Startup Project

- Start Debugging (F5)

Debugging Python scripts

See Python scripting page for detailed instructions.

10.2.8 Common errors

See list of issues common to all operating systems on Common errors page.

10.3 Mac OS

10.3.1 Prerequisites

The prerequisites listed below are required to be able to configure/build/package/test Slicer.

- XCode command line tools must be installed:

  xcode-select --install

- El Capitan is what most developers use.

- CMake 3.12.2 is recommended. Check the minimum required CMake version here
• Large File Storage for git is required. (brew install git-lfs)

• Qt 5: tested and recommended.
  – For building Slicer: download and execute qt-unified-mac-x64-online.dmg, install Qt 5.10, make sure to select qtsheet and qtwebengine components.
  – For packaging and redistributing Slicer: build Qt using qt-easy-build

• Setting CMAKE_OSX_DEPLOYMENT_TARGET CMake variable specifies the minimum macOS version a generated installer may target. So it should be equal to or less than the version of SDK you are building on. Note that the SDK version is set using CMAKE_OSX_SYSROOT CMake variable automatically initialized during CMake configuration.

Mac OSX 10.9.4 (Mavericks)

• Make sure to install this update: http://support.apple.com/kb/DL1754

• Use CMake 3.12.2 - it is known to be working and is supported (if you want to use CMake already installed on your system, 2.8.12.2 is known to work on Mac OS X 10.9.5)

Mac OSX >= 10.5 (Leopard)

• CMake >= 2.8.9
  – For Mac Os X >= 10.8 (Mountain Lion) and/or recent XCode >= 4.5.X - CMake >= 2.8.11 is required. See http://www.cmake.org/files/v2.8/cmake-2.8.11-Darwin64-universal.tar.gz

```shell
$ curl -O http://www.cmake.org/files/v2.8/cmake-2.8.11-Darwin64-universal.tar.gz
$ tar -xzvf cmake-2.8.11-Darwin64-universal.tar.gz --strip-components=1

$ CMake\ 2.8-11.app/Contents/bin/cmake --version
  cmake version 2.8.11
```

• Git >= 1.7.10

• Svn >= 1.7

• XCode (for the SDK libs)
  – After installing XCode, install XCode command line developer tools:

```shell
xcode-select --install
```

• XQuartz - For Mac Os X >= 10.8 (Mountain Lion) install XQuartz (http://xquartz.macosforge.org) to get X11 (no longer a default in OS X).

• Qt 4 >= 4.8.5. We recommend you install the following two packages:
  – Download and install [http://download.qt-project.org/official_releases/qt/4.8.6/qt-opensource-mac-4.8.6-1.dmg qt-opensource-mac-4.8.6-1.dmg]
  – Download and install [http://download.qt-project.org/official_releases/qt/4.8.6/qt-opensource-mac-4.8.6-1-debug-libs.dmg qt-opensource-mac-4.8.6-1-debug-libs.dmg]

10.3. Mac OS
Mac OSX 10.11 (El Capitan)

XCode up to version 7 is known to work for Slicer compilation. XCode 8 breaks things on several levels for now. Remember to install XCode command line tools (see above) if a reinstall for XCode is needed.

The standard Qt4 installers fail on this version and the 4.8.6 source code won’t build. But as described on the slicer-devel mailing list it is possible to install the homebrew version of qt4 which patches it to work on El Capitan (see below).

- Install the OS, Xcode, and XQuartz (see MacOSX 10.10 above)
- Install Qt4 by running the following two commands:

```bash
brew install qt4
xcode-select --install
```

- TCL does not build correctly on El Capitan as of 2015-12-03, so when building Slicer turn Slicer_USE_PYTHONQT_WITH_TCL off.

Mac OSX 10.12 (Sierra)

Similar to 10.11 (El Capitan), there are new issues with Qt4 (caused by Phonon). The GitHub user Cartr offered a patch to the brew team, and even though it was not integrated (the homebrew team decided to stop patching their recipe to encourage people to use Qt5), he turned his formula into a tap that can be installed (see below).

Follow instructions for 10.11 (Installing Xcode, XQuartz, run without TCL) but install Qt4 like shown below instead:

```bash
brew install cartr/qt4/qt
xcode-select --install
```

Confirmed with Xcode:

- Version 8.1 (8B62) and cmake version 3.4.20151021-g8fbc8e
- Version 8.3.3 and cmake 3.5.2

Mac OSX 10.14 (Mojave)

(Associated discussion topic is https://discourse.slicer.org/t/building-on-mac-10-14-mojave/4554/21)

- Install Qt 5.11.2 using Qt Online Installer for macOS
- Install XCode:

```bash
xcode-select --install
```

- Build qt from homebrew

```bash
brew install qt
ccmake -DCMAKE_OSX_DEPLOYMENT_TARGET=10.14 -DQt5_DIR=/usr/local//Cellar/qt/5.13.2/lib/ →cmake/Qt5 ~/slicer/latest/Slicer
```

- Explicitly set the SDK when running make

```bash
SDKROOT=/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/ →Developer/SDKs/MacOSX10.15.sdk make -j20
```

Note if you have build errors in dcmtk related to iconv symbols, you may need to uninstall the icu4c and dcmtk homebrew packages during the build process. See here and here.
10.3.2 Checkout Slicer source files

Notes:

• While it is not enforced, we strongly recommend you to avoid the use of spaces for both the source directory and the build directory.

• Due to maximum path length limitations during build the build process, source and build folders must be located in a folder with very short total path length. This is especially critical on Windows and MacOS. For example, /sq5 has been confirmed to work on MacOS.

Check out the code using git:

• Clone the github repository

  git clone git://github.com/Slicer/Slicer.git

The ‘Slicer’ directory is automatically created after cloning Slicer.

• Setup the development environment:

  ./.Utilities/SetupForDevelopment.sh

10.3.3 Configure and generate Slicer solution files

• Configure using the following commands. By default CMAKE_BUILD_TYPE is set to Debug (replace /path/to/QtSDK with the real path on your machine where QtSDK is located):

  mkdir Slicer-SuperBuild-Debug
  cd Slicer-SuperBuild-Debug
  cmake -DCMAKE_BUILD_TYPE:STRING=Debug -DQt5_DIR:PATH=/path/to/Qt5.15.0/5.15.0/gcc_64/lib/cmake/Qt5 ../Slicer

• If using Qt from the system, do not forget to add the following CMake variable to your configuration command line: -DSlicer_USE_SYSTEM_QT:BOOL=ON

• Remarks:
  – Instead of cmake, you can use ccmake or cmake-gui to visually inspect and edit configure options.
  – Using top-level directory name like Slicer-SuperBuild-Release or Slicer-SuperBuild-Debug is recommended.
  – Step-by-step debug instructions
  – Additional configuration options to customize the application are described here.

• On Mac OS X 10.9 (Mavericks) / 10.10 (Yosemite) / 10.11 (El Capitan), also set the following variables (see discussion):
  – Slicer_USE_PYTHONQT_WITH_TCL -> OFF
  – CMAKE_OSX_DEPLOYMENT_TARGET -> 10.9 or 10.10 or 10.11

General information

Two projects are generated by either cmake, ccmake or cmake-gui. One of them is in the top-level bin directory Slicer-SuperBuild and the other one is in the subdirectory Slicer-build.
• Slicer-SuperBuild manages all the external dependencies of Slicer (VTK, ITK, Python, ...). To build Slicer for the first time, run make (or build the solution file in Visual Studio) in Slicer-SuperBuild, which will update and build the external libraries and if successful will then build the subproject Slicer-build.

• Slicer-SuperBuild/Slicer-build is the “traditional” build directory of Slicer. After local changes in Slicer (or after an svn update on the source directory of Slicer), only running make (or building the solution file in Visual Studio) in Slicer-SuperBuild/Slicer-build is necessary (the external libraries are considered built and up to date).

Warning: An significant amount of disk space is required to compile Slicer in Debug mode (>10GB)

Warning: Some firewalls will block the git protocol. See more information and solution here.

10.3.4 Build Slicer

After configuration, start the build process in the Slicer-SuperBuild directory

• Start a terminal and type the following (you can replace 4 by the number of processor cores in the computer):

```bash
cd ~/Projects/Slicer-SuperBuild
make -j4
```

In case of file download hash mismatch error, you need to acquire the latest wget, and build cmake with OpenSSL turned on. For more information, see here and here.

When using the -j option, the build will continue past the source of the first error. If the build fails and you don’t see what failed, rebuild without the -j option. Or, to speed up this process build first with the -j and -k options and then run plain make. The -k option will make the build keep going so that any code that can be compiled independent of the error will be completed and the second make will reach the error condition more efficiently.

10.3.5 Run Slicer

Start a terminal and type the following:

```bash
Slicer-SuperBuild/Slicer-build/Slicer
```

10.3.6 Test Slicer

After building, run the tests in the Slicer-SuperBuild/Slicer-build directory.

Start a terminal and type the following (you can replace 4 by the number of processor cores in the computer):

```bash
cd ~/Projects/Slicer-SuperBuild/Slicer-build
cmake -j4
```

10.3.7 Package Slicer

Start a terminal and type the following:

```bash
cd ~/Projects/Slicer-SuperBuild
cd Slicer-build
make package
```
10.3.8 Common errors

See list of issues common to all operating systems on Common errors page.

CMake complains during configuration

CMake may not directly show what’s wrong; try to look for log files of the form BUILD/CMakeFiles/*.log (where BUILD is your build directory) to glean further information.

error: ‘class QList’ has no member named ‘reserve’

You have multiple Qt versions installed on your machine. Try removing the Qt version installed on the system.

libarchive.so: undefined reference to ‘SHA256_Update’

Solution:

cd Slicer-SuperBuild
rm -rf LibArchive-*
make -j4

Details:

• http://na-mic.org/Mantis/view.php?id=1616

• http://viewvc.slicer.org/viewvc.cgi/Slicer4?view=revision&revision=18923

• http://viewvc.slicer.org/viewvc.cgi/Slicer4?view=revision&revision=18969

ld: framework not found QtWebKit
See Slicer issue 2860.

Workaround:
- See http://public.kitware.com/Bug/view.php?id=13765#c31824

Solution:
- Use a recent CMake. It includes patch cc6763a. Note also that the coming version of CMake 2.8.11 will work as expected.

**On MacOSX 10.8, CMake hangs forever**

Issue: http://www.na-mic.org/Bug/view.php?id=2957


Details:
CMake folks (Thanks Brad King :) ) fixed an issue that was preventing the most recent nightly from being used to successfully compile Slicer. The fix has been tested and is known to work. Tomorrow nightly build of CMake and by extension the coming release of CMake 2.8.11 will work.

If you are curious about the details, check the commit message: http://cmake.org/gitweb?p=cmake.git;a=commitdiff;h=1df09e57
The associated CMake test have also been updated: http://cmake.org/gitweb?p=cmake.git;a=commitdiff;h=bff3d9ce

**On MacOSX 10.8/10.9 with XQuartz, ‘X11/Xlib.h’ file not found**

Issue: http://www.na-mic.org/Bug/view.php?id=3405

Workaround: Build with -DSlicer_USE_PYTHONQT_WITH_TCL:BOOL=OFF
Tcl is required only for EMSegment module.


‘QSslSocket’ : is not a class or namespace name

This error message occurs if Slicer is configured to use SSL but Qt is built without SSL support.
Either set Slicer_USE_PYTHONQT_WITH_OPENSSL to OFF when configuring Slicer build in CMake, or build Qt with SSL support.

**error: Missing Qt module QTWEBKIT**

```
CMake Error at CMake/SlicerBlockFindQtAndCheckVersion.cmake:43 (message):
  error: Missing Qt module QTWEBKIT
Call Stack (most recent call first):
  CMake/SlicerBlockFindQtAndCheckVersion.cmake:88 (__SlicerBlockFindQtAndCheckVersion__find_qt)
  CMakeLists.txt:607 (include)
```

Solution:
```
sudo apt-get -y install libqtwebkit-dev
```
**error when starting Slicer: NameError: name 'getSlicerRCFileName' is not defined**

```
./Slicer
Traceback (most recent call last):
  File "<string>", line 7, in <module>
    File "'/home/fbudin/Devel/Slicer-SuperBuild-Release/Slicer-build/bin/Python/slicer/_slicerqt.py", line 6, in <module>
      import vtk
    File "'/home/fbudin/Devel/Slicer-SuperBuild-Release/VTKv7-build/Wrapping/Python/vtk/__init__.py", line 41, in <module>
      from .vtkCommonKit import *
    File "'/home/fbudin/Devel/Slicer-SuperBuild-Release/VTKv7-build/Wrapping/Python/vtk/vtkCommonKit.py", line 9, in <module>
      from vtkCommonKitPython import *
```

Solution and details here

**macOS: error while configuring PCRE: “cannot run C compiled program”**

If the XCode command line tools are not properly set up on OS X, PCRE could fail to build in the Superbuild process with the errors like below:

```
configure: error: in `'/Users/fedorov/local/Slicer4-Debug/PCRE-build'
configure: error: cannot run C compiled programs.
```

To install XCode command line tools, use the following command from the terminal:

```
xcode-select --install
```

**macOS: dyld: malformed mach-o: load commands size (...) > 32768**

Path of source or build folder is too long. For example building Slicer in `/User/somebody/projects/something/dev/slicer/slicer-qt5-rel` may fail with malformed mach-o error, while it succeeds in `/sq5` folder. To resolve this error, move source and binary files into a folder with shorter full path and restart the build from scratch (the build tree is not relocatable).

### 10.4 GNU/Linux Systems

The instructions to build Slicer for GNU/Linux systems are slightly different depending on the linux distribution and the specific configuration of the system. In the following sections you can find instructions that will work for some of the most common linux distributions in their standard configuration. If you are using a different distribution you
can use these instructions as guidelines to adapt the process to your system. You can also ask questions related to the building process in the Slicer forum.

10.4.1 Pre-requisites

First, you need to install the tools that will be used for fetching the source code of slicer, generating the project files and build the project.

- Git and Subversion for fetching the code and version control.
- GNU Compiler Collection (GCC) for code compilation.
- CMake for configuration/generation of the project.
  - (Optional) CMake curses gui to configure the project from the command line.
  - (Optional) CMake Qt gui to configure the project through a GUI.
- GNU Make
- GNU Patch

In addition, Slicer requires a set of support libraries that are not included as part of the superbuild:

- Qt5 with the following components:
  - Multimedia
  - UiTools
  - XMLPatterns
  - SVG
  - WebEngine
  - Script
  - X11Extras
  - Private
- libXt

**Debian 10 Stable (Buster)**

Install the development tools and the support libraries:

```bash
sudo apt install git subversion build-essential cmake cmake-curses-gui cmake-qt-gui qt5-default qt5multimedia-dev qttools5-dev libqt5xmlpatterns5-dev libqt5svg5-dev libqt5webengine5-dev qtscript5-dev lqtbase5-private-dev libqt5x11extras5-dev libxt-dev libssl-dev
```

**Debian Testing (Bullseye) and Debian 9**

*This option is not suggested since it does not work with standard packages. Debian 9 Qt 5.7 packages will not work with current Slicer 4.11. Checked 2020-08-19. May be possible to build from source or install other packages. In addition, for Debian 9 you also need to build cmake from source as described here or otherwise get a newer version than is supplied by the distribution.*

Install the development tools and the support libraries:
### Ubuntu 20.04 (Focal Fossa)

Install the development tools and the support libraries:

```bash
sudo apt install git subversion build-essential cmake cmake-curses-gui cmake-qt-gui
qt5-default qtmultimedia5-dev qttools5-dev libqt5xmlpatterns5-dev libqt5svg5-dev
qtwebengine5-dev qtscript5-dev qtbases5-private-dev libqt5x11extras5-dev libxt-dev
libssl-dev
```

### ArchLinux

Install the development tools and the support libraries:

```bash
sudo pacman -S git make patch subversion gcc cmake qt5-base qt5-multimedia qt5-tools
qt5-xmlpatterns qt5-svg qt5-webengine qt5-script qt5-x11extras libxt
```

### 10.4.2 Checkout Slicer source files

The recommended way to obtain the source code of SLlicer is cloning the repository using `git`:

```bash
git clone git://github.com/Slicer/Slicer.git
```

This will create a `Slicer` directory containing the source code of Slicer. Hereafter we will call this directory the `source directory`.

After obtaining the source code, we need to set up the development environment:

```bash
cd Slicer
./Utilities/SetupForDevelopment.sh
```

[comment]: <> (TODO: Link to the readthedocs equivalent of https://www.slicer.org/wiki/Documentation/Nightly/Developers/DevelopmentWithGit)

### 10.4.3 Configure and generate the Slicer build project files

Slicer is highly configurable and multi-platform. To support this, Slicer needs a configuration of the build parameters before the build process takes place. In this configuration stage, it is possible to adjust variables that change the nature and behaviour of its components. For instance, the type of build (Debug or Release mode), whether to use system-installed libraries, let the build process fetch and compile own libraries, or enable/disable some of the software components and functionalities of Slicer.

To obtain a default configuration of the Slicer build project use `cmake`:

```bash
mkdir Slicer-SuperBuild-Debug
cd Slicer-SuperBuild-Debug
cmake ../Slicer
```

It is possible to change variables with `cmake`. In the following example we change the built type (Debug as default) to Release:

```bash
```

10.4. GNU/Linux Systems 97
10.4.4 Build Slicer

Once the Slicer build project files have been generated, the Slicer project can be build:

```bash
cd Slicer-SuperBuild-Debug
make
```

10.4.5 Run Slicer

After the building process has successfully completed, the executable file to run slicer will be located in ./Slicer-build/Slicer

10.4.6 Test Slicer

After building, run the tests in the Slicer-SuperBuild/Slicer-build directory. Start a terminal and type the following (you can replace 4 by the number of processor cores in the computer):

```bash
cd Slicer-SuperBuild-Debug/Slicer-build
ctest -j4
```

10.4.7 Package Slicer

Start a terminal and type the following:

```bash
cd Slicer-SuperBuild-Debug/Slicer-build
make package
```

10.4.8 Common errors

See list of issues common to all operating systems on Common errors page.

10.5 Common errors

10.5.1 Firewall is blocking git protocol

Some firewalls will block the git protocol. A possible workaround is to configure Slicer by disabling the option `Slicer_USE_GIT_PROTOCOL`. Then the http protocol will be used instead. Consider also reading https://github.com/commontk/CTK/issues/33.
CHAPTER 11

Slicer API

11.1 Python

11.1.1 mrml module

11.1.2 saferef module

11.1.3 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)

Create 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 execuction (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)
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, parameter-Value) 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.testing module

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

slicer.util module

slicer.util.DATA_STORE_URL = 'https://github.com/Slicer/SlicerDataStore/releases/download/'
Base URL for downloading data from Slicer Data Store. Data store contains atlases, registration case library images, and various sample data sets.
Datasets can be downloaded using URL of the form DATA_STORE_URL + "SHA256/" + sha256ofDataSet

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.

slicer.util.TESTING_DATA_URL = 'https://github.com/Slicer/SlicerTestingData/releases/download/'
Base URL for downloading testing data.
Datasets can be downloaded using URL of the form TESTING_DATA_URL + "SHA256/" + sha256ofDataSet

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.addParameterEditWidgetConnections (parameterEditWidgets, updateParameterNodeFromGUI)
Add connections to get notification of a widget change.
The function is useful for calling updateParameterNodeFromGUI method in scripted module widgets.
Note: Only a few widget classes are supported now. More will be added later. Report any missing classes at discourse.slicer.org.

Example:

```python
class SurfaceToolboxWidget(ScriptedLoadableModuleWidget, VTKObservationMixin):
    ...
    setup(self):
        ...
        self.parameterEditWidgets = [
            (self.ui.inputModelSelector, "inputModel"),
            (self.ui.outputModelSelector, "outputModel"),
            (self.ui.decimationButton, "decimation"),
        ]
        slicer.util.addParameterEditWidgetConnections(self.parameterEditWidgets, self.
        →updateParameterNodeFromGUI)

    def updateGUIFromParameterNode(self, caller=None, event=None):
        if self._parameterNode is None or self._updatingGUIFromParameterNode:
            return
        self._updatingGUIFromParameterNode = True
        slicer.util.updateParameterEditWidgetsFromNode(self.parameterEditWidgets,
        →self._parameterNode)
        self._updatingGUIFromParameterNode = False

    def updateParameterNodeFromGUI(self, caller=None, event=None):
        if self._parameterNode is None or self._updatingGUIFromParameterNode:
            return
        wasModified = self._parameterNode.StartModify()  # Modify all properties in a
        →single batch
        slicer.util.updateNodeFromParameterEditWidgets(self.parameterEditWidgets,
        →self._parameterNode)
        self._parameterNode.EndModify(wasModified)
```

`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. 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.

**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
- `nodeClassName` – type of created volume, default: 
vtkMRMLScalarVolumeNode. Use vtkMRMLLabelMapVolumeNode for
  labelmap volume, vtkMRMLVectorVolumeNode for vector volume.

**Returns** created new volume node

Example:

```python
# create zero-filled volume
import numpy as np
volumeNode = slicer.util.addVolumeFromArray(np.zeros((30, 40, 50)))
```
# create labelmap volume filled with voxel value of 120

```python
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.arrayFromModelPointDataModified(modelNode, arrayName)

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

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()`.
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), ... i(0,n(0)), ..., n(j), i(j,0), ... i(j,n(j)), ...] 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:

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

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

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:

segmentationNode.GetSegmentation().GetSegment(segmentID).Modified()
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 have 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 (tableNode, 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 have 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 numpy.copy().
2. Perform any computations using the copied array.
3. Write results back to the image data using updateVolumeFromArray().

slicer.util.arrayFromVolumeModified (volumeNode)
Indicate that modification of a numpy array returned by arrayFromVolume() has been completed.

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:
uiWidget = slicer.util.loadUI(myUiFilePath)
self.ui = slicer.util.childWidgetVariables(uiWidget)
slicer.util.setInputSelector(self.ui, slicer.mrmlScene)
slicer.util.setOutputSelector(self.ui, slicer.mrmlScene)

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

- **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.computeChecksum(algo, filePath)

Compute digest of filePath using algo.

Supported hashing algorithms are SHA256, SHA512, and MD5.

It internally reads the file by chunk of 8192 bytes.

**Raises**

- ValueError – if algo is unknown.
- IOError – if 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:
# 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.dataframeFromMarkups(markupsNode)
Convert table node content to pandas dataframe. Table content is copied. Therefore, changes in table node do not affect the dataframe, and dataframe changes do not affect the original table node.

slicer.util.dataframeFromTable(tableNode)
Convert table node content to pandas dataframe. Table content is copied. Therefore, changes in table node do not affect the dataframe, and dataframe changes do not affect the original table node.

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:cc211f0dfd9a05ca3841ce1141b292898b2dd2d3f08286affadf823a7e58df93.

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:cc211f0dfd9a05ca3841ce1141b292898b2dd2d3f08286affadf823a7e58df93.

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.

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.

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.
    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 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.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.loadMarkups(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.loadMarkupsClosedCurve(filename)`
Deprecated. Use loadMarkups function instead.

`slicer.util.loadMarkupsCurve(filename)`
Deprecated. Use loadMarkups function instead.

`slicer.util.loadMarkupsFiducialList(filename, returnNode=False)`
Deprecated. Use loadMarkups function instead.

`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.loadSequence(filename, properties={})
Load sequence (4D data set) 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
  - show: display volume in slice viewers after loading is completed
  - colorNodeID: color node to set in the proxy nodes’s display node

Returns loaded sequence node.

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.loadTable(filename)
Load table node from file. :param filename: full path of the file to load. :return: loaded table node

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.

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

`slicer.util.longPath(path)`
Make long paths work on Windows, where the maximum path length is 260 characters. For example, the files in the DICOM database may have paths longer than this limit. Accessing these can be made safe by prefixing it with the UNC prefix (\?).

**Parameters** path (string) – Path to be made safe if too long

**Return** string Safe path

`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:

```python
slicer.util.messageBox("Some message", dontShowAgainSettingsKey = "MainWindow/DontShowSomeMessage")
```

`slicer.util.moduleNames()`
`slicer.util.modulePath(moduleName)`
`slicer.util.moduleSelector()`
`slicer.util.openAddColorTableDialog()`
`slicer.util.openAddDataDialog()`
`slicer.util.openAddFiberBundleDialog()`
`slicer.util.openAddFiducialDialog()`
`slicer.util.openAddMarkupsDialog()`
`slicer.util.openAddModelDialog()`
`slicer.util.openAddScalarOverlayDialog()`
`slicer.util.openAddSegmentationDialog()`
`slicer.util.openAddShaderPropertyDialog()`
slicer.util.openAddTransformDialog()
slicer.util.openAddVolumeDialog()
slicer.util.openSaveDataDialog()
slicer.util.pip_install(requirements)
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 to use this method call instead of a plain pip install. 

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.pip_uninstall(requirements)
Uninstall python packages. Currently, the method simply calls python -m pip uninstall but in the future further checks, optimizations, user confirmation may be implemented, therefore it is recommended to use this method call instead of a plain pip uninstall. 

Example: calling from Slicer GUI
```
pip_uninstall("tensorflow keras scikit-learn ipywidgets")
```

Example: calling from PythonSlicer console
```
from slicer.util import pip_uninstall
pip_uninstall("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 = {}

# 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(verbosetrue)
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.removeParameterEditWidgetConnections(parameterEditWidgets, updateParameterNodeFromGUI)

Remove connections created by addParameterEditWidgetConnections().
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*)’.

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.setDataProbeVisible(visible)
    Show/hide Data probe at the bottom 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(backend='keep-current', foreground='keep-current', label='keep-current', foregroundOpacity=None, labelOpacity=None, fit=False, rotateToVolumePlane=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
• **rotateToVolumePlane** – rotate views to closest axis of the selected background, foreground, or label volume

• **fit** – fit slice views to their content (position&zoom to show all visible layers)

```python
slicer.util.setStatusBarVisible(visible)
Show/hide status bar
```

```python
slicer.util.setToolbarsVisible(visible, ignore=None)
Show/hide all existing toolbars, except those listed in ignore list.
```

```python
slicer.util.setViewControllersVisible(visible)
Show/hide view controller toolbar at the top of slice and 3D views
```

```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.startQtDesigner(args=None)
Start Qt Designer application to allow editing UI files.
```

```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.

```python
>>> [toBool(x) for x in range(-2, 2)]
[True, True, False, True]
```

```python
>>> [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.updateNodeFromParameterEditWidgets(parameterEditWidgets, parameterNode)
Update vtkMRMLScriptedModuleNode from widgets.

The function is useful for implementing updateParameterNodeFromGUI.

Note: Only a few widget classes are supported now. More will be added later. Report any missing classes at discourse.slicer.org.

See example in addParameterEditWidgetConnections() documentation.

slicer.util.updateParameterEditWidgetsFromNode(parameterEditWidgets, parameterNode)
Update widgets from values stored in a vtkMRMLScriptedModuleNode.

The function is useful for implementing updateGUIFromParameterNode.

Note: Only a few widget classes are supported now. More will be added later. Report any missing classes at discourse.slicer.org.

See example in addParameterEditWidgetConnections() documentation.

slicer.util.updateTableFromArray(tableNode, 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:

```python
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.

11.1.4 teem module
11.1.5 vtkAddon module
11.1.6 vtkITK module
12.1 Introduction

Medical Reality Modeling Language (MRML) is a data model developed to represent all data sets that may be used in medical software applications.

- MRML software library: An open-source software library implements MRML data in-memory representation, reading/writing files, visualization, processing framework, and GUI widgets for viewing and editing. The library is based on VTK toolkit, uses ITK for reading/writing some file format, and has a few additional optional dependencies, such as Qt for GUI widgets. The library kept fully independent from Slicer and so it can be used in any other medical applications, but Slicer is the only major application that uses it and therefore MRML library source code is maintained in Slicer’s source code repository.

- MRML file: When an MRML data is saved to file then an XML document is created (with .mrml file extension), which contains an index of all data sets and it may refer to other data files for bulk data storage. A variant of this file format is the MRML bundle file, which contains the .mrml file and all referenced data files in a single zip file (with .mrb extension).

12.2 MRML Scene

- All data is stored in a MRML scene, which contains a list of MRML nodes.

- Each MRML node has a unique ID in the scene, has a name, custom attributes (key:value pairs), and a number of additional properties to store information specific to its data type. Node types include image volume, surface mesh, point set, transformation, etc.

- Nodes can keep references (links) to each other.

- Nodes can invoke events when their contents or internal state change. Most common event is “Modified” event, which is invoked whenever the node content is changed. Other nodes, application logic objects, or user interface widgets may add observers, which are callback functions that are executed whenever the corresponding event is invoked.
12.3 MRML nodes

12.3.1 Basic MRML node types

- **Data nodes** store basic properties of a data set. Since the same data set can be displayed in different ways (even within the same application, you may want to show the same data set differently in each view), display properties are not stored in the data node. Similarly, the same data set can be stored in various file formats, therefore file storage properties are not stored in a data node. Data nodes are typically thin wrappers over VTK objects, such as vtkPolyData, vtkImageData, vtkTable. Most important Slicer core data nodes are the followings:

  - **Volume** (vtkMRMLVolume and its subclasses): stores a 3D image. Each voxel of a volume may be a scalar (to store images with continuous grayscale values, such as a CT image), label (to store discrete labels, such as a segmentation result), vector (for storing displacement fields or RGB color images), or tensor (MRI diffusion images). 2D image volumes are represented as single-slice 3D volumes. 4D volumes are stored in sequence nodes (vtkMRMLSequenceNode).

  - **Model** (vtkMRMLModelNode): stores a surface mesh (polygonal elements, points, lines, etc.) or volumetric mesh (tetrahedral, wedge elements, unstructured grid, etc.)

  - **Segmentation** (vtkMRMLSegmentationNode): complex data node that can store image segmentation (also known as contouring, labeling). It can store multiple representations internally, for example it can store both binary labelmap image and closed surface mesh.

  - **Markups** (vtkMRMLMarkupsNode and subclasses): store simple geometrical objects, such as point sets, lines, angles, curves, planes for annotation and measurements. Points are called “fiducials” in Slicer terminology as they are individually identified, named points, similar to “fiducial markers” that are used for identifying points in radiology images. Annotations module is the old generation of markups functionality and is being phased out.

  - **Transform** (vtkMRMLTransformNode): stores geometrical transformation that can be applied to any transformable nodes. A transformation can contain any number of linear or non-linear (warping) transforms chained together. In general, it is recommended to use vtkMRMLTransformNode. Child types (vtkMRMLLinearTransformNode, vtkMRMLBSplineTransformNode, vtkMRMLGridTransformNode) are kept for backward compatibility and to allow filtering for specific transformation types in user interface widgets.

  - **Text** (vtkMRMLTextNode): stores text data, such as configuration files, descriptive text, etc.

  - **Table** (vtkMRMLTableNode): stores tabular data (multiple scalar or vector arrays), used mainly for showing quantitative results in tables and plots

- **Display nodes** (vtkMRMLDisplayNode and its subclasses) specify properties how to display data nodes. For example, a model node’s color is stored in a display node associated with a model node.

  - Multiple display nodes may be added for a single data, each specifying different display properties and view nodes. Built-in Slicer modules typically only show and allow editing of the first display node associated with a data node.

  - If a display node specifies a list of view nodes then the associated data node is only displayed in those views.

  - Display nodes may refer to **color nodes** to specify a list of colors or color look-up-tables.

  - When a data node is created then default display node can be added by calling its `CreateDefaultDisplayNodes()` method. In some cases, Slicer detects if the display and storage node is missing and tries to create a default nodes, but the developers should not rely on this error-recovery mechanism.
• **Storage nodes** (vtkMRMLStorageNode and its subclasses) specify how to store a data node in file. It can store one or more file name, compression options, coordinate system information, etc.
  – Default storage node may be created for a data node by calling its `CreateDefaultStorageNode()` method.

• **View nodes** (vtkMRMLAbstractViewNode and subclasses) specify view layout and appearance of views, such as background color. Additional nodes related to view nodes include:
  – vtkMRMLCameraNode stores properties of camera of a 3D view.
  – vtkMRMLClipModelsNode defines how to clip models with slice planes.
  – vtkMRMLCrosshairNode stores position and display properties of view crosshair (that is positioned by holding down Shift key while moving the mouse in slice or 3D views) and also provides the current mouse pointer position at any time.
  – vtkMRMLLayoutNode defines the current view layout: what views (slice, 3D, table, etc.) are display and where. In addition to switching between built-in view layouts, custom view layouts can be specified using an XML description.
  – vtkMRMLInteractionNode specifies interaction mode of viewers (view/transform, window/level, place markups), such as what happens the user clicks in a view
  – vtkMRMLSelectionNode stores global state information of the scene, such as active markup (that is being placed), units (length, time, etc.) used in the scene, etc

• **Plot nodes** specify how to display table node contents as plots. Plot series node specifies a data series using one or two columns of a table node. Plot chart node specifies which series to plot and how. Plot view node specifies which plot chart to show in a view and how user can interact with it.

• **Subject hierarchy node** (vtkMRMLSubjectHierarchyNode) allows organization of data nodes into folders. Subject hierarchy folders may be associated with display nodes, which can be used to override display properties of all children in that folder. It replaces all previous hierarchy management methods, such as model or annotation hierarchies.

• **Sequence node** stores a list of data nodes to represent time sequences or other multidimensional data sets in the scene. Sequence browser node specifies which one of the internal data node should be copied to the scene so that it can be displayed or edited. The node that represents a node of the internal scene is called a proxy node. When a proxy node is modified in the scene, all changes can be saved into the internal scene.

Detailed documentation of MRML API can be found in [here](#).

### 12.3.2 MRML node attributes

MRML nodes can store custom attributes as (attribute name and value) pairs, which allow storing additional application-specific information in nodes without the need to create new node types.

To avoid name clashes custom modules that adds attributes to nodes should prefix the attribute name with the module’s name and the ‘.’ character. Example: the DoseVolumeHistogram module can use attribute names such as DoseVolumeHistogram.StructureSetName, DoseVolumeHistogram.Unit, DoseVolumeHistogram.LineStyle.

MRML nodes attributes can be also used filter criteria in MRML node selector widgets in the user interface.

### 12.3.3 MRML Node References

MRML nodes can reference and observe other MRML nodes using the node reference API. A node may reference multiple nodes, each performing a distinct role, and each addressed by a unique string. The same role name can be
used to reference multiple nodes.

Node references are used for example for linking data nodes to display and storage nodes and modules can add more node references without changing the referring or referred node.

For more details, see this page.

12.3.4 MRML Events and Observers

- Changes in MRML scene and individual nodes propagate to other observing nodes, GUI and Logic objects via VTK events and command-observer mechanism.
- `vtkSetMacro()` automatically invokes ModifiedEvent. Additional events can be invoked using `InvokeEvent()` method.
- Using `AddObserver()`/`RemoveObserver()` methods is tedious and error-prone, therefore it is recommended to use EventBroker and vtkObserverManager helper class, macros, and callback methods instead.
  - MRML observer macros are defined in Libs/MRML/vtkMRMLNode.h
  - `vtkSetMRMLObjectMacro` - registers MRML node with another vtk object (another MRML node, Logic or GUI). No observers added.
  - `vtkSetAndObserveMRMLObjectMacro` - registers MRML node and adds an observer for `vtkCommand::ModifyEvent`.
  - `vtkSetAndObserveMRMLObjectEventsMacro` - registers MRML node and adds an observer for a specified set of events.
  - `SetAndObserveMRMLScene()` and `SetAndObserveMRMLSceneEvents()` methods are used in GUI and Logic to observe Modify, NewScene, NodeAdded, etc. events.
  - `ProcessMRMLEvents()` method should be implemented in MRML nodes, Logic, and GUI classes in order to process events from the observed nodes.

12.4 Advanced topics

12.4.1 Scene undo/redo

MRML Scene provides Undo/Redo mechanism that restores a previous state of the scene and individual nodes. By default, undo/redo is disabled and not displayed on the user interface, as it increased memory usage and was not tested thoroughly.

Basic mechanism:

- Undo/redo is based on saving and restoring the state of MRML nodes in the Scene.
- MRML scene can save snapshot of all nodes into a special Undo and Redo stacks.
- The Undo and Redo stacks store copies of nodes that have changed from the previous snapshot. The node that have not changed are stored by a reference (pointer).
- When an Undo is called on the scene, the current state of Undo stack is copied into the current scene and also into Redo stack.
- All Undoable operations must store their data as MRML nodes
Developer controls at what point the snapshot is saved by calling `SaveStateForUndo()` method on the MRML scene. `SaveStateForUndo()` saves the state of all nodes in the scene. It should be called in GUI/Logic classes before changing the state of MRML nodes. This is usually done in the `ProcessGUIEvents` method that processes events from the user interactions with GUI widgets. `SaveStateForUndo()` should not be called while processing transient events such as continuous events sent by KW UI while dragging a slider (for example `vtkKWScale::ScaleValueChangedEvent`).

The following methods on the MRML scene are used to manage Undo/Redo stacks:

- `vtkMRMLScene::Undo()` restores the previously saved state of the MRML scene.
- `vtkMRMLScene::Redo()` restores the previously undone state of the MRML scene.
- `vtkMRMLScene::SetUndoOff()` ignores following `SaveStateForUndo` calls (useful when making multiple changes to the scene/nodes that does not need to be undone).
- `vtkMRMLScene::SetUndoOn()` enables following `SaveStateForUndo` calls.
- `vtkMRMLScene::ClearUndoStack()` clears the undo history.
- `vtkMRMLScene::ClearRedoStack()` clears the redo history.

### 12.4.2 Creating Custom MRML Node Classes

- Custom MRML nodes provide persistent storage for the module parameters.
- Custom MRML nodes should be registered with the MRML scene using `RegisterNodeClass()` so they can be saved and restored from a scene file.
- Classes should implement the following methods:
  - `CreateNodeInstance()` similar to VTK `New()` method only not static.
  - `GetNodeTagName()` returns a unique XML tag for this node.
  - `ReadXMLAttributes()` reads node attributes from XML file as name-value pairs.
  - `WriteXML()` writes node attributes to output stream.
  - `Copy()` – copies node attributes.
12.4.3 Slice view pipeline

Another view of VTK/MRML pipeline for the 2D slice views.

Notes: the MapToWindowLevelColors has no lookup table set, so it maps the scalar volume data to 0,255 with no “color” operation. This is controlled by the Window/Level settings of the volume display node. The MapToColors applies the current lookup table to go from 0-255 to full RGBA.
Modules usually interact with each other only indirectly, by making changes and observing changes in the MRML scene. However, modules can also directly use functions offered by other modules, by calling its logic functions. Examples and notes for using specific modules are provided below.

## 13.1 Markups

### 13.1.1 How Tos

**How to change color of a markups node?**

Markups have `Color` and `SelectedColor` properties. `SelectedColor` is used if all control points are in “selected” state, which is the default. So, in most cases `SetSelectedColor` method is used to set markups node color.

## 13.2 Volume rendering

### 13.2.1 Key classes

- `vtkMRMLVolumeRenderingDisplayNode` controls the volume rendering properties. Each volume rendering technique has its own subclass.
- `vtkSlicerVolumeRenderingLogic` contains utility functions
- `vtkMRMLScalarVolumeNode` contains the volume itself
- `vtkMRMLVolumePropertyNode` points to the transfer functions
- `vtkMRMLAnnotationROINode` controls the clipping planes
- `vtkMRMLVolumeRenderingDisplayableManager` responsible for adding VTK actors to the renderers
### 13.2.2 Format of Volume Property (.vp) file

Volume properties, separated by newline characters.

Example:

```
1 => interpolation type
1 => shading enabled
0.9 => diffuse reflection
0.1 => ambient reflection
0.2 => specular reflection
10 => specular reflection power
14 -3024 0 45.3791 0.169643 139.919 0.589286 347.907 0.607143 1224.16 0.616071 => scalar opacity transfer function (total number of values, each control point is defined by a pair of values: intensity and opacity)
4 0 1 255 1 => gradient opacity transfer function (total number of values, each control point is defined by a pair of values: intensity gradient and opacity)
28 -3024 0 0 0 45.3791 0 0.25098 1 45.3791 1 0 0 139.919 1 0.894893 0.894893 347.907 1 1 0.25098 1224.16 1 1 3071 0.827451 0.658824 1 => color transfer function (total number of values, each control point is defined by 4 of values: intensity and R, G, B color components)
```

### 13.2.3 How Tos

#### How to programmatically volume render your volume node

**C++**

```c++
qSlicerAbstractCoreModule* volumeRenderingModule = qSlicerCoreApplication::application()->moduleManager()->module("VolumeRendering");
vtkSlicerVolumeRenderingLogic* volumeRenderingLogic = volumeRenderingModule ? vtkSlicerVolumeRenderingLogic::SafeDownCast(volumeRenderingLogic->logic()) : 0;
vtkMRMLVolumeNode* volumeNode = mrmlScene->GetNodeByID('vtkMRMLScalarVolumeNode1');
if (volumeRenderingLogic)
{
  vtkSmartPointer<vtkMRMLVolumeRenderingDisplayNode> displayNode = vtkSmartPointer<vtkMRMLVolumeRenderingDisplayNode>::Take(volumeRenderingLogic->CreateVolumeRenderingDisplayNode());
mrmlScene->AddNode(displayNode);
  volumeNode->AddAndObserveDisplayNodeID(displayNode->GetID());
  volumeRenderingLogic->UpdateDisplayNodeFromVolumeNode(displayNode, volumeNode);
}
```

**Python**

```python
logic = slicer.modules.volumerendering.logic()
volumeNode = slicer.mrmlScene.GetNodeByID('vtkMRMLScalarVolumeNode1')
displayNode = logic.CreateVolumeRenderingDisplayNode()
displayNode.UnRegister(logic)
slicer.mrmlScene.AddNode(displayNode)
volumeNode.AddAndObserveDisplayNodeID(displayNode.GetID())
logic.UpdateDisplayNodeFromVolumeNode(displayNode, volumeNode)
```
How to programmatically apply a custom color/opacity transfer function

**C++**

```cpp
vtkColorTransferFunction* colors = ...  
vtkPiecewiseFunction* opacities = ...  
vtkMRMLVolumeRenderingDisplayNode* displayNode = ...  
vtkMRMLVolumePropertyNode* propertyNode = displayNode->GetVolumePropertyNode();  
propertyNode->SetColor(colorTransferFunction);  
propertyNode->SetScalarOpacity(opacities);  
// optionally set the gradients opacities with SetGradientOpacity
```

The logic has utility functions to help you create those transfer functions:

- `volumeRenderingLogic->SetWindowLevelToVolumeProp(...)`
- `volumeRenderingLogic->SetThresholdToVolumeProp(...)`
- `volumeRenderingLogic->SetLabelMapToVolumeProp(...)`

**Python**

```python
propertyNode = displayNode.GetVolumePropertyNode()
...
```

How to programmatically limit volume rendering to a subset of the volume

**C++**

```cpp
vtkMRMLAnnotationROINode* roiNode = ...  
vtkMRMLVolumeRenderingDisplayNode* displayNode = ...  
displayNode->SetAndObserveROINodeID(roiNode->GetID());  
displayNode->SetCroppingEnabled(1);
```

**Python**

```python
displayNode.SetAndObserveROINodeID(roiNode.GetID())
displayNode.CroppingEnabled = 1
```

How to register a new Volume Rendering mapper

You need to derive from `vtkMRMLVolumeRenderingDisplayNode` and register your class within `vtkSlicerVolumeRenderingLogic`.

```cpp
void qSlicerMyABCVolumeRenderingModule::setup()
{
  vtkMRMLThreeDViewDisplayableManagerFactory::GetInstance()->
    RegisterDisplayableManager("vtkMRMLMyABCVolumeRenderingDisplayableManager");
  this->Superclass::setup();
  qSlicerAbstractCoreModule* volumeRenderingModule = qSlicerCoreApplication::application()->moduleManager()->module("VolumeRendering");
  if (volumeRenderingModule)
  {
```

(continues on next page)
If you want to expose control widgets for your volume rendering method, then register your widget with addRenderingMethodWidget().

How to register custom volume rendering presets

Custom presets can be added to the volume rendering module by calling AddPreset() method of the volume rendering module logic. The example below shows how to define multiple custom volume rendering presets in an external MRML scene file and add them to the volume rendering module user interface.

Create a MyPresets.mrml file that describes two custom volume rendering presets:

```xml
<MRML version="Slicer4.4.0">
  <VolumeProperty id="vtkMRMLVolumeProperty1" name="MyPreset1" references="IconVolume: vtkMRMLVectorVolumeNode1;" interpolation="1" shade="1" diffuse="0.66" ambient="0.1" specular="0.62" specularPower="14" scalarOpacity="10 -3.52844023704529 0 56.7852325439453 0 79.255027709961 0.428571432828903 415.119384765625 1 641 1" gradientOpacity="4 0 1 160.25 1" colorTransfer="16 0 0 0 0.945098039215686 0.956862745098039 412.406 0 0.592157 0.807843 641 1 1 1" />
  <VectorVolume id="vtkMRMLVectorVolumeNode1" references="storage: vtkMRMLVolumeArchetypeStorageNode1;" />
  <VolumeArchetypeStorage id="vtkMRMLVolumeArchetypeStorageNode1" fileName="MyPreset1.png" fileListMember0="MyPreset1.png" />

  <VolumeProperty id="vtkMRMLVolumeProperty2" name="MyPreset2" references="IconVolume: vtkMRMLVectorVolumeNode2;" interpolation="1" shade="1" diffuse="0.66" ambient="0.1" specular="0.62" specularPower="14" scalarOpacity="10 -3.52844023704529 0 56.7852325439453 0 79.255027709961 0.428571432828903 415.119384765625 1 641 1" gradientOpacity="4 0 1 160.25 1" colorTransfer="16 0 0 0 0.945098039215686 0.956862745098039 412.406 0 0.592157 0.807843 641 1 1 1" />
  <VectorVolume id="vtkMRMLVectorVolumeNode2" references="storage: vtkMRMLVolumeArchetypeStorageNode2;" />
  <VolumeArchetypeStorage id="vtkMRMLVolumeArchetypeStorageNode2" fileName="MyPreset2.png" fileListMember0="MyPreset2.png" />
</MRML>
```

For this example, thumbnail images for the presets should be located in the same directory as MyPresets.mrml, with the file names MyPreset1.png and MyPreset2.png.

Use the following code to read all the custom presets from MyPresets.mrml and load it into the scene:

```c++
presetsScenePath = "MyPresets.mrml"
```
13.3 Volumes

13.3.1 How to create a volume node from scratch?

You need to create a vtkImageData, a vtkMRMLScalarVolumeNode and a vtkMRMLScalarVolumeDisplayNode.

In C++:

```c++
vtkNew<vtkImageData> imageData;
imageData->SetDimensions(10,10,10); // image size
imageData->AllocateScalars(VTK_UNSIGNED_CHAR, 1); // image type and number of components
// initialize the pixels here

vtkNew<vtkMRMLScalarVolumeNode> volumeNode;
volumeNode->SetAndObserveImageData(imageData);
volumeNode->SetOrigin(-10., -10., -10.);
volumeNode->SetSpacing(2., 2., 2.);
mrmlScene->AddNode(volumeNode.GetPointer());

volumeNode->CreateDefaultDisplayNodes()
```

In Python:

```python
imageData = vtk.vtkImageData()
imageData.SetDimensions(10,10,10) # image size
imageData.AllocateScalars(vtk.VTK_UNSIGNED_CHAR, 1) # image type and number of components
# initialize the pixels here
volumeNode = slicer.vtkMRMLScalarVolumeNode()
volumeNode.SetAndObserveImageData(imageData)
volumeNode = slicer.mrmlScene.AddNode(volumeNode)
volumeNode.CreateDefaultDisplayNodes()
```

Note that the origin and spacing must be set on the volume node instead of the image data. The Image Maker extension module contains a module that allows creating a volume from scratch without programming.
### 13.3.2 Reading from file

The following options can be passed to load volumes programmatically when using `qSlicerVolumesReader`:

- **name** (string): Node name to set for the loaded volume
- **labelmap** (bool, default=false): Load the file as labelmap volume
- **singleFile** (bool, default=false): Force loading this file only (otherwise the loader may look for similar files in the same folder to load multiple slices as a 3D volume)
- **autoWindowLevel** (bool, default=true): Automatically compute the window level based on the volume pixel intensities
- **show** (bool, default=true): Show the volume in views after loading
- **center** (bool, default=false): Apply a transform that places the volume in the patient coordinate system origin
- **discardOrientation** (bool, default=false): Discard file orientation information.
- **fileNames** (string list): List of files to be loaded as a volume
14 Extensions

14.1 Overview

An extension could be seen as a delivery package bundling together one or more Slicer modules. After installing an extension, the associated modules will be presented to the user the same way as built-in modules.

The Slicer community maintains a website referred to as the Slicer Extensions Manager (also known as Slicer App Store or Slicer Extensions Index) to allow users to find, download, and install of extensions.
14.2 How to create an extension?

If you have developed a script or module that you would like to share with others then it is recommended to add it to an extension to distribute it.

- Scan through the user and developer extension FAQs
- Inform a community about your plans on the [Slicer forum](https://discourse.slicer.org) to get information about potential parallel efforts (other developers may already work on a similar idea and you could join or build on each other’s work), past efforts (related tools might have been available in earlier Slicer versions or in other software that you may reuse), and get early feedback from prospective users. You may also seek advice on the name of your extension and how to organize features into modules. All these can save you a lot of time in the long term.
- If developing C++ loadable or CLI modules (not needed if developing in Python): build Slicer application in “Release” mode.
- Use the Extension Wizard module in Slicer to create an extension that will contain your module(s). To learn about extension description file format see here.
- Upload source code of your extension to a publicly available repository. It is recommended to start the repository name with “Slicer” (to make Slicer extensions easier to identify) followed by your extension name (for example, “Sequences” extension is stored in “SlicerSequences” repository). However, this is not a mandatory requirement. If you have a compelling reason not to use Slicer prefix, please make a note while making the pull request. See more requirements in the new extension submission checklist.
  - GitHub is recommended (due to large user community, free public project hosting): join Github and setup Git.
- If developing C++ loadable or CLI modules (not needed if developing in Python): Build your extension
- Test your extension:
  - If you have built your extension then build the PACKAGE target to create a package file that you can install from the Extension Manager by clicking the small tool icon in the top-right corner.
  - If you have not built your extension then set up your extension manually: Build your extension

14.3 Documentation

Keep documentation with your extension’s source code and keep it up-to-date whenever the software changes.

Add at least a README.md file in the root of the source code repository, which describes what the extension is for and how it works. Minimum information that is needed to make your extension usable is described in the extension submission checklist.

Extension documentation examples:
- SegmentMesher
- SequenceRegistration
- AI-assisted annotation client
- SlicerDMRI - large extension documented using Github pages

CLI module documentation can be automatically generated in mediawiki format using this script.

Thumbnails to YouTube videos can be generated by downloading the image from [https://img.youtube.com/vi/your-youtube-video-id/0.jpg](https://img.youtube.com/vi/your-youtube-video-id/0.jpg) and adding a playback button using [http://addplaybuttontoimage.way4info.net/](http://addplaybuttontoimage.way4info.net/) (the second red arrow is recommended).
14.4 Distributing an extension

If your extension is ready for distribution (you have completed extension submission checklist) then submit it to the Slicer Extensions Index:

- Fork ExtensionIndex repository on GitHub by clicking “Fork” button on https://github.com/Slicer/ExtensionsIndex page
- Add your .s4ext file to your forked repository: it can be done using a git client or simply by clicking “Upload files” button
- Create a pull request: by clicking “Create pull request” button

14.5 Continuous Integration

If you shared your extension by using the ExtensionWizard, make sure you know about the Slicer testing dashboard:

The dashboard will attempt to check out the source code of your extension, build, test and package it on Linux, MacOSX and Windows platforms.

To find your extension, use the following link replacing SlicerMyExtension with the name of your extension:

http://slicer.cdash.org/index.php?project=Slicer4&filtercount=1&showfilters=1&field1=buildname&compare1=63&value1=SlicerMyExtension

For example, here is the link to check the status of the SlicerDMRI extension:

http://slicer.cdash.org/index.php?project=Slicer4&filtercount=1&showfilters=1&field1=buildname&compare1=63&value1=SlicerDMRI

If you see red in any of the columns for your extension, click on the hyperlinked number of errors to see the details.

Always check the dashboard after you first introduce your extension, or after you make any changes to the code.
Frequently asked questions about how to write Python scripts for Slicer.

15.1 How to find a Python function for any Slicer features

All features of Slicer are available via Python scripts. Slicer script repository contains examples for the most commonly used features.

To find out what Python commands correspond to a feature that is visible on the graphical user interface, search in Slicer’s source code where that text occurs, find the corresponding widget or action name, then search for that widget or action name in the source code to find out what commands it triggers.

Complete example: How to emulate selection of FOV, spacing match Volumes checkbox in the slice view controller menu?

- Go to Slicer project repository on github
- Enter text that you see on the GUI near the function that you want to use. In this case, enter "FOV, spacing match Volumes" (adding quotes around the text makes sure it finds that exact text)
- Usually the text is found in a .ui file, in this case it is in qMRMLSliceControllerWidget.ui, open the file
- Find the text in the page, and look up what is the name of the widget or action that it is associated with - in this case it is an action named actionSliceModelModeVolumes
- Search for that widget or action name in the repository, you should find a source file(s) that use it. In this case it will is qMRMLSliceControllerWidget.cxx
- Search for the action/widget name, and you’ll find what it does - in this case it calls setSliceModelModeVolumes method, which calls this->setSliceModelMode(vtkMRMLSliceNode::SliceResolutionMatchVolumes), which then calls d->MRMLSliceNode->SetSliceResolutionMode (mode)
- This means that this action calls someSliceNode->SetSliceResolutionMode(vtkMRMLSliceNode::SliceResolutionMatchVolumes) in Python syntax it is someSliceNode.SetSliceResolutionMode(slicer.
 For example, for the red slice node this will be:

```python
sliceNode = slicer.mrmlScene.GetNodeByID('vtkMRMLSliceNodeRed')
sliceNode.SetSliceResolutionMode(slicer.vtkMRMLSliceNode.SliceResolutionMatchVolumes)
```
CHAPTER 16

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.

16.1 The PR Process, Circle CI, and Related Gotchas

16.1.1 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 `Utilities/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.
16.1.2 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.

16.1.3 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

### 16.1.4 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. **LGT**M 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.

### 16.1.5 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 blog post.

### 16.1.6 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.

### 16.1.7 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.
The initial version of these guidelines was established during the winter project week 2017.

16.1.8 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.
Credits

Please see the GitHub project page at https://github.com/Slicer/Slicer/graphs/contributors
CHAPTER 18

Indices and tables

- genindex
- modindex
- search
S
slicer.118
slicer.cli.99
slicer.testing.100
slicer.util.100
addObserver() *(slicer.util.VTKObservationMixin method)*, 100
addParameterEditWidgetConnections() *(in module slicer.util)*, 100
addVolumeFromArray() *(in module slicer.util)*, 101
array() *(in module slicer.util)*, 102
arrayFromGridTransform() *(in module slicer.util)*, 102
arrayFromGridTransformModified() *(in module slicer.util)*, 102
arrayFromMarkupsControlPoints() *(in module slicer.util)*, 102
arrayFromMarkupsCurvePoints() *(in module slicer.util)*, 102
arrayFromModelPointData() *(in module slicer.util)*, 102
arrayFromModelPointDataModified() *(in module slicer.util)*, 102
arrayFromModelPoints() *(in module slicer.util)*, 102
arrayFromModelPointsModified() *(in module slicer.util)*, 103
arrayFromModelPolyIds() *(in module slicer.util)*, 103
arrayFromSegment() *(in module slicer.util)*, 103
arrayFromSegmentBinaryLabelmap() *(in module slicer.util)*, 103
arrayFromSegmentInternalBinaryLabelmap() *(in module slicer.util)*, 103
arrayFromTableColumn() *(in module slicer.util)*, 104
arrayFromTableColumnModified() *(in module slicer.util)*, 104
arrayFromTransformMatrix() *(in module slicer.util)*, 104
arrayFromVolume() *(in module slicer.util)*, 104
arrayFromVolumeModified() *(in module slicer.util)*, 104
arrayFromVTKMatrix() *(in module slicer.util)*, 104
cancel() *(in module slicer.cli)*, 99
choice() *(in module slicer.util)*, 104
downloadAndExtractArchive() *(in module slicer.util)*, 106
downloadFile() *(in module slicer.util)*, 106
downloadFile() *(in module slicer.util)*, 106
downloadAndExtractArchive() *(in module slicer.util)*, 106
downloadFile() *(in module slicer.util)*, 106
pythonShell() (in module slicer.util), 114

Q
quilt() (in module slicer.util), 114

R
reloadScriptedModule() (in module slicer.util), 114
removeObserver() (slicer.util.VTKObservationMixin method), 100
removeObservers() (slicer.util.VTKObservationMixin method), 100
removeParameterEditWidgetConnections() (in module slicer.util), 114
resetSliceViews() (in module slicer.util), 114
resetThreeDViews() (in module slicer.util), 114
restart() (in module slicer.util), 115
run() (in module slicer.cli), 99
runSync() (in module slicer.cli), 99
runUnitTest() (in module slicer.testing), 100

S
saveNode() (in module slicer.util), 115
saveScene() (in module slicer.util), 115
selectedModule() (in module slicer.util), 115
selectModule() (in module slicer.util), 115
setApplicationLogoVisible() (in module slicer.util), 115
setDataProbeVisible() (in module slicer.util), 115
setMenuBarsVisible() (in module slicer.util), 115
setModuleHelpSectionVisible() (in module slicer.util), 115
setModulePanelTitleVisible() (in module slicer.util), 115
setNodeParameters() (in module slicer.cli), 100
setPythonConsoleVisible() (in module slicer.util), 115
setSliceViewerLayers() (in module slicer.util), 115
setStatusBarVisible() (in module slicer.util), 116
settingsValue() (in module slicer.util), 116
setToolbarsVisible() (in module slicer.util), 116
setViewControllersVisible() (in module slicer.util), 116
showStatusMessage() (in module slicer.util), 116
slicer (module), 118
slicer.cli (module), 99
slicer.testing (module), 100
slicer.util (module), 100
sourceDir() (in module slicer.util), 116
startQtDesigner() (in module slicer.util), 116

T
tempDirectory() (in module slicer.util), 116
TESTING_DATA_URL (in module slicer.util), 100
toBool() (in module slicer.util), 116
toLatin1String() (in module slicer.util), 117
toVTKString() (in module slicer.util), 117

U
updateMarkupsControlPointsFromArray() (in module slicer.util), 117
updateNodeFromParameterEditWidgets() (in module slicer.util), 117
updateParameterEditWidgetsFromNode() (in module slicer.util), 117
updateTableFromArray() (in module slicer.util), 117
updateTransformMatrixFromArray() (in module slicer.util), 117
updateVTKMatrixFromArray() (in module slicer.util), 117

V
vtkMatrixFromArray() (in module slicer.util), 118
VTKObservationMixin (class in slicer.util), 100

W
warningDisplay() (in module slicer.util), 118